## Document Information

| | |
|---|---|
| **Analyzed document** | OOPS with C++.pdf (D166063612) |
| **Submitted** | 5/6/2023 6:52:00 AM |
| **Submitted by** | Mumtaz B |
| **Submitter email** | mumtaz@code.dbuniversity.ac.in |
| **Similarity** | 20% |
| **Analysis address** | mumtaz.dbuni@analysis.urkund.com |

## Sources included in the report

| | | |
|---|---|---|
| W | URL: https://www.msuniv.ac.in/Download/Pdf/a6241a9e41024aa<br>Fetched: 6/13/2022 12:18:00 PM | 4 |
| SA | **DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf**<br>Document DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140) | 9 |
| SA | **120E1240_ Object Oriented Programming Using C++.doc**<br>Document 120E1240_ Object Oriented Programming Using C++.doc (D165245825) | 37 |
| W | URL: http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAM...<br>Fetched: 12/18/2022 5:30:46 AM | 75 |
| W | URL: https://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf<br>Fetched: 11/5/2021 12:07:08 PM | 10 |
| SA | **248E1110-Object Oriented Programing using C++(Id 2732).doc**<br>Document 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029) | 23 |
| SA | **C++ From The Ground Up_ 3rd Edition (2003).pdf**<br>Document C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784) | 19 |
| SA | **248E1160-Lab-1_Object Oriented Programming.doc**<br>Document 248E1160-Lab-1_Object Oriented Programming.doc (D165247743) | 1 |
| SA | **ODL Learning Materials (ALL 5 UNITS).pdf**<br>Document ODL Learning Materials (ALL 5 UNITS).pdf (D109014230) | 14 |
| W | URL: https://www.vidyarthiplus.com/vp/attachment.php?aid=46806<br>Fetched: 11/13/2021 11:48:10 AM | 6 |
| SA | **Object Oriented Programming through C++ Block 1.pdf**<br>Document Object Oriented Programming through C++ Block 1.pdf (D164970258) | 10 |
| SA | **137E1240-Object Oriented Programming using C++_120E1240.docx**<br>Document 137E1240-Object Oriented Programming using C++_120E1240.docx (D165245896) | 9 |
| W | URL: https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf<br>Fetched: 6/28/2022 11:36:59 AM | 20 |

| | | |
|---|---|---|
| SA | **INF_1016.pdf**<br>Document INF_1016.pdf (D164968061) | 6 |
| SA | **010E2340-Programming in C and C++.pdf**<br>Document 010E2340-Programming in C and C++.pdf (D165445451) | 7 |
| SA | **ECAP 444.docx**<br>Document ECAP 444.docx (D142426097) | 34 |
| W | URL: http://apsacollege.com/wp-content/uploads/2014/09/CPP_4BIT3C1.pdf<br>Fetched: 8/28/2021 7:10:58 AM | 6 |
| SA | **OOP through C++ (Block 2).pdf**<br>Document OOP through C++ (Block 2).pdf (D148964031) | 10 |
| SA | **odl C++ lecture notes unit-5.docx**<br>Document odl C++ lecture notes unit-5.docx (D109013221) | 3 |
| SA | **ODMCA-102_T_Intro_to_Programming_Section_D_25th_Oct.docx**<br>Document ODMCA-102_T_Intro_to_Programming_Section_D_25th_Oct.docx (D43197291) | 1 |

## Entire Document

Basic Concepts of OOPS 1 Notes Unit 1: Basic Concepts of OOPS Structure 1.1 Introduction 1.2 Object Oriented Paradigms and Metaphors 1.3 Distinction between Procedural and oops

55% **MATCHING BLOCK 1/304** W

programming 1.4 Basic Concepts of Object-oriented Programming 1.5 Benefits of OOP 1.5.1 Others are extended conventional languages 1.5.2 Applications of OOP 1.6

C++ 1.6.1 Invoking Turbo C++ 1.6.2 Naming Your Program 1.6.3 Using the Editor 1.6.4 Saving Your Programs 1.6.5 Compiling and Linking 1.6.6 Running the Program 1.6.7 Exiting from the IDE 1.6.8 Opening an existing file 1.6.9 A Simple C++ Program 1.6.10 Output 1.7 Summary 1.8 Check Your Progress 1.9 Questions and Exercises 1.10 Key Terms 1.11

89% **MATCHING BLOCK 2/304** SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

Further Readings Objectives After studying this unit, you should be able to: Understand the

object oriented
paradigms and metaphors. Discuss the basic concepts of oops. Explain the difference between oops and pop. 1.1 Introduction Software system development is basically a modeling exercise where the user requirements are deciphered into a working software model. The translation goes through various stages including sub-activities, every stage refining the initial model given to it. Precisely how the system is seen, analyzed, planned, tested, implemented and maintained, while under development is briefly termed as software development paradigm.

2 Object Oriented Programmimg with C++ Notes 1.2 Object Oriented Paradigms and Metaphors In conventional software development paradigm, the software system is seen as a procedure that changes the given input into desired output. Accordingly, analysis, design, testing and other development phases have been developed each having the same process view of the system. Under this paradigm, even a printer is seen as a procedure, as shown in the figure 1.1. Figure 1.1: Conventional Software Model In contrast to the conventional development approach, OO approach sees a software system as an object having remarkable characteristics and services it gives to its surrounding objects. The system is displayed as a real world object, which is made of smaller and simple objects interacting with one another in a predefined way, as shown in the figure 1.2 . Figure 1.2: Object Oriented Software Model Object oriented programming, since its inception, has progressed significantly. In the beginning, and for a many years hence, the term object oriented (OO) was use to denote a software development approach that utilized one of available object-oriented programming languages (e.g., Ada95, Java, C++ Eiffel, Smalltalk). While programming (a somewhat minor part in software system development) could be accomplished using an object-oriented language, the major development activities remained conventional. This situation could not exploit the rich features of object oriented programming optimally. In present times, however, OO approach has outgrown into a huge structure incorporating a complete field of software engineering. Only written software into an object-oriented programming language does not exploit all the benefit it offers. Object oriented approach is not only limited to programming but the entire gamut of development processes giving it a larger scope. Thus, many object-oriented techniques have gradually emerged, like Basic Concepts of OOPS 3 Notes Object Oriented Analysis (OOA) Object Oriented Design (OOD) Object Oriented Testing (OOT) Object Oriented Data Base Management Systems (OODBMS) Object Oriented Programming (OOP) Object Oriented Domain Analysis (OODA) OO approach, undoubtedly, has number of advantage over conventional software development process - simplicity of maintenance and reusability being the foremost. That - a (software) system might be developed using object oriented techniques, as indicated by the specified development framework in every phase to understand its full advantage is known to as Object Oriented Paradigm as against conventional development paradigm. Various distinctive process models for software engineering have been proposed by the researchers and employed by the developer. Although any of these models could be adopted for use with OO, the best decision would recognize that OO systems have a tendency to evolve over time. Therefore, an evolutionary process model, coupled with an approach that encourages component assembly reuse, is the best paradigm for OO software engineering. Figure 1.3, displays the component-based development process model tailored for OO software engineering. The OO process moves through an evolutionary spiral that begins with customer communication. It is here that the problem domain is defined and that basic problem classes, which models for real world objects are identified. Planning and risk analysis lays a foundation for the OO project plan. The technical work associated with OO software engineering follows the iterative path shown in the shaded box of figure 1.3. OO software engineering emphasizes reuse. Therefore, classes are "looked up" in a library (of existing OO classes) before they are built. When a class cannot be found in the library, the software engineer applies object-oriented analysis (OOA), object- oriented design (OOD), object-oriented programming (OOP), and object-oriented testing (OOT) to create the class and the objects derived from the class. The new class is then put into the library so that it may be reused in the future. Figure 1.3: Component Based Development Process The object- oriented view demands an evolutionary approach to deal with software engineering since it is becoming exceedingly hard to define all necessary classes for a major system or product in a single iteration. As the OO analysis and design models
4 Object Oriented Programmimg with C++ Notes develop, the requirement for additional classes becomes apparent. It is for the reason that the paradigm simply described works best for OO. A metaphor refers to something substantial onto which one can outline various features of the software being composed. In other words, it is a physical model that abstracts all the desired features of the software intended to be designed. One major advantage of OO paradigm is that the component objects can be developed more or less independently. The main limitation is that the interfaces of the respective objects remain intact or in case of any modification, is communicated to different developers. The component objects, therefore, can be developed simultaneously. This is the concept of concurrent object oriented systems. In this development model, the objects are developed concurrently reducing the overall development time. A schematic presentation of the same is given below. (Figure 1.4) Figure 1.4: One Element of the Concurrent Process Model All the software engineering activities - customer communication, analysis, coding implementation, testing etc. - are completed concurrently. All the activities turn undergo a cycle of development phases. The activities exist simultaneously in the development environment. The activities exist simultaneously in the development environment. However, they might be in various phases of development as showed by the diagram above. (Figure 1.4) The significant inspiring
factor in the innovation of object-oriented approach is to salvage some of the flaws encountered in the procedural approach. OOP treat

| 95% | MATCHING BLOCK 4/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

data as a critical element in the program development and does not permit it to flow freely around the system. It ties data more closely to the functions that operate on it and protects it from accidental modification from outside functions. OOP allows us to decompose a problem into a number of entities called objects and then builds data and functions around these

entries. The organization of
data and functions in object-oriented programs is shown in
Figure 1.5.
Basic Concepts of OOPS 5 Notes Figure 1.5:
Organization of

89% MATCHING BLOCK 3/304 W

Data and Functions in OOP The data of an object can be accessed only by the functions associated with that object. However, functions of one object can access the functions of other objects.

Some of
the striking

98% MATCHING BLOCK 5/304 SA 120E1240_ Object Oriented Programming Using C+ ... (D165245825)

features of object-oriented programming are: Emphasis is on data rather than procedure. Programs are divided into what are known as objects. Data structures are designed such that they characterize the objects. Functions that operate on the data of an object are tied together in the data structure. Data is hidden and cannot be accessed by external functions. Objects may communicate with each other through functions. New data and functions can be easily added whenever necessary. Follows bottom-up approach in program design. 1.3

Distinction between Procedural and oops programming Main difference between pop and oops programming is:
6 Object Oriented Programmimg with C++ Notes Table 1.1: Difference between pop and oops 1.4 Basic Concepts of Object-oriented Programming "Object-oriented" remains a term which is deciphered differently by various individuals.
It is subsequently important
understand some of the concepts used extensively

59% MATCHING BLOCK 6/304 W

in object-oriented programming. We shall discuss in this section the following general concepts: Objects Classes Data abstraction Data encapsulation Inheritance Polymorphism Dynamic binding Message passing Objects Objects are the fundamental run-time entities in an object-oriented system. They might represent a person, a place,

or anything that the program must handle. Subject of Difference Procedure Oriented Programming (POP) Object Oriented Programming (OOP) Problem decomposition Decompose the main problem in small parts called functions. Decompose the main problem in small parts called objects. Connections of parts Connects small parts of the program by passing parameters & using operating system. Connects small parts of the program by passing messages. Emphasizing Emphasizes on functions. Emphasizes on data. Use of data In large programs, most functions use global data. Each object controls data under it. Passing of data Data may get passed from one function to another. Data never get passed from one object to another. Security of data Appropriate & effective techniques are unavailable to secure the data. Data stay secured as no external function can use data of an object. Modification of program Modification of a completed program is very difficult and it may affect the whole program. Modifications are easy as objects stay independent to declare and define. Designing approach Employs top-down approach for designing programs. Employs bottom-up approach for designing. Data identification In large programs, it is very difficult to find what data has been used by which function. As data and functions stay close, it is easy to identify data. Used languages Languages like C, FORTRAN, COBOL etc. use POP. Languages like C++, JAVA etc. use OOP.
Basic Concepts of OOPS 7 Notes

96% MATCHING BLOCK 7/304 SA 120E1240_ Object Oriented Programming Using C+ ... (D165245825)

When a program is executed, the objects interact by sending messages to one another. For example, if "customer" and "account" are two objects in a program, then the customer object may send a message to the account object requesting for the bank balance. Each object

contains data and code to manipulate
the

98% MATCHING BLOCK 8/304 W

data. Objects can interact without having to know details of each other's data or code. It is sufficient to know the type of message accepted and the type of response returned by the objects.

Figure 1.6: Two Ways of Representing an

100% MATCHING BLOCK 9/304 SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

Object Class Class is a blueprint for a data type,

its

98% MATCHING BLOCK 10/304 SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword class as follows: class Box { public: double length; // Length of a box double breadth; // Breadth of a box double height; // Height of a box }; The keyword public determines the access attributes of the members of the class that follow it. A public member can be accessed from outside the class anywhere within the scope of the class object.

Data abstraction Data abstraction refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details. Data abstraction is a programming technique that depends on the separation of interface and implementation. We should take one genuine case of a TV, which you can turn on and off, change the channel, adjust the volume, and include external component, for example, speakers, VCRs, and DVD players, BUT you don't have a clue
8 Object Oriented Programmimg with C++ Notes about its inside details, that is, you don't know how it gets signals over the air or through a cable, how it translate them, and finally how it shows them on the screen. Thus, we can say a television clearly separates its internal implementation from its external interface and you can play with its interfaces like the power button, channel changer, and volume control without having zero knowledge of its internals. Data encapsulation Encapsulation is an Object Oriented Programming idea that ties together the data and functions that manipulates

91% MATCHING BLOCK 11/304 SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

the data, and that keeps both safe from outside interference and misuse.

Data encapsulation is a mechanism of packaging

75% MATCHING BLOCK 12/304 SA 248E1160-Lab-1_Object Oriented Programming.doc (D165247743)

the data, and the functions that use them and data abstraction is a system of uncovering just the interfaces and hiding the implementation details from the user.

97% MATCHING BLOCK 14/304 SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

Inheritance Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time. When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class. Polymorphism Polymorphism occurs when there is a hierarchy of classes and they are related by inheritance. C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

80% MATCHING BLOCK 13/304 W

Dynamic binding Dynamic Binding refers to linking a procedure call to the code

that will be executed only at run time. The code associated with the procedure in not known until the program is executed, which is also known as late binding. Message passing Message Passing is nothing but sending and receiving of information by the objects same as people exchange information. So this helps in building systems that simulate real life. Following are
the basic steps in message passing.

91% MATCHING BLOCK 15/304 SA 120E1240_ Object Oriented Programming Using C+ ... (D165245825)

Creating classes that define objects and its behavior. Creating objects from class definitions Establishing communication among objects

In OOPs,

| 97% | MATCHING BLOCK 16/304 | SA | ODL Learning Materials (ALL 5 UNITS).pdf (D109014230) |
|---|---|---|---|

Message Passing involves specifying the name of objects, the name of the function, and the information to be sent. 1.5 Benefits of OOP OOP offers several benefits to both the program designer and the user. Object- orientation contributes to the solution of many problems associated with the development and quality of software products. The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost. The principal advantages are: Through inheritance:

Basic Concepts of OOPS 9 Notes ?

| 93% | MATCHING BLOCK 18/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

We can eliminate redundant code. ? We can extend the use of existing classes. We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program. It is possible to have multiple instances of an object to co-exist without any interference. It is possible to map objects in the problem domain to those objects in the program. It is easy to partition the work in a project based on objects. The data-centered design approach enables us to capture more details of a model in implementable form. Object-oriented systems can be easily upgraded from small to large systems.

Message passing techniques
for communication between objects makes
the

| 100% | MATCHING BLOCK 17/304 | W |
|---|---|---|

interface descriptions with external systems much simpler. Software complexity can be easily managed. 1.5.1

Others are extended conventional languages Real-time systems Simulation and modeling Object-oriented databases Hypertext, hypermedia and expert text. AI and expert systems 1.5.2 Applications of OOP Main application areas of OOP are User interface design such as windows, menu.
Real Time Systems
Simulation and Modeling Object oriented databases AI and Expert System Neural Networks Parallel programming Decision support Office automation
system
To develop software, the object-oriented concepts need to be implemented in any high- level language. The high-level language that implements the concepts of object- oriented programming is known as an object-oriented language(also called an OO language). In general, an object-oriented language must support all or some of these OO concepts. ? Encapsulation and data hiding ? Inheritance ? Polymorphism and dynamic binding ? All built-in and user-defined data types are objects ? All operations are performed using the message passing techniques Depending on the extent to which they support OO concepts, the OO languages are classified into several categories which are listed here.
10 Object Oriented Programmimg with C++ Notes ? Pure languages: Languages that not only support but also enforce all object-oriented concepts are called pure OO languages. In these languages, everything from character and punctuations to modules is treated as an object. Smalltalk, Eiffel and Ruby are the examples of pure OO languages. ? Hybrid languages: Languages that support some (not all) of the OO concepts are called hybrid languages. Java, Python and C# are the examples of hybrid languages. ? Multi-paradigm languages: Languages that support many programming paradigms (such as procedural programming, generic programming, etc.), one of which is object-oriented paradigm are called multi-paradigm languages. c++ is the example of multi-paradigm language. ? Object-based languages: Languages that support the concept of abstract data types and also other OO concepts like encapsulation, data hiding and operator overloading are known as known as object-.based languages. However, these languages do not support the concept of inheritance and dynamic binding. Ada and Modula-2 are the examples of object-based languages. An evaluation and comparison of some of the popularly used programming languages, based on the OO concepts they support, is listed in Table A Comparision of some popular OO Language There is no fixed rule or principle based on which a particular language can be chosen for developing software. The decision of choice of language entirely depends on the characteristics and basic needs of the application to be developed, re-usability of the existing code, the impact of the organization on the choice of the programming language and various other aspects. However, C++ is the most successful and widely used general-purpose OO language. Advantages of OOP The object-oriented programming paradigm came into use as it overcomes certain limitations of other conventional programming paradigms like the structured and unstructured paradigms. The new and advanced features of OOP such as encapsulation, abstraction, inheritance, and polymorphism help in developing high- quality software. The high-quality software can be developed due to its certain advantages. Some of the advantages of OOP are listed here.

Basic Concepts of OOPS 11 Notes ? In OOP, writing programs with the help of objects is much similar to working with real-world objects. That is, the real world objects can be conveniently represented in a program which reduces the complexity of the program and also makes the program structure clear. ? In object-oriented programs, each object is an independent and separate entity which makes modifications, locating and fixing problems in a program an easy task. In addition, any changes made inside the class do not affect the other parts of a program. Thus, object-oriented programs are easy-to-write and easy-to-maintain. ? In object-oriented programming, data integrity and data security is high as it focuses on the data and its protection from manipulation by different parts of the program. As a result, object-oriented programs are less error-prone, more reliable and secure. ? Object-oriented programs are easy to extend as new features in a program can be added easily by introducing a few new objects without modifying the existing ones. ? Object-oriented programming allows re-usability of code. That is, the objects created in one program can be re-used in other programs. In addition, new classes can be created with the help of existing ones using inheritance. It leads to faster software development and high-quality programs. ? Object-oriented programs are easier to adapt and scale, that is, large system can be created by assembling re-usable subsystems. Applications of OOP Since 1960, when Simula-67 was developed, object-oriented paradigm has touched many major application areas of software development. Some of the application areas where OOP has been used to develop software are listed here. ? Simulations and Modeling: Simulation is the technique of representing the real world entities with the help of a computer program. Simula-67 and Smalltalk are two object-oriented languages are designed for making simulations. ? User-interface design: Another popular application of OOP has been in the area of designing graphical user interfaces such as Windows. C++ is mainly used for developing user-interfaces. ? Developing computer games: OOP is also used for developing computer games such as Diablo, Startcraft and Warcraft III. These games offer virtual reality environments in which a number of objects interact with each other in complex ways to give the desired result. ? Scripting: In recent years, OOP has also been used for developing HTML, XHTML and XML documents for the Internet. Python, Ruby and Java are the scripting languages based on object-oriented principles which are used for scripting. ? Object Databases: These days OOP concepts have also been introduced in database systems to develop a new DBMS named object databases. These databases store the data directly in the form of objects. However, these databases are not as popular as the traditional RDBMS. Some other areas of applications include office automation systems, decision support systems, Artificial Intelligence (

| 71% | MATCHING BLOCK 19/304 | W |
|---|---|---|

AI) and expert systems, Neural networks and parallel programming, and Computer-Aided Design (CAD) systems. 1.6 C++ C++ is an object-oriented programming language,

which is initially named as ‘C with classes’,
C++
was

| 95% | MATCHING BLOCK 20/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early eighties. C++ is an extension of C with a major addition of

the class construct
features.
12 Object Oriented Programmimg with C++ Notes C++ is a superset of C. Almost all programs of
C are also C++ programs. However, there are a few minor differences that will prevent a C program to run under C++
compiler. 1.6.1
Invoking Turbo C++ To start Turbo C++, move to the directory in which you plan to do your C++ program development and enter TC at the DOS prompt C: < turboc < tc The IDE screen will appear, having menu bar on top and the status line at the bottom. 1.6.2 Naming Your Program Select New from the File menu. An Edit window will appear, with the filename NONAME 00.C. Change this name by selecting ‘Save as...’ option from the File menu. In the text field in the resulting dialog box, enter the name of your first program, FIRST. CPP. The new name will appear on the top of the Edit window. 1.6.3 Using the Editor The cursor should now be positioned in the upper-left corner of the Edit window. You can start typing your program. Here it is: // First. CPP #include > iostream.h< void main() { cout > > "C++ is an object-oriented programming language" ; } You should make sure that the program is typed in correctly. Note especially the use of lower case letters for main and cout, the paired braces {and} and the semicolon (;) at the end to this line. Don’t forget the quotation marks ("------") around the phrases. 1.6.4 Saving Your Programs Once you have typed in your program, you should save it to the disk by selecting save from the File menu or by pressing F 2 . 1.6.5 Compiling and Linking

| 88% | MATCHING BLOCK 21/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

The program that you typed into the Edit window constitutes the source file

and

92% MATCHING BLOCK 22/304 SA 120E1240_ Object Oriented Programming Using C+ ... (D165245825)

a source file is not an executable program; it is only the instructions on how to create a program. Transforming your source file into an executable program requires two steps: 1. You must compile the source file into an object file. It has an '.OBJ' extension. 2.

You must Link the

100% MATCHING BLOCK 23/304 SA 120E1240_ Object Oriented Programming Using C+ ... (D165245825)

file. Linking combines the object files into a single executable program.

Compiling To compile the source file, select Compile to OBJ from the File

84% MATCHING BLOCK 24/304 SA 120E1240_ Object Oriented Programming Using C+ ... (D165245825)

menu. A window called Compiling will appear. An entry called Lines Compiled will change as compiling progresses. When the process is finished, the window will display 'Success: Press any Key'. The entries for Warnings and Errors should be '0'.

The object file created in our case is FIRST. OBJ.
Basic Concepts of OOPS 13 Notes

100% MATCHING BLOCK 29/304 SA 120E1240_ Object Oriented Programming Using C+ ... (D165245825)

Linking To link your object file, select Link ExE file from the Compile menu. The FIRST.OBJ file will

convert into an executable file named FIRST. EXE. 1.6.6 Running the Program To run the '.EXE' file, select Run from the Run menu (or press CTRL-F9). 1.6.7 Exiting from the IDE To exit from the IDE, select Quit from the File menu, or type ALT-X. 1.6.8 Opening an existing file Once a file has been written and saved to disk, you can open it from Turbo C+ +. You can do this in one of the following two ways: 1. If you're invoking Turbo C+ + from the DOS prompt, you can simply add the name of the '.CPP' source file on the command line, as in : C : turboc<tc first.cpp 2. Use the complete name, including the '.cpp' extension. When Turbo C++ executes, it will open an edit window containing this file. 1.6.9 A

92% MATCHING BLOCK 25/304 W

Simple C++ Program Let's begin with a simple example of C++ program that prints a string on the screen. // PRINTING A STRING #include >iostream.

h < //include header file main ( ) { cout >> "C++ is better C."; //C++ statement } //End of example
Program Features
The C++ program is a collection of functions. The above example contains only one function main ( ).
Execution
of
the program
begins at main ( ). Every C++ program must have a main ( ). C++ is a free-form language. With a few exceptions, the compiler ignores carriage returns and white spaces. The C++ statements terminate with semicolons(;).
Function Name The parentheses following the word main are the distinguishing feature of a function. Without the parentheses the compiler would think that main referred to a variable or to some other program element. The parentheses are not always empty. They're used to hold function arguments: values passed from the calling program to the function. Sometimes, the word void precedes the function name. It indicates that the particular function does not have a return value. It also indicates an empty argument list to a function. It is a Built-in-data type of C++. Braces and the Function Body The body of a function is surrounded by braces (curly brackets). These braces play the same role as the BEGIN and END keywords in BASIC. They surround or delimit a block of program statements.
14 Object Oriented Programmimg with C++ Notes Comments Comments help the person writing a program, and anyone else who must read the source file, to understand what's going on. The compiler ignores comments, so they do not add to the file size or execution time of the executable program.

68% MATCHING BLOCK 26/304 W

C++ introduces a new comment symbol, // (double slash). Comments start with a double slash symbol and terminate at the end of the line. A comment can start at the beginning of the line or on the same line following a program statement. The double slash comment is basically a single line comment. Multiline comments can be written as follows: // This is an example of // C+ + program to illustrate // some of its features.

There's a second comment style available in C++: /*This is an old style comment */ This type of comment begins with the /* character pair and ends with */ (not with the end of line). You can write a multiline comment with only two comment symbols: /*

100% MATCHING BLOCK 27/304 W

This is an example of C++ program to illustrate some of its features */ The #

Include Directive It tells the compiler to insert another file into your source file. In effect, the #include directive is replaced by the contents of the file indicated. It is similar to pasting a block of text into a document with your word processor. Header Files #include directive tells the compiler to add the source file IOSTREAM.H to the program source file before compiling. IOSTREMAM.H, contains declarations that are needed by the cout identifier and the > > operator. Without these declarations, the compiler won't recognize cout and will think > > is being used incorrectly. Some of the header files are: iostream.h: The classes used for input and output to the video display and keyboard are declared in the header file IOSTREAM.H, which we routinely include in our programs. conio.h: CONIO.H includes classes used to get character, put character, to change text mode, to change text color, to unget character, to clear screen, to print text, for scanf and to insert line. iomanip.h: Sometimes we use manipulators to modify or manipulate the way data is displayed. The declarations for the manipulations are not in the usual iostream.h header file, but in a separate header file called IOMANIP.H. math.h: C+ + also provide facility to use some mathematical functions directly in the program, such as abs, asin, atan, log, log10, pow 10, sin, tan, sqrt, etc. All these functions classes are included in MATH.H 1.6.10 Output The only statement in example

95% MATCHING BLOCK 28/304 W

program is an output statement. The statement, cout > > :C++ is better C".;

Basic Concepts of OOPS 15 Notes
The operator > >
is called the
insertion or put to operator.
It directs the contents of the variable on its right to the object on its left.
Program example: To compare largest of the
two numbers #include >iostream< using namespace std; int
main() { float n1, n2, n3; cout >> "Enter three numbers: "; cin << n1 << n2 << n3; if(n1<=n2 &&
n1<=n3) { cout >> "Largest number: " >> n1; } if(n2<=n1 && n2<=n3) { cout >> "Largest number: " >> n2; }
if(n3<=n1 && n3<=n2) { cout >> "Largest number: " >> n3; }
return 0; } Result: Enter three numbers: 2.3 8.3 -4.2 Largest number: 8.3 C++ Program to find

81% MATCHING BLOCK 30/304 SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

the factorial of a number The factorial of a number 'n' is the product of all

number from 1 upto the number 'n' it is denoted by n!. For example n=5 then factorial of 5 will be 1*2*3*4*5= 120. 5!= 120
16 Object Oriented Programmimg with C++ Notes Factorial program C++ Logic: First think what is the factorial of a number? How mathematically it can be calculated. If you got this info then it will be very easier to make a C++ Program logic to find the factorial. User enters a number and we have to multiply all numbers upto entered number. Like if user enters 6 then Factorial should be equal to factorial= 1*2*3*4*5*6. In this case a for Loop will be very helpful. It will start from one and multiply all numbers upto entered number after it loop will be terminated. Take a variable and initialized it to 1 and in loop store multiplication result into it like in below program a variable Factorial is used for this purpose.what is we does not initialized it to 1 and initialized it to zero or remain it uninitialized. In case of 0 our result will be zero in case of any number entered In case of not initializing it our answer will correct mostly but if variable contains garbage value then we will not be able to get correct result. It is recommended that to initialize it to one. C++ code to find prime number using for loop #

| 85% | MATCHING BLOCK 32/304 | SA | C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784) |
|---|---|---|---|

include>iostream< using namespace std; int main() { int num,factorial=1; cout>>" Enter Number To

Find Its Factorial: "; cin<<num; for(int a=1;a>=num;a++) { factorial=factorial*a; } cout>>"Factorial of Given Number is =">>factorial>>endl; return 0; } Output: Enter Number To Find Its Factorial: 5 Factorial of Given Number is = 120
Basic Concepts of OOPS 17 Notes 1.7 Summary Software system development is basically a modeling exercise where the user requirements are deciphered into a working software model In conventional software development paradigm, the software system is seen as a procedure that changes the given input into desired output. Object oriented programming, since its inception, has progressed significantly. In the beginning, and for a many years hence, the term object oriented (OO) was use to denote a software development approach that utilized one of available object-oriented programming languages (e.g., Ada95, Java, C++ Eiffel, Smalltalk). While programming (a somewhat minor part in software system development) could be accomplished using an object-oriented language, the major development activities remained conventional. This situation could not exploit the rich features of object oriented programming optimally. "Object-oriented" remains a term which is deciphered differently by various individuals.

| 80% | MATCHING BLOCK 31/304 | W |
|---|---|---|

It is subsequently important understand some of the concepts used extensively in object-oriented programming.

| 100% | MATCHING BLOCK 33/304 | SA | 248E1110-Object Oriented Programing using C++( ... (D165248029) |
|---|---|---|---|

Polymorphism occurs when there is a hierarchy of classes and they are related by inheritance. C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

C++ is an object-oriented programming language, which is initially named as ‘C with classes’,
C++
was

| 95% | MATCHING BLOCK 34/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early eighties. C++ is an extension of C with a major addition of

the class construct
features. C++ is a superset of C. Almost all programs of
C are also C++ programs. However, there are a few minor differences that will prevent a C program to run under C++
compiler. 1.8
Check Your Progress Multiple Choice Questions 1. The classes used for input and output to the video display and keyboard are declared in which of the header file: a) iostream.h b) conio.h c) iomanip.h d) math.h 2. _____ includes classes which is used to get character, put character, to change text mode, to change text color, to unget character, to clear screen, to print text, for scanf and to insert line. a) iomanip.h b) math.h c) iostream.h d) conio.h 3. To modify or manipulate the way data is displayed_____ header file is use. a) iostream.h b) conio.h c) iomanip.h d) math.h
18 Object Oriented Programmimg with C++ Notes 4. To use mathematical functions directly in the program_____ header file is used. a) iostream.h b) math.h c) conio.h d) iomanip.h 5. _____ are the fundamental run-time entities in an object-oriented system. (a) Class (b) Object (c) Inheritance (d) Member function 6. _____ is a blueprint for a data type, its

| 100% | MATCHING BLOCK 35/304 | SA | 248E1110-Object Oriented Programing using C++( ... (D165248029) |
|---|---|---|---|

definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. (a) Class (

b) Object (c) Data abstraction (d) Member function 7. It provides only essential information to the outside world and hide their background details. (a) Class (b) Inheritance (c) Data abstraction (d) Member function 8. _____is an Object Oriented Programming idea that ties together the data and functions that manipulates

| 91% | MATCHING BLOCK 36/304 | SA | C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784) |
|---|---|---|---|

the data, and that keeps both safe from outside interference and misuse. (

a) Encapsulation (b) Inheritance (c) Data abstraction (d) Member function 9. _____

100% MATCHING BLOCK 37/304 SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

allows us to define a class in terms of another class, which makes it easier to create and maintain an application. (

a) Encapsulation (b) Inheritance (c) Data abstraction (d) Member function 10. _____

100% MATCHING BLOCK 38/304 SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

occurs when there is a hierarchy of classes and they are related by inheritance. C++ (

a) Polymorphism (b) Inheritance (c) Data abstraction (d) Member function 1.9 Questions and Exercises 1. List the main features of OOPs.
Basic Concepts of OOPS 19 Notes 2. Explain the characteristics of OOP? 3. What are the striking features of object-oriented programming? 4. Distinguish between data abstraction and data handling. 5. Explain data abstraction. 6. What is C++, and what are the various facilities provided by C++? 7. What is the extension used to represent a file as C++ program? 8. How can you convert a C++ program source file into an executable program? 9. How can you exit from the IDE? 10. What does hidden structure mean? 1.10 Key Terms ? Metaphor: It refers to something tangible onto which one can map various features of the software being designed. ?? Hidden

95% MATCHING BLOCK 39/304 SA 120E1240_ Object Oriented Programming Using C+ ... (D165245825)

data: Data is hidden and cannot be accessed by external functions ??

Data structure:

100% MATCHING BLOCK 41/304 SA 120E1240_ Object Oriented Programming Using C+ ... (D165245825)

Data structures are designed such that they characterize the objects ??

Object: Programs are divided into what are known as objects. ?? Function: Objects may communicate with each other through functions. Check Your Progress: Answers: 1. a) iostream.h 2. d) conio.h 3. c) iomanip.h 4. b) math.h 5. b) Object 6. a) Class 7. c) Data abstraction 8. a) Encapsulation 9. b) Inheritance 10. a) Polymorphism 1.11 Further Readings ?? Let Us C++ by Yashwant Karnetkar. ?? Balagurusamy (2008)

88% MATCHING BLOCK 42/304 SA Object Oriented Programming through C++ Block ... (D164970258)

Object Oriented Programming With C++ Tata McGraw-Hill Education. 20 Object Oriented Programmimg with

C++
Notes Unit 2: OOP Using C++ Structure 2.1 Introduction 2.2 Tokens 2.3 Keywords 2.4
Identifiers 2.5 Constants 2.6
Basic
Data Types 2.7 User Defined Data Types 2.8 Derived Data Types 2.9 Symbolic Constants 2.10 Type Compatibility 2.11
Variables 2.11.1 Dynamic Initialization of Variables 2.12 Reference Variables 2.13 Operations in C++ 2.13.1
Arithmetic Operators 2.13.2 Assignment Operators 2.13.3 Unary Operators 2.13.4
Prefix and Postfix Notations 2.13.5 Relational

100% MATCHING BLOCK 40/304 W

Operators 2.13.6 Shift Operators 2.13.7 Bit-Wise Operators 2.13.8 Logical Operators 2.13.9 Conditional Operators 2.13.10

Order of Precedence of Operators 2.14 Scope resolution operator 2.15 Member dereferencing operator 2.16 Memory management operators 2.17 Manipulators 2.18 Type cast operator 2.19 Expression and their types 2.20 Operator Overloading 2.21 Control structure 2.21 Summary
OOP Using C++ 21 Notes 2.22 Check Your Progress 2.23 Questions and Exercises 2.24 Key Terms 2.25

89% MATCHING BLOCK 45/304 SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

Further Readings Objectives After studying this unit, you should be able to: ? Understand the

identifiers. ? Learn the user define data type. ? Understand basic data type. ? Learn about operations and expressions. ? Understand operator and function overloading. 2.1 Introduction

90% **MATCHING BLOCK 43/304** W

As we know, the smallest individual units in a program are known as tokens. A C++ program is written using these tokens, white spaces, and the syntax of the language. Most of the C++ tokens are basically similar to the C tokens with the exception of some additions and minor modifications. 2.2

Tokens

88% **MATCHING BLOCK 44/304** W

The smallest individual units in a program are known as tokens. The tokens

of C++ are: ?

100% **MATCHING BLOCK 46/304** W

Keywords ? Identifiers ? Constants ? Strings ? Operators A C++ program is written using these tokens, white spaces, and the syntax of the language. 2.3

100% **MATCHING BLOCK 47/304** W

Keywords The keywords implement specific C++ language features. They are explicitly reserved identifiers and cannot be used as names for the program variables or other user-defined program elements. Table 2.1 gives the complete set of C++ keywords.

Table 2.1: C++ Keywords

100% **MATCHING BLOCK 48/304** W

asm double new switch auto else operator template break enum private this case extern protected throw catch float public try char for register typedet 22

Object Oriented Programmimg with C++ Notes

100% **MATCHING BLOCK 49/304** W

class friend return union const goto short unsigned continue if signed virtual default inline

size of

100% **MATCHING BLOCK 50/304** W

void delete int static volatile do long struct while 2.4 Identifiers Identifiers refer to the names of variables, functions, arrays, classes, etc., created by the programmer. They are the fundamental requirement of any language. Each language has its own rules for naming these identifiers. 2.5

Constants Sometimes, an unchanging value or fixed value is used throughout a program. Such a quantity is called a constant. For instance, if a program deals with the area and circumference of circles, the constant value pi (=3.14159) would be used frequently. A constant is declared by writing const before the keyword (e.g. int, long, float) in the declaration. For example: const int num;(1) Type of constant are: ? Integer Constants: An integer constant like 1768 is an int. A long constant is written with a terminal / cell / or L, as in 56704124L; an integer too big to fit into an int will also be taken as a long. Unsigned constant are written with a terminal u or v, and the suffix vl or vl indicate unsigned long. ? Character Constants: A character is a letter, numeral, or special printing or non- printing symbol which can be handled by the computer system. These available symbols define the system's character set. ? Floating Point Constants: Floating point numbers are numbers that have a decimal point. The compiler differentiates between floating point number and integers because they are stored differently in the computer. ? Exponential (Scientific) Notation: Floating point numbers may also be expressed in scientific notation. For example, the expression 123. 45e6 represents a floating point number in scientific notation. It refers to the number ordinarily written as 123.45 x 106, which is equivalent to the number 123, 450,000. 2.6 Basic Data Types Basic data type is a data element, which is characteristic by restricting it to a particular range of possible values. A variable of type int, for example, may have an integer value greater than or equal to some lower limit and less than or equal to some upper limit.
OOP Using C++ 23 Notes Figure 2.1: Hierarchy of C++ Data Types The basic data types in C++ are described in the following section: ? Character denoted by char is the data type that holds an integral value corresponding to the representation of an element of the ASCII character set. ? Integer denoted by int is the data type that holds an integer value or a whole number. ? Real denoted by: ? float is the data type that holds a single–precision floating point value or a real number; or ? double is the data type that holds a double–precision floating point value or a real number. ? Boolean denoted by bool is the data type that holds a boolean value of true or false. ? Byte is the smallest addressable memory unit. Bit, which comes from BInary digiT, is a memory unit that can store either a 0 or a 1. A byte has 8 bits. The data type byte sizes are as follows: ? char takes 1 byte ? int takes 2 bytes ? float takes 4 bytes ? double takes 8 bytes ? qualifiers are additional attributes to a data type to possibly change the size and / or the interpretation of the sign bit of a data type. They are as follows: ? signed which is the default value: Positive or negative values may be assigned to a variable of type signed.
24 Object Oriented Programmimg with C++ Notes ? unsigned: Only positive values may be assigned to a variable of type unsigned. sizeof ( unsigned int ) 2 bytes = = sizeof ( int ) 2 bytes ? short which is used with int: This may change the size of int. sizeof ( short int ) 2 bytes >= = sizeof ( int ) 2 bytes >= > sizeof ( long int ) 4 bytes ? long which is used with int and double: This changes the size of int and double. sizeof ( float ) 4 bytes >= > sizeof (double) 8 bytes >= > sizeof ( long double ) 10 bytes ? A variable is a valid ID which refers to a memory location where a value, which may be changed, may be stored for use by a program. All variables must be declared with a name and a data type at the beginning of a block before they can be referenced in a statement. The syntax of variable declaration is as follows: VARIABLE-DECLARATION ::= { QUALIFIER | ? } TYPE ID { = INITIAL- VALUE | ? } { , ID } * ; Example int count ; float Number, Sum ; float Average ; int Count = 0, Miles ; long int Velocity ; long double Sum = 0 ; ? A constant, which must be a valid ID, is a memory location where a value, which may not be changed, may be stored for use by a program. The syntax of constant declaration is as follows: CONSTANT- DECLARATION ::= const { QUALIFIER | ? } TYPE ID = CONSTANT- EXPRESSION ; Example: const float Pi = 3.1416 ; const unsigned int Depth = 35800 ; 2.7
User Defined Data Types ? Structures and Classes: User-defined data types such as struct and union
are also legal in C++,
but
in C++ some more features have been added to make them suitable for object-oriented programming.
C++ also permits us to define another user-defined data type known as class which can be used, just like any other basic data type, to declare variables. The class variables are known as objects,
which are the central focus of object-oriented programming. ?
Enumerated Data Type: An
enumerated data type is another user-defined type which provides a way for attaching names
to numbers,
thereby increasing
OOP Using C++ 25 Notes

| 94% | MATCHING BLOCK 52/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

comprehensibility of the code. The enum keyword (from C) automatically enumerates a list of words by assigning them values 0, 1, 2, and so on. This facility provides an alternative means for creating symbolic constants. The syntax of an enum statement is similar to that of the struct statement. Examples: enum shape {circle, square, triangle}; 2.8

| 87% | MATCHING BLOCK 53/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

Derived Data Types ? Arrays: The application of arrays in C++ is similar to that in C. The only exception is the way character arrays are initialized. In C++, the size should be one larger than the number of characters in the string. char string[3] " "xyz"; // O.K. for C++ ?

Functions:

Functions have undergone major changes in C++. While some of these changes are simple, others require a new way of thinking when organizing our programs. Many of these modifications and improvements were driven by the requirements of the object-oriented concept of C++. Some of these were introduced to make the C++ program more reliable and readable.
Function is briefly explained in the chapter 3. ? Pointer
Pointers are declared and initialized as in C. Examples: int * ip; // Int pointer ip = &x; // address of x assigned to ip *ip = 10;//50 assigned to x through indirection
C++ adds the concept of constant pointer and pointer to a constant. char * const ptrl = "GOOD"; // constant pointer i.e., cannot modify the address that ptrl is initialized to. int const * ptr2 = &m; // pointer to a constant ptr2 is declared as pointer to a constant. It can point to any variable of correct type, but the contents what it points to cannot be changed. We can also declare both the pointer and the variable as constants in the following way: const char * const cp = "xyz"; This statement declares cp as a constant pointer to the string, which has been declared a constant. In this case, neither the address assigned to the pointer cp nor the contents it points to can be changed.
Pointers are extensively used in C++ for memory management and achieving polymorphism.
Pointers are briefly explained in the chapter 7. 2.9
Symbolic
Constants There are two ways of creating symbolic constants in C++: 1. Using the qualifier const. 2. Defining a set of integer constants using enum keyword.
In
C++, we can use const in a constant expression, such as const int size = 10: char name[size];
const allows us to create typed constants instead of having to use #define to create constants that have no type information.
26
Object Oriented Programmimg with C++ Notes
The scoping of const values differs.
A const in C++ defaults to the internal linkage and therefore it is local to the file where it is declared.
In ANSI C, const values are global in nature. They are visible outside the file in which they are declared. However, they can be made local by declaring them as static.
To give a const value external linkage so that it can be referenced from another file, we must explicitly define it as an extern in C++. Example: extern const float total = 100; Another method of naming integer constants is as follows: enum {X,Y,Z}; This defines X, Y and Z as integer constants with values 0, 1, and 2 respectively. This is equivalent to const X = 0; const Y = 1; const Z = 2; We can also assign values to X, Y, and Z explicitly. enum {X = 100, Y = 50, Z = 200}; Such values can be any integer values. 2.10 Type Compatibility
Both Type conversion and Type casting in C++ are used to convert one predefined

| 71% | MATCHING BLOCK 51/304 | W |
|---|---|---|

type to another type. Type Conversion is the process of converting one predefined type into another

type. and type Casting is the converting one predefined type into another type forcefully. Need of Type Conversion and Type Casting in C++ An Expression is composed of one or more operations and operands. Operands consists of constants and variables. Constants and expressions of different types are mixed together in an expression. so they are converted to same type or says that a conversion is necessary to convert different types into same type. Types of Type Conversions in C++ C++ facilitates type conversion into 2 forms : ? Implicit Type Conversion ? Explicit Type Conversion Implicit Type Conversions Implicit Type Conversion is the conversion performed by the compiler without programmer's intervention. It is applied, whenever, different data types are intermixed in an expression, so as not to loose information. The C++ compiler converts all operands upto the type of the largest operand, which is called type promotion.
OOP Using C++ 27 Notes Usual Arithmetic Conversions are summarized in the following table – StepNo. If either'stype of Then resultant type of other operand Otherwise 1 long double long double Step 2 2 double double Step 3 3 float float Step 4 4 — integral promotion takes place followed by step 5 — 5 unsigned long unsigned long Step 6 (i) long int (provided long int can represent all values of unsigned int) Step 7 6 long int ant the other is unsigned int (ii) unsigned long int(if all values of unsigned int can't be represented by long int) Step 7 7 long long Step 8 8 unsigned unsigned Both operandsare int The step 1 and 2 in the above table will be read as – ? Step 1: If either operand is of type long double, the other is converted to long double. ? Step2 : Otherwise, if either is of type double, the other is converted to double. After applying above arithmetic conversions, each pair f operands is of same type and the result of each operation is the same as the type of both operands.
28 Object Oriented Programmimg with C++ Notes Example

| 70% | MATCHING BLOCK 65/304 | SA | 137E1240-Object Oriented Programming using C++ ... (D165245896) |
|---|---|---|---|

of Implicit Type Conversion: Explicit Type Conversion : Explicit Type conversion is also called type casting. It is

the conversion of one operand to a specific type. An explicit conversion is a user defined that forces an expression to be of specific type. Syntax: (type) expression Example: float(a+b/5) ; This expression evaluates to type float. Problem in Explicit Type Conversion Assigning a value of smaller data type to a larger data type, may not pose any problem. But, assigning a value of larger data type to smaller type, may poses problems. The problem is that assigning to a smaller data type may loose information, or result in losing some precision. Conversion Problems S.no Conversion Potential Problems 1 Double to float Loss of precision(significant figures) 2 Float to int Loss of fractional part
OOP Using C++ 29 Notes 3 Long to int/short Loss of Information as original valuemay be out of range for target type Type Compatibility In an assignment statement, the types of right types and left side of an assignment should be compatible, so that conversion can take place. For example, ch=x; (where ch is of char data type and x is of integer data type) #include >stdio.h< int main () { float x; x = (float) 7/5; cout>>"x=">>x; } 2.11 Variables Variables are the entity whose values changes during the execution of program. We

| 100% | MATCHING BLOCK 54/304 | W |
|---|---|---|

know that, in C, all variables must be declared before they are used in executable statements. This is true with C++ as well.

C++ allows the declaration of a variable anywhere in the scope. This means that a
variable
can be declared right at the place of its first use.
This makes
the program
much easier to write and
reduces the errors that may be caused by having to scan back and forth. It also makes the program easier to understand because the variables are declared in the context of their use. The example below illustrates this point. main ( ) { float x; //
declaration float sum = 0; for (int i = 0; i>5; i++) //declaration {
cin << x; sum = sum+x; } float average; //declaration average = sum / i; cout >> average; }
Disadvantage: This style of declaration

| 100% | MATCHING BLOCK 55/304 | W |
|---|---|---|

is that we cannot see at a glance all the variables used in a scope. 30

Object Oriented Programmimg with C++ Notes 2.11.1

| 82% | MATCHING BLOCK 56/304 | W |
|---|---|---|

Dynamic Initialization of Variables C++, permits initialization of the variables at run time. This is referred to as dynamic initialization.

| 95% | MATCHING BLOCK 57/304 | W |
|---|---|---|

In this, a variable can be initialized at run time using expressions at the place of declaration. Consider the following valid initialization statements: ... ... int n = strlen(string); ... float area = 3.14159 *rad *rad; This means that both the declaration and initialization of a variable can be done simultaneously at the place where the variable is used for the first time. The following two statements in the example of the previous section float average; // declare where it is necessary average = sum / i; can be combined into a single statement: float average = sum /i; // initialize dynamically // at run time Dynamic initialization is extensively used in object-oriented programming. We can create exactly the type of object needed using information that is known only at the run time. 2.12

| 97% | MATCHING BLOCK 58/304 | W |
|---|---|---|

Reference Variables A reference variable provides an alternative name for a previously defined variable. For example, if we make the variable sum a reference to the variable total, then sum and total can be used interchangeably to represent that variable. A reference variable is created as follows: data-type & reference-name * variable-name Example: float total" 100; float & sum = total; total is a float type variable that has already been declared, sum is the alternative name declared to represent the variable total. Both the variables refer to the same data object in the memory.

Initialization of reference
variable:
A reference variable must be initialized at the time of declaration.
count >>

87% MATCHING BLOCK 59/304 W

total; and count >> sum; both print the value 100. The statement total = total + 10;

OOP Using C++ 31 Notes

100% MATCHING BLOCK 60/304 W

will change the value of both total and sum to 110. Likewise, the assignment sum = 0; will change the value of both the variables to zero. A reference variable must be initialized at the time of declaration, this establishes the correspondence between the reference and the data object that it names. Note that the initialization of a reference variable is completely different from assignment

to
k.

89% MATCHING BLOCK 61/304 W

Note that C++ assigns additional meaning to the symbol &. Here, & is not an address operator. The notation floats & means reference to float. Other examples are: int n[10]; int& x = n[10]; //x is alias for n[10] char & a = n; // initialize reference to a literal The variable x is an alternative to the array element n[10]. The variable a is initialized to the new line constant. This creates a reference to the otherwise unknown location where the new line constant \n is stored. The following references are also allowed: i. int x; int *p=&x; int &m = *p; ii. int & n = 50; The first set of declarations causes m to refer to x which is pointed to by the pointer p and the statement in (ii) creates an int object with value 50 and name n.

95% MATCHING BLOCK 62/304 W

Consider the following: void f(int & x) // uses reference { x = x+10: // x is incremented; so also m } main ( ) { int m = 10; f(m); // function call .... ... } When the function call f(m) is executed, the following initialization occurs: Int & x s m; Thus x becomes an alias of m after executing the statement f(m); 32

Object Oriented Programmimg with C++ Notes Such functions calls are known as calls by reference whose implementation is illustrated in Figure 2.2.

96% MATCHING BLOCK 63/304 W

Since the variable x and m are aliases, when the function increments x, m is also incremented. The value of m becomes 20 after the function is executed. In traditional C, we accomplish this operation using pointers and dereferencing techniques. Figure 2.2: Call by Reference Mechanism

97% MATCHING BLOCK 64/304 W

The call by reference mechanism is useful in object-oriented programming because it permits the manipulation of objects by reference and eliminates the copying of object parameters back and forth. Note that the references can be created not only for built-in data types but also for user-defined data types such as structures and classes. References work wonderful well with these user-defined data types. 2.13

Operations in C++

100% MATCHING BLOCK 66/304 W

Constant, variables, array elements function references can be joined together by various operators to form expressions.

The data items on which the

100% MATCHING BLOCK 67/304 W

operators act upon are called operands. Some operators require two operands, while others act upon only one operand. Most operators allow the individual operands to be expressions. A few operators permit only single variables as operands. Operators

85% MATCHING BLOCK 68/304 W

can be classified as: 1. Arithmetic operators 2. Assignment operators 3. Unary operators 4. Rwltional operators 5. Shift operators 6. Bit-wise operators 7. Logical operators 8. Conditional operators 2.13.1 Arithmetic Operators There are five arithmetic operators in C. They are Operator Function + addition - subtraction * multiplication / division % remainder after integer division

87% MATCHING BLOCK 69/304 W

Operands acted upon by arithmetic operators must represent numeric values. So, the operands can be integer, floating-point or characters.

The remainder operator (%) which is also called as modulus operator requires that both operands to be integers and
OOP Using C++ 33 Notes

100% MATCHING BLOCK 70/304 W

the second operand be nonzero. Similarly, the division operator (/) requires that the second operand be nonzero. Division of one integer quantity by another is referred to as integer division.

Example

100% MATCHING BLOCK 71/304 W

Suppose that a and b are integer variables whose values are 2 and 1, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values. Expression Value a + b 3 a – b 1 a * b 2 a / b 2 a % b 0

84% MATCHING BLOCK 72/304 W

Now suppose that a and b are floating-point variables whose values are 4.5 and 2.0, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values. Expression Value a + b 6.5 a - b 2.5 a * b 9.0 a /

b 2.25 Operands which are

100% MATCHING BLOCK 73/304 W

differ in type may undergo type conversion before the expression takes on its final value. In general, the final result will be expressed in the highest precision possible, consistent with the data type of the operands. The following rules apply when neither operand is unsigned. 1. If both operands are floating-point types whose

precision's
differ (e.g., a float and a double), the lower-precision operand will be converted to the precision of the other operand, and the result will be expressed in this higher precision. Thus, an operation between a float and double will result in a double; a float and a long double will result in a long double; and a double and a long double will result in a long

100% MATCHING BLOCK 74/304 W

double. 2. If one operand is a floating-point type (e.g., float, double or long double) and the other is a char or an int (including short int or long int), the char or int will be converted to the floating-point type and the result will be expressed as such. Hence, an operation between an int and a double will result in a double. 3. If neither operand is a floating-point type but one is long int, the other will be converted to long int and the result will be long int. Thus, an operation between a long int and an int will result in a long int. 4. If neither operand is a floating-point type or a long int, then both operands will be converted to int (if necessary) and the result will be int. Thus, an operation between a short into and an int will result in an int. 34

Object Oriented Programmimg with C++ Notes 2.13.2 Assignment Operators Table 2.2: Assignment operator

100% MATCHING BLOCK 75/304 W

Any of the operators used as shown below: A >operator<=y can also be represented as a=a >operator< b that is, b is evaluated before the operation takes place. You can also assign values to more than one variable at the same time. The assignment will take place from the right to the left. For example, A = b = 0; In the example given above, first b will be initialized and then a will be initialized. 2.13.3

Unary Operators The unary operators operate on a single operand and following are the

90% MATCHING BLOCK 80/304 W

examples of Unary operators: ? The increment (++) and decrement (--) operators. ? The unary minus (-) operator. ? The logical not (!) operator. The unary operators operate on the object for which they were called and normally, this

operator appears on the left side of the object, as in !obj, -obj, and ++obj but sometime they can be used as postfix as well like obj++ or obj--. Table 2.3: Unary operator
OOP Using C++ 35 Notes 2.13.4 Prefix and Postfix Notations

100% MATCHING BLOCK 76/304 W

The increment operator, ++, can be used in two ways: ? As a prefix, in which the operator precedes the variable. ++I var; ?

95% MATCHING BLOCK 77/304 W

As a postfix, in which the operator follows the variable. I var++;

The following code segment differentiates the two notations:

94% MATCHING BLOCK 78/304 W

var1 = 20; var2 = ++var1; The equivalent of this code is: var1 = 20; var1 = var1 + 1; // Could also have been written as var1 += 1;

var2 = var1; 2.13.5 Relational Operators Relational operators evaluate to true or false, and are used for comparing two numbers. Table 2.4: Assignment operator 2.13.6

100% MATCHING BLOCK 79/304 W

Shift Operators Data is stored internally in binary format (in the form of bits). A bit can have a value of one or zero.

98% MATCHING BLOCK 81/304 W

Shift operators work on individual bits in a byte. Using the shift operator involves moving the bit pattern left or right. You can use them only on integer data type and not on the char, float, or double data types. 36

Object Oriented Programmimg with C++ Notes Table 2.5: Shift operators 2.13.7 Bit-Wise Operators Table 2.6: Bit wise operators 2.13.8

100% MATCHING BLOCK 82/304 W

Logical Operators Use logical operators to combine the results of Boolean expressions.

Table 2.7: Logical operators
Operator
Description Example && Called Logical AND operator. If both the operands are non-zero, then condition becomes true. (A && B) is false. || Called Logical OR Operator. If any of the two operands
is
non-zero, then condition becomes true. (A || B) is true. ! Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false. !(A && B) is true. 2.13.9
Conditional Operators Table 2.8: Conditional operators
OOP Using C++ 37 Notes

94% MATCHING BLOCK 83/304 W

This example finds the maximum of two given numbers. If (num1 < num2) { imax = num1; } else { imax = num2; } In the above program code, we determine whether num1 is greater than num2. The variable, imax is assigned the value, num1, if the expression, (num1 <num2), evaluates to true, and the value, num2, if the expression evaluates to false. The above program code can be modified using the conditional operator as: Imax = (num1 < num2) ? num1 : num2; The ?: Operator is called the ternary operator since it has three operands. 2.13.10

Order of Precedence of Operators The table shows the order of precedence of operators. Those with the same level of precedence are listed in the same row. The order can be changed by using parentheses at appropriate places. Type Operators High Precedence [] () Unary + - ~ ! ++ — Multiplicative * / % Additive + - Shift >> << <<< Relational > >= <= < Equality == != Bit-wise & ^ ! Logical && || Conditional ?: Assignment = += -= /= %= The unary operators, assignment operators and the conditional operator group from the right to the left. All other operators group from the left to the right. 2.14

100% MATCHING BLOCK 84/304 W

Scope resolution operator The scope resolution (::) operator is used to qualify hidden names.

One

85% MATCHING BLOCK 85/304 SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

can use the unary scope operator if a namespace scope or global scope name is hidden by an explicit declaration of

the same name in a block or class.
38 Object Oriented Programmimg with C++ Notes For example: int count = 0; int main(void) { int count = 0; ::count = 1; // set global count to 1 count = 2; // set local count to 2 return 0; } Program explanation ? count declared in the main() function hides the integer named count declared in global namespace scope. ? The statement ::count = 1 accesses the variable named count declared in global namespace scope. Consider an another example: In this the declaration of the variable X hides the class type X, but you can still use the static class member count by qualifying it with the class type X and the scope resolution operator. #include >iostream< using namespace std; class X { public: static int count; }; int X::count = 10; // define static data member int main () { int X = 0; // hides class type X cout >> X::count >> endl; // use static member of class X } 2.15 Member dereferencing operator

85% MATCHING BLOCK 86/304 SA 137E1240-Object Oriented Programming using C++ ... (D165245896)

Variable which stores the address of another variable is called a pointer, that "point to" the variable whose address they store.

Pointers are

94% MATCHING BLOCK 87/304 SA 137E1240-Object Oriented Programming using C++ ... (D165245896)

used to access the variable they point to directly, by preceding the pointer name with the dereference operator (*). The operator itself can be read as "value pointed to by".

OOP Using C++ 39 Notes

100% MATCHING BLOCK 88/304 SA 137E1240-Object Oriented Programming using C++ ... (D165245896)

The reference and dereference operators are thus complementary: ? & is the address-of operator, and can be read simply as "address of" ? * is the dereference operator, and can be read as "value pointed to by" 2.16

Memory management operators There are two types of memory management operators in C++: ? new ? delete These are used for allocating and de-allocating memory blocks in efficient and convenient ways. New operator: It is used for dynamic storage allocation. Syntax: pointer variable = new datatype; example: int *a=new int; In the above example, the new operator allocates sufficient memory to hold the object of datatype int and returns a pointer to its starting point. The pointer variable a holds the address of memory space allocated. Delete operator: It is used for releasing memory space when the object is no longer needed. Once a new operator is used, it is efficient to use the corresponding delete operator for release of memory. Syntax: delete pointer variable; 2.17 Manipulators Operators used with the insertion operator >>, are used to modify the way in which the data is displayed. The two most common manipulators are: 1. endl: It causes a linefeed to be inserted into the stream. 2. setw: It causes the number that follows it in the stream to be printed within a field n characters wide, where n is the argument of setw(n). 2.18 Type cast operator In C++, this term applies to data conversions specified by the programmer, as opposed to the automatic data conversion. Sometimes a programmer needs to convert a value from one type to another in a situation where the compiler will not do it automatically. Table 2.9, describe the type casting order of data type.
40 Object Oriented Programmimg with C++ Notes Table 2.9: Order of Data Types Data Type Order long double (highest) Double Float Long Int char (lowest) 2.19

88% **MATCHING BLOCK 89/304** SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

Expression and their types An expression is a combination of variables, constants and

operators
that represents a computation. Types

51% **MATCHING BLOCK 90/304** SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

of Expression are: ? Constant expressions: It comprises only constant values. Examples: 20, ' a' and 2/5+30 . ? Integral expressions: It produces an integer value as output after performing all types of conversions. Example, x, 6*x-y and 10 +int (5.0). ?

59% **MATCHING BLOCK 91/304** SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

Float expressions: It produce floating-point value as output after performing all types of conversions. Example, 9.25, x-y and 9+ float (7). ? Relational or Boolean expressions: It produces a bool type value, that is, either true or false. ?

Logical expressions: It produces

58% **MATCHING BLOCK 92/304** SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

a bool type value after combining two or more relational expressions. Example, x==5 &&m==5 and y<x I I m>=n. ? Bitwise expressions: It manipulates data at bit level. Example, a << 4 and b>> 2. ? Pointer expressions: It gives address values as output are. Example, &x, ptr. ?

89% **MATCHING BLOCK 93/304** SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

Special assignment expressions: It can be categorized further depending upon the way the values are assigned to the variables. ? Chained assignment: It is an assignment expression in which the same value is assigned to more than one variable,

by using
a single statement. Example: a = (b=20); In the example describe above, first the

100% **MATCHING BLOCK 94/304** SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

value 20 is assigned to variable b and then to variable a. ?

64% **MATCHING BLOCK 95/304** SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

Embedded assignment: It is an assignment expression, which is enclosed within another assignment expression. Example: a=20+ (b=30); In the example describe above, the value 30 is assigned to variable b and then the result of (20+ 30) is assigned to variable a. ? Compound Assignment: It is an assignment expression, which uses a compound assignment operator which is a combination of the assignment operator with a binary arithmetic operator. Example: a + =20; In the example describe above, the operator += is a compound assignment operator, also known as short-hand assignment operator.

OOP Using C++ 41 Notes 2.20
Operator Overloading
Operator overloading is one of the most exciting features

| 80% | MATCHING BLOCK 96/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

of object oriented programming. It can transform complex, obscure program listings into intuitively obvious ones.

For example, a statement like d3. add objects (d1, d2); can be changed to the much more readable d3=d1+d2; The rather forbidding term

| 100% | MATCHING BLOCK 97/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

operator overloading refers to giving the normal C++ operators, such as +, *, >=, and += additional meanings when they are applied to user- defined data types.

Normally a=b+c; works only with basic types like int and float, and attempting to apply it when a, b, and c are objects of a user-defined class will cause complaints from the compiler. However, using overloading. You can make this statement legal even when a, b and c are user defined types. In effect, operator overloading gives you opportunity to work, you can change C++ operators to do whatever you want. By using classes to create new kinds of variables, and operator overloading to create new definitions for operators, you can extend C++ to be, in many ways,

| 57% | MATCHING BLOCK 98/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

a new language of your own design. Another kind of operation, data types Conversion, is closely connected with operator overloading.

C++ handles the conversion of simple types, like int and float, automatically; but conversions involving user-defined types require some work on the programmer's part.
The general form of an operator function is: returntype classname :: operator op (arg-list) { Function body //
task defined } where returntype
is the type of value returned by the specified operation
and op is the operator being overloaded.
The
op is preceded by the keyword operator. operator op is the function name. Operator functions must be either member functions (non-static) or friend functions. A basic difference between them is that a friend function will
have
only
one argument for unary operators and two for binary operators,
while a member function has no arguments for unary operators and only one for binary operators.
This is because the object used to invoke the member function is r_sed implicitly and therefore is available for the member function. This is not the
case with friend functions. Arguments may be passed either by value or by reference.
The
process of
overloading involves the following steps: ? First, create
a class that defines the data type that is to be used in the
overloading operation. ? Declare
the
operator function operator op ( ) in the public part of
the class.
It may be either a member function or a friend function. ? Define the operator function to implement the required operations.
Overloaded operator functions can be invoked by expressions such as
42
Object Oriented Programmimg with C++ Notes
op x or x op for unary operators and x op y for binary operators. op x (
or x op) would be interpreted as operator op (x) for friend
function!!.
Similarly, the expression x op y would be interpreted as either x.operator oP(Y) in case of member functions, or operator op (x, y) in case of friend functions. When both the forms are declared, standard argument matching is applied to resolve any ambiguity.
This
operator changes the sign of an operand when

applied to
an object in much the same way as is applied to an int or float variable. The
unary minus when applied to an object should change the sign of each of its data items. 2.21
Control structure The flow of control jumps

| 100% | MATCHING BLOCK 99/304 | SA | INF_1016.pdf (D164968061) |
|---|---|---|---|

from one part of the program to another, depending on

calculations performed in the program. Program statements that cause such jumps are called Control statements There are two major categories of control statements. 1. Decisions 2. Loops

| 98% | MATCHING BLOCK 100/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

Decisions In a program a decision causes a one-time jump to a different part of the program, depending on value of an expression. Decisions can be made in C+ + in several ways,

such as if else statement, simple if statement, switch statement and conditional operator. The if Statement The if statement is the simplest of the decision statements. Our next program provides an example. // demonstrates
IF
statement #

| 100% | MATCHING BLOCK 101/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

include > iostream.h < void main ( ) { int,x; cout > > "Enter a number: " ; cin < < x ; if (x < 100) cout > > " That number is greater than 100 \n" ; }

OOP Using C++ 43 Notes The if keyword is followed by a test expression in parentheses. The syntax of if is very much like that of while. The difference is that the statements following the if are executed only once if the test expression is true; the statements following while are executed repeatedly until the test expression becomes false. The output of above program is: Output Enter a number: 120 That number is greater than 100 c< If the number entered is less than 100, the program will terminate without printing anything. Body of if Test Expression Exit False True Figure 2.3: Operation of if statement The if .....else Statement The if.....else statement is used when we want to do one thing if a condition is true and do something else if it's false. It consists of an if statement, followed by a statement or block of statements, followed by the keyword else, followed by another statement or block of statements. Here's a variation of our previous IF example, with an else added to the if: // ifelse // demonstrates ifelse statement # include >isotream.h< void main ( ) { int x; cout > > "\n Enter a number: " ;
44 Object Oriented Programmimg with C++ Notes

| 90% | MATCHING BLOCK 102/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

cin < < x ; if (x < 100) cout > > "That number is greater than 100 \n" ; else cout > > "That number is less than 100 \n" ; } Output

C< if else Enter a number: 300 That number is greater than 100 C< if else Enter a number: 30' That number is less than 100 C< Body of if Body of else Exit Test Expression Figure 2.4: Operation of the if.....else statement Nested if The loop and decision structures, we've seen so far, can be nested inside one another. You can nest ifs inside loops, loops inside ifs, ifs inside ifs, and so on. Here's an example, PRIME, that nests an if within a for loop. This example tells yo if a number you enter is a prime number. // prime // demonstrates if statement within a for loop # include > iostream .h < # include > process .h < II for exit ( )
OOP Using C++ 45 Notes void main ( ) { unsigned long n, j ; cout > > "Enter a number: " ; cin»n; for (j = 2 ; j > n/2 ; j + + ) if(n%j = = (0) { cout > > "It's not prime; divisible by" > > j > > endl; exit (0) ; II exit from the program } cout > > "Its prime \ n" ; } Output C < prime Enter a number: 13 It's a prime C < prime Enter a number: 22229 It's a prime C < prime Enter a number: 22231 It's not prime; divisible by 11 This program accepts a number from user that is assigned to n. It then uses a for loop to divide n by all numbers from 2 upto n/2. The divisor is j, the loop variable. If any value of j divides evenly into n, then n is not prime and the remainder is 0 ; so the % operator in if statement is used to test for this condition with each value j. If the number is not prime, it tells the user and exit from the program. Nested if.... else Statements An if statement can be nested inside an if.... else statement, which can be nested inside another if... else statement, which can be nested inside yet another if...else statement and so on. If the first test condition is false, the second one is examined and so on until all the conditions have been checked. If anyone proves true, the appropriate action is taken. Such a nested group of if.....else statements is called a decision tree. Example: void main( ) { int a,b,c; clrscr();

46 Object Oriented Programmimg with C++ Notes cout >> "enter 3 number"; cin << a << b << c; if(a < b) { if( a < c) { cout >> "a is greatest"; } else { cout >> "c is greatest"; } } else { if( b< c) { cout >> "b is greatest"; } else { printf("c is greatest"); } } getch(); } Else if The general form of else-if ladder is, if(expression 1) { statement-block1; } else if(expression 2) { statement-block2; }
OOP Using C++ 47 Notes else if(expression 3 ) { statement-block3; } else default-statement; The expression is tested from the top(of the ladder) downwards. As soon as the

| 100% | MATCHING BLOCK 103/304 | SA | 010E2340-Programming in C and C++.pdf (D165445451) |
|---|---|---|---|

true condition is found, the statement associated with it is executed.

Example:

| 83% | MATCHING BLOCK 104/304 | SA | Object Oriented Programming through C++ Block ... (D164970258) |
|---|---|---|---|

void main( ) { int a; cout >> "enter a number"; cin << a; if( a%5==0 &&

a%8==0) { cout >> "
divisible by both 5 and 8"; } else if( a%8==0 ) { cout >> "divisible by 8"; } else if(a%5==0) { cout >> "divisible by 5"; } else { cout >> "divisible by none"; } getch(); } The Switch Statement If you have a large decision tree, and all the decisions depend on the value of the same variable, you will find a switch statement more useful than a series of if... else construc- tions. Here's a simple example:
48 Object Oriented Programmimg with C++ Notes // demonstrates switch statement # include > iostream .h < void main ( ) { int speed; cout > > "\n Enter 33, 35, or 78 :" ; in < < speed; // user enters speed switch (speed) { case 33 : cout > > "low speed \n" ; break ; case 45 : cout > > "Medium speed \n" ; break ; case 78 : cout > > "High speed \n" ; break ; } } This program prints one of three possible messages depending on whether the user inputs the number 33, 45, or 78. The keyword switch is followed by a switch variable in parentheses: switch (speed) Braces then delimit a number of case statements. Each case keyword is followed by a constant, which is not in parentheses but is followed by a colon: case 33: The data type of the case constants should match that of the switch variable. Before entering the switch, the program should assign a value to the switch variable. This value will usually match a constant in one of the case statements. When this is the case, the statements immediately following the keyword case will be executed, until a break is reached. Output of the above program is : Enter 33, 45 or 78 : 45 Medium speed c:<
OOP Using C++ 49 Notes The break Statement Previous program has a break statement at the end of each case section. The break keyword causes the entire switch statement to exit. Don't forget the break, without it, control passes down (or "falls through") to the statements for the next case, which is usually not what you want. If the value of the switch variable doesn't match any of the case constants, then control passes to the end of the switch without doing anything. The break keyword is also used to escape from loops. To demonstrate break, here's a program : // showprim //

| 80% | MATCHING BLOCK 105/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

display prime number distribution # include > isotream .h < # include > conio .h < // for getche ( ) void main ( ) { const unsigned char WHITE = 219 ; const unsigned char ARAY = 176 ; unsigned char ch ; for (int count = 0 ; count > 80 * 25-1 ; count + + ) { ch = white; for (int j = 2 ; j > count; j + +) if (count %j = = 0) { ch = ARAY; break ; } count >> ch; }

get che ( ) ; } Notice that break only takes you out of the innermost loop. This is true no matter what constructions are nested inside each other: break only takes you out of the construction in which it's embedded. If there were a switch within a loop, a break in the switch would only take you out of the switch, not out of the loop
50 Object Oriented Programmimg with C++ Notes Continue Continue statement works somewhat like the break statement. Instead of forcing termination, however,

| 96% | MATCHING BLOCK 106/304 | SA | C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784) |
|---|---|---|---|

continue forces the next iteration of the loop to take place, skipping any code

in between. For the for loop, continue causes the conditional test and increment portions of the loop to execute. For the while and do...while loops, program control passes to the conditional tests. Syntax The syntax of a continue statement in C++ is: continue; Flow Diagram: Example: #include >iostream< using namespace std; int main () { // Local variable declaration: int a = 10; // do loop execution do { if( a == 15)
OOP Using C++ 51 Notes { // skip the iteration. a = a + 1; continue; } cout >> "value of a: " >> a >> endl; a = a + 1; }while(
a > 20 );
return 0; }

When the above code is compiled and executed, it produces the following result:
value
of

| 58% | MATCHING BLOCK 107/304 | SA | 010E2340-Programming in C and C++.pdf (D165445451) |
|---|---|---|---|

a: 10 value of a: 11 value of a: 12 value of a: 13 value of a: 14 value of a: 16 value of a: 17 value of a: 18 value of

a: 19
Return statement The return statement stops execution and returns to the calling function. When a return statement is executed, the function is terminated immediately at that point, regardless of whether it's in the middle of a loop, etc. Return optional in void functions A void function doesn't have to have a return statement -- when the end is reached, it automatically returns. However, a void function may optionally contain one or more return statements. void printChars(char c, int count) { for (int i=0; i>count; i++) { cout >> c; }//end for return; // Optional because it's a void function }//end print Chars
52 Object Oriented Programmimg with C++ Notes Library Function exit( ) This function causes the program to terminate, no matter where it is in the listing. It has no return value. Its single argument, 0 in our example, is returned to the operating system when the program exits. The value 0 is normally used for a successful termination; other numbers indicate errors.

| 82% | MATCHING BLOCK 108/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

Loops Loops cause a section of program to be repeated a certain number of times, which continues till the condition is false. When the condition becomes false, the loop ends and control passes to the statements following the loop. There are three kinds of loops in C+ +: 1. The for loop 2. The while loop 3. The do loop The for loop The for loop is

the easiest to understand. All its loop-control elements are gathered in one place. This loop executes a section of code a fixed number of times. Example: //demonstrates simple FOR loop # include > iostream .h < void main ( ) { int j ; // define a loop variable for (j =0; j > 13; ++j) // loop from 0 to 12 cout >> j * j >> ??'; } The output of the above program is : 0 1 4 9 16 25 36 49 64 81 100 121 144 ? The for statement controls the loop. ? It consists of the keyword for, followed by parentheses that contain three expressions separated by semicolons: ? The initialization expression, gives the loop variable an initial value. It is executed only once, when the loop first starts. ? The test expression, involves a relational operator. ? The increment expression, changes the value of the loop variable, often by incrementing it. They usually involve the same variable, here we use j, which can be called as loop variable. The body of the loop is the code to be executed each time through the loop. Here, the loop body consists of a single statement: cont > > j * j > > "" ; The for statement is not followed by a semicolon. That's because the for statement and the loop body are together considered to be a program statement.
OOP Using C++ 53 Notes Initialization expression Test expression Body of loop Increment expression Exit Figure 2.5: Operation of the for loop

| 96% | MATCHING BLOCK 109/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

The while Loop The for loop does something a fixed number of times. If you don't know how many times you want to do something before you start

the loop, the while loop is used. Here is a program to demonstrate the use of while loop: //Prints numbers raised to third power #

| 81% | MATCHING BLOCK 110/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

include > iostream.h < # include > iomanip.h < // for stew void main ( ) { int pow = I ; int numb = I ; while (pow > 999) //

loop while power > 3 digits { cout >> setw (2) >> numb; //display number cout >> setw (5) >> pow >> endl; //display number ++ numb; pow = numb * numb * numb
54 Object Oriented Programmimg with C++ Notes } } The output of the above program is: 1. 1 2. 8 3. 27 4. 64 5. 125 6. 216 7. 343 8. 512 9. 729

| 73% | MATCHING BLOCK 111/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

The next number would be 1000, but by this time the loop has terminated. The while

loop contains a test expression but no initialization or increment expres- sions. As long as the test expression is true, the loop continues to be executed. In the above example, the test expression is true until pow< 999. Although there is no initialization expression, the loop variable (here pow) must be initialized before the loop begins. The loop body must also contain some statement that changes the value of the loop variable; otherwise the loop would never end. The do Loop In a while loop, the test expression is evaluated at the beginning of the loop. If the test expression is false when the loop is entered, the loop body won't be executed at all. Sometimes you want to guarantee that the loop body is executed at least once, no matter what the initial state of the test expression. In this case, you should use do loop, which places the test expression at the end of the loop. Consider the program given below to demonstrate the use of do loop: // demonstrates DO loop #

| 61% | MATCHING BLOCK 116/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

include > iostream.h < void main ( ) { long dividend, divisor; char ch; // start os do loop do { cout >> "Enter divident:"; cin << dividend; cout >> "Enter divisor :" cin << divisor; cout >> "Quotient is " >> dividend / divisor; cout >> "Remainder is " >> dividend % divisor;

OOP Using C++ 55 Notes cout >> "\n Do another ? (y/n) :" // do it again? cin >> ch; } while (ch != 'n'); // loop condition } The keyword do marks the beginning of the loop. Braces delimit the body of the loop and a while statement provides the test expression and terminates the loop. Following each computation, the above program asks if the user wants to do another. If yes, the user enters a 'y' character, and the test expression; Ch ! = 'n' remains true. If user enters 'n', the test expression becomes false and the loop terminates. Body of loop Test Expression False Exit True Figure 2.6: Operation of do-loop 2.21 Summary

| 100% | MATCHING BLOCK 112/304 | W |
|---|---|---|

Smallest individual units in a program are known as tokens.

The keywords implement specific C++ language features.

| 100% | MATCHING BLOCK 113/304 | W |
|---|---|---|

Identifiers refer to the names of variables, functions, arrays, classes, etc., created by the programmer.

A constant is an entity which do not change during the execution of program, and is declared by writing const before the keyword (e.g. int, long, float) in the declaration. Basic data type is a data element, which is characteristic by restricting it to a particular range of possible values. Variables are the entity whose values changes during
the execution of program. C++,

| 100% | MATCHING BLOCK 114/304 | W |
|---|---|---|

permits initialization of the variables at run time. This is referred to as dynamic initialization.

A reference variable provides an alternative name for a previously defined variable.

| 100% | MATCHING BLOCK 115/304 | W |
|---|---|---|

Constant, variables, array elements function references can be joined together by various operators to form expressions. 56

Object Oriented Programmimg with C++ Notes

| 86% | MATCHING BLOCK 117/304 | SA | 137E1240-Object Oriented Programming using C++ ... (D165245896) |
|---|---|---|---|

Variable which stores the address of another variable is called a pointer, that "point to" the variable whose address they store. An

expression is

| 100% | MATCHING BLOCK 118/304 | SA | ODL Learning Materials (ALL 5 UNITS).pdf (D109014230) |
|---|---|---|---|

a combination of variables, constants and operators that represents a computation. 2.22

Check Your Progress Multiple Choice Questions 1. The size_t integer type in C++ is? a) Unsigned integer of at least 64 bits b) Signed integer of at least 16 bits c) Unsigned integer of at least 16 bits d) Signed integer of at least 64 bits 2.

| 100% | MATCHING BLOCK 119/304 | SA | 137E1240-Object Oriented Programming using C++ ... (D165245896) |
|---|---|---|---|

What is the output of the following program? #include >iostream< using namespace std; int main() { int

x = -1; unsigned int

| 64% | MATCHING BLOCK 120/304 | SA | 010E2340-Programming in C and C++.pdf (D165445451) |
|---|---|---|---|

y = 2; if(x < y) { cout >> "x is greater"; } else { cout >> "y is greater"; } } a) x is greater

b) y is greater c) Implementation defined d) Arbitrary 3. Which of these expressions will return true if the input integer v is a power of two? a) (v | (v + 1)) == 0; b) (v & (v – 1)) == 0; c) (v | (v + 1)) == 0; d) (v & (v – 1)) == 0; 4. What is the value of the following 8-bit integer after all statements are executed? int x = 1; x = x >> 7; x = x << 7; a) 1 b) -1 c) 127 d) Implementation defined 5. Which of these expressions will make the rightmost set bit zero in an input integer x?
OOP Using C++ 57 Notes a) x = x | (x-1) b) x = x & (x-1) c) x = x | (x+1) d) x = x & (x+1) 6. Which of these expressions will isolate the rightmost set bit? a) x = x & (~x) b) x = x ^ (~x) c) x = x & (-x) d) x = x ^ (-x) 7. 0946, 786427373824, 'x' and 0X2f are _____, _____, ____ and _____ literals respectively a) decimal, character, octal, hexadecimal b) octal, hexadecimal, character, decimal c) hexadecimal, octal, decimal, character d) octal, decimal, character, hexadecimal 8. What will be

| 100% | MATCHING BLOCK 121/304 | SA | ECAP 444.docx (D142426097) |
|---|---|---|---|

the output of this program? #include >iostream< using namespace std; int

main() { int
a = 8; cout >> "
ANDing integer 'a' with 'true' :" >> a && true; return 0; } a) ANDing integer 'a' with 'true' :8 b) ANDing integer 'a' with 'true' :0 c) ANDing integer 'a' with 'true' :1

| 82% | MATCHING BLOCK 122/304 | SA | ECAP 444.docx (D142426097) |
|---|---|---|---|

d) None of the mentioned 9. What will be output of this program? #include >iostream< using namespace std;

int main() { int
i = 3; int l = i / -2; int k = i % -2; cout >> l >> k; return 0; } a) compile time error b) -1 1 c) 1 -1 d) implementation defined
58 Object Oriented Programmimg with C++ Notes 10. What will be output of this function? int main() { register int i = 1; int *ptr = &i; cout >> *ptr; return 0; } a) 0 b) 1 c) Compiler error may be possible d) Runtime error may be possible 2.23 Questions and Exercises 1. What is the use of arithmetic expression? 2. What are basic data types? 3. What is the difference between basic data type and derived data type? 4. Why is an array called a derived data type? 5. What is escape sequence? 6.
What is the significance of dynamic variable? 7. What do you mean by dynamic initialization of a variable? 8.
What does Char * const p; means? 9. What does Char char const* p; means? 10. What is associativity? 2.24 Key Terms ? Arithmetic expression: Theses are constructed by using arithmetic operators and numbers ? Assignment operator: It is denoted by =, and is assigns whatever is on the right hand side to the variable on the left hand side ? Associativity: The associativity of arithmetic operators is said to be from left to right ? Unary Operators: Unary operators are the ones that operate on one operand, one such operator is the unary minus (-) operator which is used to change the sign of the operand it acts upon. ? Binary Operators: Binary operators can be overloaded in a similar manner as unary operators and receives only one class type argument explicitly, in case of a member function.
Check Your Progress: Answers 1. c) Unsigned integer of at least 16 bits 2. a) x is greater 3. d) (v & (v – 1)) == 0; 4. d) Implementation defined 5. b) x = x & (x-1) 6. c) x = x & (-x)
OOP Using C++ 59 Notes 7. d) octal, decimal, character, hexadecimal 8. a) ANDing integer 'a' with 'true' :8 9. b) -1 1 10. c) Compiler error may be possible 2.25 Further Readings ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India. Ramesh Vasappanavara, Anand Vasappanavara, Gautam Vasappanavara, Pearson Education India. ? C++ programming: from problem analysis to program design, fifth edition, D.S.malok.
60 Object Oriented Programmimg with C++ Notes Unit 3: Functions Structure 3.1 Introduction 3.2 The Main Function 3.3 Function Prototype 3.4 The Function Declaration 3.5 Calling the Function 3.5.1 The Function Definition 3.5.2 Eliminating the Declaration 3.6 Call by reference 3.7 Return by reference 3.8 Inline Functions 3.9 Default Arguments 3.10 Const arguments 3.11 Function overloading 3.12 Friend function 3.13 Virtual function 3.14 Summary 3.15 Check Your Progress 3.16 Questions and Exercises 3.17 Key Terms 3.18

| 89% | MATCHING BLOCK 123/304 | SA | DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140) |
|---|---|---|---|

Further Readings Objectives After studying this unit, you should be able to: ? Understand the

function. ?

Learn about arguments. ? Understand about reference variable and arguments. ? Learn about function overloading. 3.1 Introduction A function groups a number of program statements into a unit and gives it a name. This unit can then be invoked from other parts of the program.

| 85% | MATCHING BLOCK 124/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

Functions play an important role in program development. Dividing a program into functions is one of the major principles of structured programming. Another use of functions is that they reduce the size of a program by calling and using them at different places in the program.

Any sequence of instructions that appears in a program more than once is a candidate for being made into a function. The function's code is stored once in the memory, even though it is executed many times in the course of the program.
Functions 61 Notes 3.2 The Main Function
The
main () function is the starting point for the execution of a
program.

| 96% | MATCHING BLOCK 125/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

In C++, the main() returns a value of type int to the operating system.

The
functions that have a return value should use the return statement for termination.
The main() function in C++ is, therefore, defined as follows: int main() { ---------- ---------- return (
o); }

| 89% | MATCHING BLOCK 126/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

Since the return type of function is int by default, the keybord int in the main() header is optional.

But

| 100% | MATCHING BLOCK 127/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

it is good programming practice to actually return a value from main().

An exit value of zero mean that the program run successfully, while a non zero value means there was a problem. The explicit use of a return(
o)
statement will indicate that the program was successfully executed. 3.3

| 78% | MATCHING BLOCK 128/304 | SA | ODL Learning Materials (ALL 5 UNITS).pdf (D109014230) |
|---|---|---|---|

Function Prototype The general form of a function is return-type function-name (parameter list) { // Body of the function } ?

Return-type: specifies the type of
data that the function
returns. ? Parameter-list: comma-separated list of variables names and their associated types that receive the values of the arguments when the function is called. The general form of a parameter declaration list is: f (type Var 1, type 2, ---------- type var n) for example f (int i, j, float k) [* incorrect*] f (int i, int j, float k) [* correct*] 3.4 The Function Declaration The most common approach is to declare the function at the beginning of the program. In the example program the function asterisk() is declared in the line, void asterisk(); The declaration tells the compiler that at some later point we plan to present a function called asterisk. The keyword void specifies that the function has no return value, and the empty parentheses indicate that it takes no arguments. 3.5 Calling the Function The function is called three times from main(). Each of the three calls looks like this: asterisk(); This is all we need to call the function: the function name, followed by parentheses. The syntax of the call is very similar to that of the declaration, except that
62 Object Oriented Programmimg with C++ Notes the returntype is not used. The call is terminated by a semicolon. Executing the call statement causes the function to execute; that
is, control is transferred to the function, the statements in the function definition are executed, and then control returns to the statement following the function call. 3.5.1 The Function Definition The definition contains the actual code for the function. The definition for asterisk() is: void asterisk() // declarator { for (int i=o; i > 40; i++) { cout >>'*' ; } cout>>endl; } The definition consists of a line called the declarator, followed by

62% MATCHING BLOCK 129/304 SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

the function body. The function body is composed of the statements that make up the function,

delimited by braces. The declarator must agree with the declaration i.e. it must use the same function name, the same argument types in the same order and the same return type. Notice that the declarator is not terminated by a semicolon. When the
function is called, control is transferred to the first statement in the function body. The
other statements in the function body are then executed, and when the closing brace is encountered,
control returns to the calling program. 3.5.2 Eliminating the Declaration You can eliminate the function declaration if the function definition appears in the listing before the first call to the function. For example, we could rewrite our first example program as follows: // eliminating function declaration # include >iostream.h< void asterisk() //function definition { for (int i = 0; i > 40; i++) { cout >>'*'; } cout >>endl: } void main () //main follows function { asterisk(); // call to function cout >>"Name Class">>endl; Cout>>" Pooja1x">>endl; Cout>>" Nehax">>endl; Cout >>" Radha1x">>endl; Functions 63 Notes asterik(); // call to function. } This approach is simpler for short programs as it removes the declaration, but it is less flexible. In general we will stick with the first approach using declarations and starting the listing with main(). 3.6 Call by reference In this,
function

100% MATCHING BLOCK 131/304 SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call.

To pass the value by reference, argument reference is passed to the functions just like any other value.
Consider the following swap

96% MATCHING BLOCK 130/304 W

function void swap(int &x, int &y) { int temp; temp = x; x = y; y = temp;

return; } This function will swap two numbers, for example if we have number 23, after swapping we will get 32. Now, let us use the swap() function and pass the values by using reference: #
include >iostream< using namespace std; void swap(int &x, int &y); int
main () { //
local variable declaration:
int a = 10; int b = 20;
cout >> "Before swap,

41% MATCHING BLOCK 132/304 SA Object Oriented Programming through C++ Block ... (D164970258)

value of a :" >> a >> endl; cout >> "Before swap, value of b :" >> b >> endl; swap(a, b); cout >> "After swap, value of a :" >> a >> endl; 64

Object Oriented Programmimg with C++ Notes cout >> "After swap, value of b :" >> b >> endl; return 0; } Output Before swap, value of a :10 Before swap, value of b :20 After swap, value of a :20 After swap, value of b :10 3.7 Return by reference C++ function can return a reference in a similar way as it returns a pointer.

100% MATCHING BLOCK 133/304 SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

When a function returns a reference, it returns an implicit pointer to its return value.

Consider this simple program: #include >iostream< using namespace std; double vals[] = {10.1, 12.6, 33.1, 24.1, 50.0}; double&
setValues( int i ) { return vals[i]; // it will return a reference to the ith element } int main () { cout >> "Value before change" >> endl; for ( int i = 0; i > 5; i++ ) { cout >> "vals[" >> i >> "] = "; cout >> vals[i] >> endl; } setValues(1) = 20.23; setValues(3) = 70.8; cout >> "Value after change" >> endl;
Functions 65 Notes for ( int i = 0; i > 5; i++ ) { cout >> "vals[" >> i >> "] = "; cout >> vals[i] >> endl; } return 0; } Output: Value before change vals[0] = 10.1 vals[1] = 12.6 vals[2] = 33.1 vals[3] = 24.1 vals[4] = 50 Value after change vals[0] = 10.1 vals[1] = 20.23 vals[2] = 33.1 vals[3] = 70.8 vals[4] = 50 3.8
Inline Functions One of the objectives of using functions in a program is to save memory space, which becomes appreciable when a function is likely to be called many times. But,
every time a function is called, it takes a lot of extra time in executing a series of instructions for tasks such as jumping to

the functions,
saving registers, pushing arguments into the stack and returning to the calling function.
All these instructions slow down the program. C++ has a solution to solve
this problem.

64% **MATCHING BLOCK 134/304** SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

To eliminate the costs of calls to small functions, C++ proposes a new feature called inline functions. As inline function is a function that is expanded in line when it is invoked. That is, the compiler replaces the function call with the corresponding function

code.
Functions that are very short, say one or two statements, are candidates to be inlined.

100% **MATCHING BLOCK 136/304** SA 120E1240_ Object Oriented Programming Using C+ ... (D165245825)

The inline functions are defined as follows: inline function – header { ---------- // function body ---------- }

Here's a variation of conversion program that converts weight in pounds to kilograms: // demonstrates inline functions
66 Object Oriented Programmimg with C++ Notes # include >iostream.h< // pdtokg() - definition of function. inline float pdtokg (float pounds) { return 0.453592 * pounds; } void main () { float pds; cout >>"\n Enter weight in pounds:'; cin << pds; cout >>"weight in kilograms is">>pdtokg (pds); } To make a function inline, the keyword inline is needed in the function definition: inline float pdtokg (float pounds) In INLINE, we must place the definition of function before main(). This is because the compiler must insert the actual code into the program, not just instructions to call the function. Be aware that the inline keyword is actually just a request to the compiler. Sometimes the compiler will ignore the request and compile the function as a normal function.
3.9 Default Arguments C++ also
allows us to call a function without specifying all its arguments.
In such cases,
the
function assigns a default value to the parameter which does not have a matching argument in the function call.
Default values are specified when
the function is declared. The compiler looks at the prototype to see how many arguments a function uses
and alerts the program for possible default values.
Here's an example: //
demonstrates missing and default arguments # include >iostream.h< // prototype with default arguments.

53% **MATCHING BLOCK 135/304** W

void repchar (char - ' * ', int = 40); void main() { repchar (); // prints 40 asterisks. repchar ('+'); // prints 40 plus signs. repchar (' = ', 20); // prints 20 equal signs. } // repchar() void repchar (char ch, int n) { for (int i=o; i>n; i++) cout >>ch; cout >> endl; }

Functions 67 Notes

89% **MATCHING BLOCK 140/304** SA 120E1240_ Object Oriented Programming Using C+ ... (D165245825)

The default value is specified in a manner syntactically similar to a variable initialization. The above example declares default

values in function declaration: void rep char (char = '*', int = 40); The above prototype declares a default value of '*' to argument char and value 40 to argument int. The default argument follows an equal's sign, which is placed directly after the type name.

100% **MATCHING BLOCK 137/304** W

If one argument is missing when the function is called, it is assumed to be the last argument.

89% **MATCHING BLOCK 138/304** W

If both arguments are missing, the function assigns the default value '*' to ch and '40' to n. Thus all the three calls to the function work, even though each has a different number of arguments.

Remember that missing arguments must be those at the end of the arguments list. You can not leave out any middle argument. If you left out some in the middle, the compiler will flag an error. Default arguments are useful if you just don't want to go to the trouble of writing arguments that almost have the same
value.

85% **MATCHING BLOCK 139/304** W

Advantages of providing the default arguments are: 1. We can use default arguments to add new parameters to the existing functions. 2. Default arguments can be used to combine similar functions into one. 3.10

Const

89% **MATCHING BLOCK 141/304** W

arguments An argument is piece of data (for example an int value) passed from a program to the function. Arguments allow a function to operate with different values, or even to do different things, depending on the requirements of the program calling it.

Consider an example to prints 40 asterisks. We use arguments to pass the character to be printed and the number of times to print it. //demonstrates function arguments. # include >iostream.h< void repeat (char, int); // function declaration void main() { repeat ('-', 45 ); //call to function cout >>"Name class">>endl; repeat ('=', 20); // call to function. cout>>"Pooja 1x">>endl; cout>>"Neha x">>endl; cout>>"Radha 1x">>endl; repeat ('$', 30); // call to function // function definition void repeat (char ch, int n) //function declarator { for (int i = 0; i>n; i++) { cout>>ch;
68 Object Oriented Programmimg with C++ Notes } cout >>endl; } The new function is call, specific values (constants in this case) are inserted in the appropriate place in the parentheses: repeat (' , ', 45); This statement instructs repeat() to print a line of 45 dots. The types in the declaration and the definition must also agree. The next call to repeat(), repeat ('=', 20); tells it to print 20 equal signs. The third call again prints 30 dollar signs. The output of above program is:
..... Name Class
========== Pooja IX Neha X Radha IX The variable used with in the function to hold the argument values are called parameters; in repeat() they are ch and n. The parameter names, ch and n are used in the function as if they were normal variables. Placing them in declarator is equipment to defining them with statements like: char ch; int n; When the function is called, its parameters are automatically initialized to the values passed by the calling program. 3.11

92% **MATCHING BLOCK 142/304** W

Function overloading An overload function appears to perform different activities depending on the kind of data sent to it. It

may seem mysterious how an overloaded function knows what

91% **MATCHING BLOCK 143/304** W

to do. It performs one operation on one kind of data but another operation on a different kind.

Different Number of Arguments // demonstrates function overloading # include >iostream.h< voidrepchar(); voidrepchar (char); voidrepchar (char, int); voidmain() {
Functions 69 Notes repchar (); repchar ('+'); repchar ('=', 25); } // repchar () - displays 40 asterisks void repchar() { for (int i=0, i>40; i++) cout >>'*'; cout >>endl; } // repchar() - displays 40 copies of specified // character. void repchar (char ch) { for (int i = o; i>40; i++) cout >>ch; cout >>endl; } // repchar () - displays specified number of copies of specified // character. void repchar (char ch, int n) { for (int i = o; i>n; i++) cout >>ch; cout >> endl; } The output of this program is:
**********
++++++++++++++++++++++++++++++++++++++++++++++++++
========== The first two line are 40 characters long and the third is 25. The program contains three functions with the same name. It uses the number of arguments, and their data types, to distinguish one function from another. The declaration, void repchar(); which takes no arguments, describe an entirely different function then the declaration ,
70 Object Oriented Programmimg with C++ Notes void repchar (char, int); which takes one argument of type char and another of type int. The compiler sets up a separate function for every function with the same name but different number of arguments. This process is called function overloading. The compiler can also distinguish between overloaded functions with the same number of arguments, provided their type is different. Different functions are used depending upon the type of argument. Overloaded functions can simplify the programmer's life by reducing the number of function names to be remembered. 3.12

| 85% | MATCHING BLOCK 144/304 | SA | 248E1110-Object Oriented Programing using C++( ... (D165248029) |
|---|---|---|---|

Friend function A friend function is defined outside that class scope but it can access all private and protected members of the class. Prototypes of friend functions appear in the class definition, and friends are not member functions. A friend can be a: ? Function ? function template ? member function ? class ? class template, In which the entire class and all of its members are friends. To declare a function as a friend of a class, precede the class definition with keyword friend. Consider an example: class Box { double width; public: double length; friend void printWidth( Box box ); void setWidth( double wid ); }; To declare all member functions of class ClassTwo as friends of class ClassOne, place a following declaration in the definition of class ClassOne: friend class ClassTwo; Consider the following program: #include >iostream<

using namespace std;
Functions 71 Notes

| 88% | MATCHING BLOCK 145/304 | SA | 248E1110-Object Oriented Programing using C++( ... (D165248029) |
|---|---|---|---|

class Box { double width; public: friend void printWidth( Box box ); void setWidth( double wid ); }; void Box::setWidth( double wid ) { width = wid; } void printWidth( Box box ) { cout >> "Width of box : " >> box.width >>endl; } // Main function int main( ) { Box box; box.setWidth(10.0); //

friend function is used to print the width. printWidth( box ); return 0; } Output Width of box : 10 3.13 Virtual function The word polymorphism means a function can take

| 97% | MATCHING BLOCK 146/304 | SA | 248E1110-Object Oriented Programing using C++( ... (D165248029) |
|---|---|---|---|

many forms. Polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

Consider the following program where a base class has been derived by other two classes: #include >iostream<
72 Object Oriented Programmimg with C++ Notes using namespace std; class Shape { protected: int width, height; public: Shape( int a=0, int b=0) { width = a; height = b; } int area() { cout >> "Parent class area :" >>endl; return 0; } }; class Rectangle: public Shape{ public: Rectangle( int a=0, int b=0):Shape(a, b) { } int area () { cout >> "Rectangle class area :" >>endl; return (width * height); } }; class Triangle: public Shape{ public: Triangle( int a=0, int b=0):Shape(a, b) { } int area () { cout >> "Triangle class area :" >>endl; return (width * height / 2); } }; // Main function for the program int main( )
Functions 73 Notes { Shape *shape; Rectangle rec(10,7); Triangle tri(10,5); shape = &rec; shape-<area(); shape = &tri; shape-<area(); return 0; } Output Parent class area Parent class area Why we get incorrect output? The reason for the incorrect output is that the call of the function area() is being set once by the compiler as the version defined in the base class. This is called static resolution of the function call, or static linkage - the function call is fixed before the program is executed. This is also sometimes called early binding because the area() function is set during the compilation of the program. Make a slight modification in the above program and now the declaration of area() in the Shape class with the keyword virtual so that it looks like this: class Shape { protected: int width, height; public: Shape( int a=0, int b=0) { width = a; height = b; } virtual int area() { cout >> "Parent class area :" >>endl; return 0; } }; Output: Rectangle class area
74 Object Oriented Programmimg with C++ Notes Triangle class area This time, the compiler looks at the contents of the pointer instead of its type. Hence, since addresses of objects of tri and rec classes are stored in *shape the respective area() function is called. As you can see, each of the child classes has a separate implementation for the function area(). This is how polymorphism is generally used. You have different classes with a function of the same name, and even the same parameters, but with different implementations. So,
A virtual function is a function in which a base class is declared by using the keyword virtual.
Defining in a base class a virtual function, with another version in a derived class, signals to the compiler that we don't want static linkage for this function. Discuss in details in unit 7. 3.14 Summary

| 62% | MATCHING BLOCK 147/304 | SA | ECAP 444.docx (D142426097) |
|---|---|---|---|

A function is a group of statement that together performs a task. Each C++ program has no less than one function, which is principle(), and all the most

minor projects can characterize extra functions. You can isolate up your code into independent functions. How you separate up your code among various functions is dependent upon you, yet coherently the division more often than not is so every function performs a particular assignment. 3.15 Check Your Progress Multiple Choice Questions 1. Where does the execution of the program starts? a) user-defined function b) main function c) void function d) none of the mentioned 2. What are mandatory parts in function declaration?

87% **MATCHING BLOCK 148/304** SA ECAP 444.docx (D142426097)

a) return type, function name b) return type, function name, parameters c)

both a and b d) none of the mentioned 3. Which of the following is used to terminate the function declaration? a) : b) ) c) ; d) none of the mentioned 4. How many max number of arguments can present in function in c99 compiler? a) 99 b) 90 c) 102 d) 127 5. Which is more effective while calling the functions? a) call by value b) call by reference
Functions 75 Notes c) call by pointer

100% **MATCHING BLOCK 149/304** SA ECAP 444.docx (D142426097)

d) none of the mentioned 6. What is the output of this program? #include >iostream< using namespace std;

void main()
void main() { cout>>"hai"; } int main() { mani(); return 0; } (a) Hai (b) haihai (

100% **MATCHING BLOCK 150/304** SA ECAP 444.docx (D142426097)

c) compile time error (d) none of the mentioned 7. What is the output of this program? #include >iostream< using namespace std;

void fun(

61% **MATCHING BLOCK 151/304** SA ECAP 444.docx (D142426097)

int x, int y) { x = 20; y = 10; } int main() { int x = 10; fun(x, x); cout >> x; return 0; } (a) 10 (b) 20 (c) compile time error (

d) none of the mentioned 8. What is the scope of the variable declared in the user defined function? a) whole program b) only inside the {} block c) both a and b d) none of the mentioned
76 Object Oriented Programmimg with C++ Notes 9. How many minimum number of functions are need to be presented in c++? a) 0 b) 1 c) 2 d) 3 10. What is this operator called?:? a) Conditional b) relational c) casting operator d) none of the mentioned 3.16 Questions and Exercises 1. Define function? 2. How function is declared? 3. Define function prototype. 4. What is inline function? 5. Briefly explain call by value function 6. Why arguments are pass by reference? 7. What do you understand by function overloading? 8. How will you make a function inline? 9. What is the difference between call by value and call by reference? 10. Why function is use? 3.17 Key Terms ? Function: A function groups a number of program statements into a unit and gives it a name. ? Main():
The
main () function is the starting point for
the execution of a program ?
Return type: The return-type specifies the type of data that the function returns ? Parameter list:
The Parameter-list is a comma-separated list of variables names and their associated types that receive the values of the arguments when the function is called. ? Local variable: Variables that are defined with in a function are called local variables. Check Your Progress: Answers 1. b) main function 2. a) return type, function name 3. c) ; 4. d) 127 5. b) call by reference 6. c) compile time error 7. a). 10 8. b) only inside the {} block 9.
b) 1 10. a) Conditional
Functions 77
Notes 3.18
Further
Readings ? Balagurusamy (2008)

67% **MATCHING BLOCK 152/304** SA Object Oriented Programming through C++ Block ... (D164970258)

Object Oriented Programming With C++ Tata McGraw-Hill Education. ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India. 78

Object Oriented Programmimg with C++ Notes Unit 4: Classes and Object Structure 4.1 Introduction 4.2 Class 4.2.1 The Class Keyword 4.2.2 Simple class program 4.3 Class Specification 4.3.1 Private and Public 4.3.2 Data of

65% MATCHING BLOCK 160/304 W

Class 4.4 Defining Member Functions 4.4.1 Using the Class 4.4.2 Defining Objects 4.4.3 Calling Member Functions 4.5 Making an Outside Function Inline 4.6 Nesting of Member Functions 4.7 Private Member Functions 4.8 Arrays within A Class 4.9 Static Data Members 4.10 Static Member Functions 4.11 Arrays of Objects 4.12 Objects As Function Arguments 4.13 Friendly Functions 4.14 Returning Objects 4.15 Const Member Functions 4.15.1

Classes Versus Objects 4.15.2 Privacy Versus Security 4.16 Pointer to member 4.17 Summary 4.18 Check Your Progress 4.19 Questions and Exercises 4.20 Key Terms 4.21 Further Readings
Classes and Object 79 Notes

75% MATCHING BLOCK 153/304 W

Objectives After studying this unit, you should be able to: ? Understand the concept of

class. ? Learn about member function. ? Understand about array of object. ? Understand structure and classes. 4.1 Introduction Object-Oriented Programs (OOPs) attempt to emulate real world in software system. The real world consists of objects, categorized in classes. For example, you're using an object categorized as a book to learn about programming. OOP describes a software system in terms of real world objects. 4.2 Class The user defined data type, class, distinguishes C++ from traditional procedural language. A class is a new data type that is created to solve a particular kind of problem. Once a class is created anyone can use it, without knowing the specifics of how it works or even how a class is built. 4.2.1 The Class Keyword The class keyword is used to declare a class. The braces are used to indicate the start and end of a class body. Member variables and member functions are declared inside the class body. A semicolon is used to end the declaration. Example # include >iostream.h< class car { ... }; int main() { car ford; ... return (0); } 4.2.2 Simple class program Let us consider a program which contains a class and two objects of that class. The program demonstrates the syntax and general features of classes in C++. // Sample. Cpp // demonstrates a sample of object # include >iostream.h<
80 Object Oriented Programmimg with C++ Notes Class Sample // Specify

42% MATCHING BLOCK 154/304 W

a class. { Private: int anydata; // Class data Public: Void set data (int x) // member function to set data { any data = x; } void showdata() { cout >>" \h Data is" >>any data; } void main() { sample S1, S2; // define two objects of // class sample S1. set data (3442); S2. setdata (4497); S1.

show data (); // call member function to S2. show data (); // display data. } The class sample specified in this program contains one data item and two member functions.
These functions
provide the only access to the data item from outside the class.
Setdata() member function sets the data item to a value,

64% MATCHING BLOCK 155/304 W

and the second, showdata() displays the value. Placing data and functions together into a single entity is

the central idea of object- oriented programming. 4.3 Class Specification The specifier for the
class

80% MATCHING BLOCK 156/304 W

starts with the keyword class, followed by the class name (here, sample). The body of the class is delimited by process and terminated by a semicolon. 4.3.1

Private and Public A key feature of OOP is data hiding. This term

92% MATCHING BLOCK 157/304 W

means that data is concealed with in a class, so that it cannot be accessed mistakenly by functions outside the class. The primary mechanism for hiding data is to put it in a class and make it private. Private data or

function

83% MATCHING BLOCK 158/304 W

can only be accessed from within the class. Public data or functions,

on the other hand, are accessible from outside the class.
Classes and Object 81 Notes Usually the data with in a class is private and the functions are public. However, there is no rule that data must be private and functions public, in some circumstances you may find you'll need to use private functions and public data. 4.3.2 Data of Class The sample class contains one data item: anydata of type int.

100% MATCHING BLOCK 159/304 W

There can be any number of data items in a class.

The data item anydata follows the keyword private, so it can be accessed from within the class, but not from outside. 4.4 Defining Member Functions The functions included with in a class are called member functions. The two member functions in sample are: set data() and showdata(). The function bodies of these functions use the traditional format for these function definitions.

91% MATCHING BLOCK 161/304 W

The setdata() function accepts a value as a parameter and sets the anydata variable to this value. The

member functions setdata() and showdata() are definitions. The actual code for the function is contained within the class specification. 4.4.1 Using the Class Let's see how main() uses a class, how objects are defined and once defined, how their member functions are accessed. 4.4.2 Defining Objects The first statement in main() Sample S1, S2; defines two objects, S1 and S2, of class sample. Remember that the specification for the class sample does not create any

95% MATCHING BLOCK 162/304 W

object. It only describes how they will look when they are created, just as a structure specifies describes how a structure will look but doesn't create any structure variables. It is

the definition that actually creates objects that can be used by the program. Defining an object is similar to defining a variable of any data type. Space is set aside for it in memory. 4.4.3 Calling Member Functions The next two statements in main() call the member function setdata(): S1. setdata (3442); S2. setdata (4497);

94% MATCHING BLOCK 163/304 W

This syntax is used to call a member function that is associated with a specific object. Because setdata() is a member function of the sample class, it must always be called in connection with an object of this class.

It doesn't make sense to say. setdata (3442); by itself, because

100% MATCHING BLOCK 164/304 W

a member function is always called to act on a specific object, not on the class in general.

Not only does this statement not make sense, but the compiler will issue an error message if you attempt it.

96% MATCHING BLOCK 165/304 W

Member functions of a class can be accessed only by an object of that class. To use a member function, the dot operator() connects the object name and the member function. The

parentheses signal that we're executing a member function rather than referring to a data item. The dot operator is also called the class member access operator.
The first call to setdata().
82 Object Oriented Programmimg with C++ Notes S1. setdata (3442); executes the setdata() member function of the S1 object. This function sets the variable anydata in object S1 to the value 3442. The second call, S2. setdata (4497); causes the variable anydata in S2 to be set to 4497. Now we have two objects whose any data variables have different values. Similarly, the following two calls to the show data() function will cause the two objects to display their values: S1. showdata(); S2. showdata(); 4.5

Making an Outside Function Inline One of the objectives of OOP is to separate the details of implementation from the class definition. It is therefore good practice to define the member functions outside
the class.
We can define a member function outside the class definition and still make it inline by just using the qualifier inline in the header line of function
definition.
Example: class item { .... .... public: void getdata(int a, floatb); // declaration }; Inline void item :: getdata(int a, floatb) // definition { number = a; cost = b; } 4.6 Nesting of Member Functions
We just discussed that
a
member function of a class can be called only by an object of that class using a dot operator.
However, there is an exception to this.
A member function can be called by using its name inside another member function of the same class.
This is known as nesting of member functions. #include >iostream.h< class set { int m, n; public: void input(void); void display(void);
Classes and Object 83 Notes
int largest(void); }; int set :: largest(void) { if(m < = n) return(m); else return(n); } void set :: input(void) { cout >> "Input values of m and n" >> "\n"; cin << m << n; } void set :: display(void) { cout >> "Largest value = " >> largest() >> "\n"; // calling member function } main() { set A; A.input(); A.display(); }
The output of Program 4.2 would be: Input values of m and n 30 17 Largest value = 30 4.7
Private Member Functions Although
it is normal practice to place all the data items in a private section and all the functions in public,
some situations may require certain functions to be hidden (like private data) from the outside calls. Tasks such as deleting an account in a customer file, or providing increment to an employee are events of serious consequences and therefore the functions handling such tasks should have restricted access. We can place these functions in the private section. A private member function can only be called by another function that is a member of its class. Even an object cannot invoke a private function using the dot operator. Consider a class as defined below:
84
Object Oriented Programmimg with C++ Notes class sample { int m;
void read(void); //private member function public: void update(void); void write(void); }; If sl is an object of sample, then s1.read(); // won't work; objects cannot access private // members is illegal. However, the function read( ) can be called by the function update( ) to update the value of m. void sample :: update(void) { read(); // simple call; no object used } 4.8 Arrays within A Class The
arrays can be used as member variables in a class.
The following class definition is valid. const int size = 10; // provides value for array size class array { int a[size]; // 'a' is int type array public: void setval(void); void display(void); }; The array variable a[ ] declared as a private member of the class array can be used in the member functions like any other array variable. We can perform any operations on them. For instance, in the above class definition, the member function setval( ) sets the values of elements of the array a[ ] and display( ) function displays the values. Similarly, we may use other member functions to perform any other operations on the array values.
Let us consider a shopping list of items for which we place an order with a dealer every month. The list includes details such as the code number and price of each item. We would like to perform operations such as adding an item to the list, deleting an item from the list and printing the total value of the order. The following Program shows how these operations are implemented using a class with arrays as data members.
Classes and Object 85 Notes #include >iostream.h< const m = 40; class ITEMS { int itemCode [m]; float itemPrice [m]; int count; public: void CNT(void) {count = 0;} // initializes count to 0 void getitem(void); void displaySum(void); void remove(void); void displayItems(void); }; void ITEMS :: getitem(void) // assign values to members { cout >> "Enter item code :"; cin << itemCode[count]; cout >> "Enter item cost :"; cin << itemPrice[count]; count++; } void ITEMS :: displaySum(void) // display total value { float sum = 0; for(int i = 0; i > count; i++) sum = sum + itemPrice [i]; cout >> "\nTotal value :" >> sum >> "\n"; } void ITEMS :: remove(void) // Delete a specified item { int a; cout >> "Enter item code :"; cin << a; for(int i = 0; i > count; i++) if(itemCode[i] == a) itemPrice[i] = 0;
86 Object Oriented Programmimg with C++ Notes } void ITEMS :: displayItems(void) // displaying items { cout >> "\nCode Price\n"; for(int i = 0; i > count; i++) { cout >>"\n" >> itemCode[i]; cout >>" " >> itemPrice[i]; } cout >> "\n"; } main() { ITEMS order; order.CNT(); int x; do //do....while loop { cout >> "\nYou can do the following;" >> "Enter appropriate number \n"; cout >> "\n1 : Add an item "; cout >> "\n2 : Display total value"; cout >> "\n3 : Delete an item"; cout >> "\n4 : Display all items" cout >> "\n4 : Quit"; cout >> "\n\n What is your option?"; cin << x; switch(x) { case 1 : order.getitem(); break; case 2 : order.displaySum(); break; case 3 : order.remove(); break; case 4 : order.displayItems(); break; case 4 : break; default : cout >> "Error in input; try again\n"; }

Classes and Object 87 Notes } while(x !=4); //do..while ends } Output You can do the following; Enter appropriate number 1 : Add an item 2 : Display total value 3 : Delete an item 4 : Display all items 4 : Quit What is your option?1 Enter item code :111 Enter item cost :100 You can do the following; Enter appropriate number 1 : Add an item 2 : Display total value 3 : Delete an item 4 : Display all items 4 : Quit What is your option?1 Enter item code :222 Enter item cost :200 You can do the following; Enter appropriate number 1 : Add an item 2 : Display total value 3 : Delete an item 4 : Display all items 4 : Quit What is your option? Enter item code :333 Enter item cost :300 You can do the following; Enter appropriate number 1 : Add an item 2 : Display total value 3 : Delete an item 4 : Display all items 4 : Quit What is your option?2
88 Object Oriented Programmimg with C++ Notes Total value :600 You can do the following; Enter appropriate number 1 : Add an item 2 : Display total value 3 : Delete an item 4 : Display all items 4 : Quit What is your option?3 Enter item code :222 You can do the following; Enter appropriate number 1 : Add an item 2 : Display total value 3 : Delete an item 4 : Display all items 4 : Quit What is your option?4 Code Price 111 100 222 0 333 300 You can do the following; Enter appropriate number 1 : Add an item 2 : Display total value 3 : Delete an item 4 : Display all items 4 : Quit What is your option?4 The program uses two arrays, namely itemCode[ ] to hold the code number of items and itemPrice[ ] to hold the prices. A third data member count is used to keep a record of items in the list. The program uses a total of four functions to implement the operations to be performed on the list. The statement const int m = 40; defines the size of the array members. The first function CNT( ) simply sets the variable count to zero. The second function getitem( ) gets the item code and the item price interactively and assigns them to the array members itemCode[count] and itemPrice[count]. Note that inside this function count is incremented after the assignment operation is over. The function displaySum( ) first evaluates the total value of the order and then prints the value. The fourth function remove( ) deletes a given item from the list. It uses the item code to locate it in the list and sets the price to zero indicating that the item is not 'active' in the list. Lastly, the function displayItems( ) displays all the items in the list.
Classes and Object 89 Notes The program implements all the tasks using a menu-based user interface. 4.9 Static Data Members Having said that each object contains its own separate data, we must now amend that slightly. If a class is defined as static, than only one such item is created for the entire class, no matter how many objects there are. A static data item is useful when all objects of the same class must share a common item of information. A member variable defined as static has similar characteristics to a normal static variable.

| 70% | MATCHING BLOCK 166/304 | W |
|---|---|---|

It is variable only with in the class, but its lifetime is the entire program.

As an example, suppose an object needed to know how many other objects of its class were in the program. In a road-racing game, for example, a race car might want to know how many other cars were still in the race. In this case a static variable count could be included as a member of the class. All the objects would have access to this variable; it would be the same variable for all of them; they would all see the same count. Here's an example, STATDATA, that demonstrates the effect: // statdata.cpp // static class data #include >iostream.h< class clue { private: static int count; // only one data item for all objects public: clue() { count ++; // increments count when } // object created int get count() {return count;} // return count }; void main() { clue f1, f2, f3; // create three objects cout >>"\n count is ">>f1.getcount(); //each object cout >>"\n count is ">>f2.getcount(); //sees the same cout >>"\n count is ">>f3.getcount(); //value of count } The class clue in this example has one data item, count, which is type static int. The constructor for this class causes count to be incremented. In main ( ) we define three objects of class clue. Since the constructor is called three times, count is incremented three times. Another member function, get count ( ), returns the value in
90 Object Oriented Programmimg with C++ Notes count. We call this function from all three objects, and–as we expected–each prints the same value. Here's the output: count is 3 ¬ static data count is 3 count is 3 If we had used an ordinary automatic variable, as opposed to a static variable for count, each constructor would have incremented its own private copy of count once, and the output would have been count is 1 ¬ automatic data count is 1 count is 1 Static class variable are not used often, but they are important in special situations, and knowing about them helps to clarify how the more common automatic variables work. 4.10
Static Member Functions Like static member variable, we can also have static functions. A
member
functions. A member function that is declared static has the following properties: ? A static function can have access to only other static members (functions or variables) declared in the same class. ? A static member function
can be called using
the class name (instead of its objects) as follows: class-name :: function-name;
The example given below illustrates the implementation of these characteristics. The static function show count ( ) displays the number of objects created fill that moment. A count of number of objects created is maintained by the static variable count. The function show code ( ) displays the code number of each object.
Note that the statement code = ++ count; is executed whenever set code ( )
function is invoked and the current value of count is assigned to code. Since each object has its own copy of code, the value contained in code represents a unique number of its object.
Example: #
include >iostream.h< class test { int code; static int count; //static member variable public: void set code(void) { code = ++ count;
Classes and Object 91 Notes }
void showcode(void) { cout >>" object number :" >>code>>"\n"; } static void

show count(
void) // static member // function {
cout >>"count :" >>count>>"\n"; } }; int test :: count; main() { test t1,t2; t1.setcode(); t2.setcode() test ::
show count(); //accessing static function test t3; t3.setcode(); test :: show count(); t1.show code(); t2.show code(); t3.show code(); }
Output of Program: count: 2 count: 3 object number: 1 object number: 2 object number: 3
Remember, the following function definition will not work: static void show count() { cout >> code; //code is not static }
92 Object Oriented Programmimg with C++ Notes 4.11 Arrays
of Objects We know that an array can be of any data type including struct. Similarly, we can also have
arrays of variables that are of the type class. Such variables are called arrays
of objects. Consider the following class definition: class books { char name[30]; float price; public: void
get data(void); void put data(
void); }; The identifier books is a user-defined data type and can be used to create objects that relate to different categories of
the books. Example: books computer[3]; //array of computer books management[14]; //array of management books economics[74];
//array of economics. The array computer contains three objects (computers), namely, computer[0], computer[1] and computer[2], of
type books class. Similarly, the management array contains 14 objects (management) and
the economics
array contains 74 objects (economics). Since an array of objects behave like any other array, we can use the usual array- accessing
methods to access individual elements and then the dot member operator to access the member functions. For example, the
statement
computer[i].put data();
will display the data of the ith element of the array manager. That is, this statement requests the object computer[i] to invoke the
member function put data( ).
An array of objects is stored inside the memory in the same way as a
multi- dimensional array.
The array computer is represented in figure given below. Note the
only the space for data item of the objects is created. Member functions are stored separately and will be used by all the objects. #
include >iostream.h< class books { char name[30]; //string as class member float price; public: void getdata(void); void
getdata(void); }; void books :: getdata(void)
Classes and Object 93 Notes {
cout >>" Enter name:"; cin << name; cout >>" Enter price:"; cin <<
price; } void books :: putdata(void) { cout >>" Name :" >>name>> "\n"; cout >>"
Price :" >>price>>"\n"; } const int
size = 3; main() {
books computer[size]; //array of computers for(int i = 0; i>size; i++) { cout >>"\n Details of computer
book>>i+1>>"\n"; computer[i].get data(); } cout >>"\n"; for (i=0; i>size; i++) { cout >>"\n computer book"
>> i+1 >>"\n"; computer[i].put data(); } }
This being an interactive program, the input data and the program output are as shown below: Interactive input: Details of computer
book1 Enter name: xxx Enter price: 102 Details of computer book2 Enter name:
yyy
94 Object Oriented Programmimg with C++ Notes Enter price: 140 Details of computer book3
Enter name: zzz Enter price: 180 Program output: Computer book1 Name: xxx Price: 102 Computer book2 Name: yyy Price: 140
Computer book3 Name: zzz Price: 180 4.12 Objects As Function
Arguments Like any other data type,
an object may be used as
a
function argument. This can be done in two ways: ?
A copy of the entire object is passed to the function. ? Only the address of the object is transferred to the function.
The first method is called
pass-
by-
value.
Since
a copy of
the
object is passed to the function, any changes made to the object inside the function
do not affect the object
used to call the function.
The
second method is called pass-by-reference. When
an
address of the
object is passed, the called function works directly on

the actual object
used in
the call.
This means that any changes made to
function works directly on
the actual object
used in the
call. This means that
any changes made to
the
object inside the function will reflect in the actual object.
The
pass-by-reference
method is more efficient
since it requires
to pass
only the address of the object and not the entire object.
The
example given below illustrates the use of objects as function arguments. If performs the addition of time in the hour and minutes format.
Since the member function sum( ) is invoked by the object T3, with
in the hour
and minutes format. Since the member function sum( ) is invoked by the object T3, with the objects T1 and T2 as arguments, it can directly access the hours and minutes variables of T3. But, the members of T1 and T2 can be accessed only by using the dot operator (like T1hours and T1minutes). Therefore, inside the function sum( ), the variables hours and minutes refer to T3, T1hours and T1minutes refer to T1, and T2.hours and T2.minutes refer to T2.
Classes and Object 95 Notes Example #include >iostream.h< class time {
int hours;
int minutes; public: void gettime(int h, int m) { hours = h; minutes = m;} void puttime(void) { cout >>hours>> " hours and";
cout >>minutes>>" minutes">>"\n"; } void sum(time, time); //
objects are
arguments }; void time :: sum(timet1, timet2) //t1, t2 are objects {
minutes = t1.minutes + t2.minutes hours = minutes/60; minutes = minutes%60; hours = hours + t1.hours + t2.hours; } main() { time T1, T2, T3; T1.gettime(2, 44); //getT1 T2.gettime(3, 30); //getT2 T3.sum(T1, T2); //T3 = T1+T2 cout >>" T1 = "; T1.puttime();
//display T1 cout >>" T2 = "; T2.puttime(); //display T2 cout >>" T3 = "; T3.puttime(); //
display T3 The output of program would be: T1 = 2 hours and 44 minutes T2 = 3 hours and 30 minutes T3 = 6 hours and 14 minutes
96
Object Oriented Programmimg with C++ Notes
An object can also be passed as an argument to a non-member function. However, such functions can have access to the public member functions only through
the objects passed as arguments to it. These functions cannot have access to the private data members. 4.13 Friendly Functions We have been emphasizing throughout this chapter that the
private members cannot be accessed from outside the class. That
is,

| 100% | MATCHING BLOCK 167/304 | W |
|---|---|---|

a non-member function cannot have an access to the private data of a class.

However, there could be a situation where we would like two classes to share a particular function. For example, consider a case where two classes, manager and scientist, have been defined. We would like to use a function income_tax( ) to operate on the objects of
both
these classes. In such situations, C++ allows the common function to be made friendly with both the classes, thereby allowing the function to have access to the private data of these classes. Such a function need not be a member of any of these classes. To make an outside function “friendly” to a class, we have to simply declare this function as a friend of the class as shown below: class.ABC { ....
.... public: .... .... friend voidxyz(void); //declaration };
The function declaration should be
preceded by the keyword friend.
The function is defined elsewhere in the program like a normal C++ function.
The function definition does not use either the keyword friend or
the scope

operator :: .
The functions that are declared with the keyword friend are known as friend functions.
A function can be
declared as friend in any number of classes.
A friend function, although not a member function, has
full access rights to the private members of
the class.
A
friend function possesses certain special characteristics: ?
It
is not in the scope of the class to which it has been declared as
friend. ?
Since
it is not in the scope of the class, it cannot be
called
using the object of that class.
It
can be invoked
like a normal function without the
help of any object. ?
Unlike member functions,
it cannot access the member names directly and has to use an object name and dot membership operator with each member name. ( e-g. A.x). ?
It can be declared either in public or private part of
a class without affecting its meaning. ? Usually, it has the objects as arguments.
The friend function is often used in operator overloading
which will be discussed later. Example: #include >iostream.h< class sample
Classes and Object 97 Notes { intx; inty; public: void setvalue() { x = 10; y = 40; } friend float mul(samples); //FRIEND declared }; float mul(samples) { return float(s.x * s.y); } main() { sample S1; //objectS1 S1.setvalue(); cout >>" Multiplied value = ">>mul(S1)>>"\
n"; } The output of the example would be: Multiplied value = 400 Note that the friend function accesses the class variables x and y by using the dot operator and the object passed to it. The function call mul(S1) passes the object S1 by value to the friend function.
Member functions of one class can be friend functions of another class. In such cases, they are defined using the scope resolution operator
as shown below: class x { .... .... int fun1(); //member function of x .... }; class y { ....
98
Object Oriented Programmimg with C++ Notes ....
friend int x :: fun1(); //fun1() of x is //friend of y .... }; The function fun1( ) is a member of class x and a friend of class y. We can also declare all the member functions of one class as the friend functions of another class.
In such cases, the class is a
called a friend class. This can be specified
as follows: class z { .... friend class x: //all member functions of x are/friends to z };
The following example demonstrates
how friend functions work as a bridge between the classes. Note that
the function max( ) has arguments from both XYZ and ABC. When the function max( ) is declared as a friend in XYZ for the first time, the compiler will not acknowledge the presence of ABC unless its name is declared in the beginning as class ABC; This is known as 'forward' declaration.
Example:
class ABC; //Forward declaration class XYZ { int x; public: void setvalue(inti) { x = i; } friend void mase (XYZ, ABC); }; class ABC { int a public: void setvalue(int
i)
Classes and Object 99 Notes {
a = i; } friend void mase(XYZ, ABC); }; void max(XYZ m, ABC n) //
Definition of //friend { if(
m.x < n.a) cout >> m.x; else cout >> n.a; } main() { ABC abc; abc.setvalue(10); XYZ xyz; xyz.setvalue(20); max(xyz, abc); }
The output of example would be 20.
As pointed out earlier, a friend function can be called by reference. In this case, local copies of the objects are not made. Instead, a pointer to
the
address of the object is passed and the called function directly works on the actual object used in the call.
This

method can be used to alter the values of the private members of a class. Remember, altering the values of private members is against the basic principles of data hiding. It should be used only when absolutely necessary. The following program shows how to use a common friend function to exchange the private values of two classes. The function is called by reference.
Example: class class_2; class class_1 { int value1; public: void indat a(int a) {
100 Object Oriented Programmimg with C++ Notes
value1 = a; } void display(void) { cout >> value1 >>"\n"; } friend void exchange(class_1 &, class_2&); }; class class_2 { int value2; public: void indata(inta) { value 2 = a; } void display(void) { cout >> value2 >>"\n"; } friend void exchange(class_1 &, class_2&); }; void exchange(class_1 & x, class_2 &y) { int temp = x.value1; x.value1 = y.value2; y.value2 = temp; } main() { class_1 C1; class_2 C2; C1. indata(100); C2. indata(200); cout >>" values before exchange ">>"\n"; C1.display(); C2.display();
Classes and Object 101 Notes
exchange(C1, C2); //SWAPPING cout >>" values after exchange ">>"\n"; C1.display(); C2.display(); } The objects x and y are aliases of C1 and C2 respectively. The statements int temp = x.value1; x.value1 = y.value2; y.value2 = temp; directly modify the values of value1 and value2 declared in class_1 and class_2. Here in the output of example: values before exchange 100 200 values after exchange 200 100 4.14 Returning Objects A function can not only receive objects as arguments but also can return them. The following example illustrates how an object can be created (within a function) and returned to another function. Example: #include >iostream.h< class complex //x + iy form { float x; //real part float y; //imaginary part public: void input(float real, float image) { x = real; y = image; } friend complex sum(complex, complex) void show(complex); }; complex sum(complex C1, complex C2)
102 Object Oriented Programmimg with C++ Notes { complex C3; //object C3 is created C3.x = C1.x + C2.x; C3.y = C1.y + C2.y; return(C3); //returns object C3 } void complex :: show (complex C) { cout >> c.x >>" + " >> c.y >>"\n"; } main() { complex A, B, C; A.input(3.1, 4.64); B.input(2.74, 1.2); C = sum(A, B); //C = A+B cout >>" A = "; A.show(A); cout >>" B = "; B.show(B); cout >>" C = "; C.show(C); } Upon execution, Example would generate the following output: A = 3.1 + i 4.64 B = 2.74 + i 1.2 C = 4.84 + i 6.84 The program adds two complex numbers A and B to produce a third complex number C and displays all the three numbers. 4.15 Const
Member Functions If a member function does not alter any data in the class, then we may declare
it as a const member function as follows: void mul(int, int) const; double get_balance() const; The qualifier const is appended to the function proto types (in both declaration and definition). The compiler will generate an error message if such functions try to alter the data values. 4.15.1
Classes Versus Objects You never pet the definition of a cat; you pet individual cats. You draw a distinction between the idea of a cat and the parcider cat that right now is shedding all over your living room. In the same way, C++ differentiates between the class cat which is the idea of a cat and each individual cat object. Thus, Frisky is an object of type cat in the same
Classes and Object 103 Notes way in which gross weight is a variable of type unsigned int. An object is an individual instance of a class. 4.15.2 Privacy Versus Security Declaring methods or data private enable the compiler to find programming mistakes before they become bugs. Any programmer worth his consulting fees can find a way around privacy if he wants. Stroustrup the inventor of C++ said, “The C++ access control mechanisms provide protection against accident-not against fraud. 4.16 Pointer to member We can have pointers to class member functions and member variables. We can define pointer of class type, which can be used to point to class objects. class Simple { public: int a; }; int main() { Simple obj; Simple* ptr; // Pointer of class type ptr = &obj; cout >> obj.a; cout >> ptr-<a; // Accessing member with pointer } Here you can see that we have declared a pointer of class type which points to class's object. We can access data members and member functions using pointer name with arrow -< symbol. Pointer to Data Members of class ? Declaration: datatype class_name :: *pointer_name ; ? Assignment: pointer_name = &class_name :: datamember_name ; 4.17 Summary When you define a class, you define

| 100% | MATCHING BLOCK 169/304 | SA | 248E1110-Object Oriented Programing using C++( ... (D165248029) |
|---|---|---|---|

a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object. A class definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. 104

Object Oriented Programmimg with C++ Notes 4.18 Check Your Progress
Multiple Choice Questions 1. Class hold _____ a) data b) functions c) both a & b d) none of
the mentioned 2. How many specifiers are present in access specifiers in class? a) 1 b) 2 c) 3 d) 4 3. Which is used to define the member of a class externally? a) : b) :: c) # d) none of the mentioned 4. Which other keywords are also used to declare the class other than class? a) struct b) union c) object d) both
a & b 4.
What is the output
of this program? #

| 50% | MATCHING BLOCK 168/304 | W |
|---|---|---|

include >iostream< using namespace std; class rect { int x, y; public: void val (int, int); int area () { return (x * y); } }; void rect::val (int a, int b) { x = a; y = b; } int main () {

Classes and Object 105 Notes rect rect; rect.val (3, 4); cout >> "rect area: " >> rect.area(); return 0; } a) rect area:12 b) rect area: 12 c) rect area:24

| 100% | MATCHING BLOCK 170/304 | SA | ECAP 444.docx (D142426097) |
|---|---|---|---|

d) none of the mentioned 6. What is the output of this program? #include >iostream< using namespace std;

class CDummy { public: int isitme (CDummy& param); }; int CDummy::isitme (CDummy& param) { if (&param == this) return true; else return false; } int main () { CDummy a; CDummy *b = &a; if (b-<isitme(a)) { cout >> "execute"; } else { cout>>"not execute"; } return 0; } a) execute b) not execute c) none of the mentioned d) both a & b 7. Which of the following is a valid class declaration? a) class A { int x; };
106 Object Oriented Programmimg with C++ Notes b) class B { } c) public class A { } d) object A { int x; }; 8. The fields in the class in c++ program are by default a) protected b) private c) public d) none of the mentioned 9. Constructors are used to a) initialize the objects b) construct the data members c) both a & b d) none of the mentioned 10. When struct is used instead of the keyword class means, what will happen in the program? a) access is public by default b) access is private by default c) access is protected by default d) none of the mentioned 4.19 Questions and Exercises 1. What is a class? 2. What are objects? 3. Briefly explain inline function. 4. How is a member-function of a class defined? 5. What is friend function? 6. How class is specified? 7. What is Const member function 8. What is the difference between member function and private member function? 9. How can be outside function can make inline. 10. Define array within class. 4.20 Key Terms ? Real world: The real world consists of objects like books, tables, chairs and TV. ? Information encapsulation (Hiding): Objects provide the benefit of information hiding. ? Abstraction: It allows us to focus only on those parts of an object that concern us. ? Inheritance: Adding a new functionality to the existing class without creating a new one from the scratch. ? Class: The class keyword is used to declare a class. The braces are used to indicate the start Check Your Progress: Answers: 1. c) both a & b 2. a) 1
Classes and Object 107 Notes 3. d) none of the mentioned 4. d) both a & b 4. b) rect area: 12 6. c) none of the mentioned 7. d) object A { int x; }; 8. a) protected 9. b) construct the data members 10. c) access is protected by default 4.21 Further Readings ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India. Ramesh Vasappanavara, Anand Vasappanavara, Gautam Vasappanavara, Pearson Education India. ?

| 100% | MATCHING BLOCK 172/304 | SA | INF_1016.pdf (D164968061) |
|---|---|---|---|

Balagurusamy (2008) Object Oriented Programming With C++ Tata McGraw-Hill Education. ?

C++ programming: from problem analysis to program design, fifth edition, D.S.malok.
108 Object Oriented Programmimg with C++ Notes Unit 5: Constructors and Destructors Structure 5.1 Introduction 5.2 Constructors 5.2.1 Need for constructor 5.3

| 81% | MATCHING BLOCK 171/304 | W |
|---|---|---|

Parameterized constructors 5.4 Multiple constructors in a class 5.5 Default constructors 5.6 Dynamic initialization of Object 5.7 Copy Constructor 5.8 Dynamic constructor 5.9 Constructing Two Dimensional Arrays 5.9.1

Two Dimensional Arrays 5.10 Const Object 5.11 Destructors 5.12 Summary 5.13 Check Your Progress 5.14 Questions and Exercises 5.15 Key Terms 5.16

| 89% | MATCHING BLOCK 176/304 | SA | DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140) |
|---|---|---|---|

Further Readings Objectives After studying this unit, you should be able to: ? Understand the

constructors. ? Learn about Default, copy, parameterized constructor. ? Understand the concept of destructor. 5.1 Introduction In this lesson, we will discuss about classes and its implementation with special emphasis on the concept of constructor and destructor, static member and This Pointer. 5.2 Constructors When an object of a class is created, its member variables are in uninitialized state, loaded with any arbitrary values. It is desirable to have the member variables initialized to some legal values at the time of object creation.

| 100% | MATCHING BLOCK 173/304 | W |
|---|---|---|

A constructor is a member function of a class, having the same name as its class and which is called automatically each time an object of that class

is

100% MATCHING BLOCK 174/304 W

created. It is used for initializing the member variables with desired initial values.

Constructors and Destructors 109 Notes Example: class student { private: int rollno; float marks; public: . . . student() //constructor of class student { rollno = 0; marks = 0.0; } . . } In this code, the class student has a member function student() that will be called each time an object of this class is created, thereby initializing the member variables with appropriate values. 5.2.1 Need for constructor

92% MATCHING BLOCK 175/304 W

A variable (including structure and array types) in C++ may be initialized with a value at the time of its declaration.

Example: int I = 4; //Integer I has been initialized with 4 float p = 9.1; //Float p has been initialized with 9.1 int a[3] = {2, 7,9}; //Array elements a[0], a[1] and a[2] have been initialized with 2, 7 and 9 respectively struct student { int rollno; float marks; }; int main() { student s1 = {0, 0.0}; .... .... }
110 Object Oriented Programmimg with C++ Notes But no such initialization is possible with objects of a class because the private members are not accessible form outside the class. 5. class student { private: int rollno; float marks; ..... ..... }; int main() { student s1 = {0, 0.0}; //Illegal!! ..... //Private data members not //accessible from non- //member ..... function main() ..... } Declaration A constructor is defined like any other member function. However, ?

43% MATCHING BLOCK 180/304 SA OOP through C++ (Block 2).pdf (D148964031)

It has same name as the name of the class it belongs to ? It does not have any return type (not even void)

Example: class abc { private: int i; public: int j, k; abc() //constructor (same //name as class, //i.e. abc) { //no return type i=0; j=0;k=0; } ..... ..... }; In this example, constructor abc::abc() was defined inline. It may also be defined outline.
Constructors and Destructors 111 Notes Example: class abc { private: int i; public: int j, k; abc(); //constructor (same name as class, i.e. abc) .... .... }; abc::abc() { //outline definition i=0; j=0;k=0; } In the examples given above, the constructors have been defined as a public member so that any function of any object may create this object. However, it can be defined as private or protected as well, but in those cases not all functions can create this object. In later cases the object cannot be created in non-member functions but can be created only in member and friend functions. Example: class abc { private: int i; abc(){ i=0; j=0;k=0; } //constructor defined //private inline public: int j, k; void aaa(void); friend void bbb(void); //friend function : }; void aaa(void) { abc a; //create an instance a of class //abc; valid here aaa being : //member function } void bbb(void) { abc b; //create an instance b of class abc; //valid here bbb being : //friend function } int main() {
112 Object Oriented Programmimg with C++ Notes abc c; //create an instance b of class abc; //invalid here main being : //non-member function abc::abc() not accessible from here } Generally, therefore, constructors are defined as

100% MATCHING BLOCK 177/304 W

public member unless otherwise there is a good reason against. 5.3

Parameterized constructors

100% MATCHING BLOCK 178/304 W

A constructor may also have parameter(s) or argument(s), which can be provided at the time of creating an object of that class.

Example: class abc { private: int i; public: int j, k; abc(int aa, int bb, int cc); //constructor with //parameters aa, bb and cc { i= aa; j=bb; k=cc; } : }; int main(){ abc abc1(5, 8, 10); //create an instance abc1 of class abc; : //initialize i, j, and //with 5, 8, and //10 respectively abc abc2(100, 200, 300);//create an instance //bc2 of class abc; : //initialize i, j, and // k with 100, 200, //and 3000 respectively } Evidently, with parameterized constructor, the correct number and valid argument values must be passed to it at the time of object instantiation. This can be done in two different ways in C++. ? Implicit call: calling the constructor without mentioning its name. ? Explicit call: calling the constructor by mentioning its name explicitly. Example: class abc { private: int i; public:
Constructors and Destructors 113 Notes int j, k; abc(int aa, int bb, int cc); //constructor with //parameters aa, bb and cc { i = aa; j = bb; k = cc; } : : }; int main(){ abc abc1(5, 8, 10); //implicit call: constructors // has not //been called by its name : abc abc2 = abc(100,200,300); //explicit call to the constructor of //abc : } Implicit call to the constructor also allows one to create a temporary instance of a class that and the object thus created does not have any name. It is anonymous. It exists in the memory as long as its member(s) is(are) executing after which it is destroyed. Example: class abc { private: int i; public: int j, k; abc(int aa, int bb, int cc); //constructor with //parameters aa, bb and cc { i = aa; j = bb; k = cc; } void show() { cout>>i>>j>>k; } : : };
114 Object Oriented Programmimg with C++ Notes int main() { abc(5, 8, 10).show(); //create a temporary instance, initialize the members //with 5, 8 and 10 and execute the function show. : //when the execution is over destroy the instance }

100% MATCHING BLOCK 179/304 W

C++ classes are derived data types and so they have constructor(s).

Similarly, the primitive data types also have constructors. If the programmer does not supply the argument, default constructor is called else the value supplied by the programmer is used to initialize the variables of that data type. int aa, bb, cc; //default constructor used int aa(5), k(89); //aa is initialized with 5 and k with 89 float xx(4.3) //xx is initialized with 4.3 5.4 Multiple constructors in a class Constructors Overloading are used to increase the flexibility of a class by having more number of constructor for a single class. Initializing objects by more than one constructor can be done using overloading constructors. Example: #include >iostream.h< class Overclass { public: int x; int y; Overclass() { x = y = 0; } Overclass(int a) { x = y = a; } Overclass(int a, int b) { x = a; y = b; } }; int main() { Overclass A; Overclass A1(4); Overclass A2(8, 12); cout >> "Overclass A's x,y value:: " >> A.x >> " , ">> A.y >> "\n";
Constructors and Destructors 115 Notes cout >> "Overclass A1's x,y value:: ">> A1.x >> " ,">> A1.y >> "\n"; cout >> "Overclass A2's x,y value:; ">> A2.x >> " , ">> A2.y >> "\n"; return 0; } Result: Overclass A's x,y value:: 0 , 0 Overclass A1's x,y value:: 4 ,4 Overclass A2's x,y value:; 8 , 12 In the above example the constructor "Overclass" is overloaded thrice with different initialized values. 5.5 Default constructors

97% MATCHING BLOCK 181/304 W

A constructor may take argument(s). A constructor taking no argument(s) is known as default constructor.

If the programmer does not provide any constructor to a class, the compiler automatically provides one with no argument(s). Default constructor provided by the compiler does nothing more than initializing the data members with dummy values. However, if a constructor is defined for the class the default constructor is no more available, it goes into hiding. Example: class abc { private: int i; public: int j, k; //no explicit constructor defined : }; int main() { abc c; //create an instance c of class //abc; valid the constructor : //abc() is provided by the //compiler } 5.6 Dynamic initialization of Object The dynamic initialization of object means that the initial values may be provided during run time. Even class objects can be initialized dynamically, by providing values at run time. The following example explains it.

52% MATCHING BLOCK 183/304 SA INF_1016.pdf (D164968061)

Example: A Program to find the factorial of an integer by using constructor. #include>iostream.h< #include>conio.h< 116

Object Oriented Programmimg with C++ Notes Class factorial { Private: Int n; Public: Factorial (int number) { N=number; } Void display () { Int fact=1; If (n==0) Cout>>"\n factorial=1"; Else For (int i=1; i>=n; i++) { Fact=fact *I; } Cout>>"\n factorial=">>fact; } }; Void main () { Int x; Clrscr (); Cout>>"\n enter the number to find its factorial "; Cin<<x; Obj.dispay (); getch (); } 5.7 Copy Constructor A copy constructor is a constructor of the form classname (classname & ). The compiler will use the copy constructor whenever you initialize an instance using values of another instance of same type. Example: student st1; // default constructor used
Constructors and Destructors 117 Notes student st2 = st1; // copy

92% MATCHING BLOCK 182/304 W

constructor used Copy constructor is called whenever an instance of same type is assigned to another instance of the same class.

The copy constructor copies the data contents of the instance being assigned to the other instance member by member. If a copy constructor is not defined explicitly, the compiler automatically, creates it and it is public. However, the programmer may define his/her own copy constructor for the class in which case the default copy constructor becomes unavailable.

100% MATCHING BLOCK 184/304 SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

A copy constructor takes a reference to an object of the same class as

argument. Example: class student { int rollno; float marks; public : student (int a, float b) // constructor { rollno = a; marks = b } student(student &s ) // copy constructor { rollno = s.rollno; marks = s.marks + 5; } : : }; These constructors may be used as follows: student s1(5, 78.5); //constructor used to initialize s1.rollno to //5 and s1.marks to 78.5 student s2(s1); //copy constructor used to initialize //s2.rollno to 5 and s2.marks to //s1.marks+5, i.e. 78.5+5=83.5 student s3 = s1; //copy constructor used to initialize //s3.rollno to 5 and s3.marks to //s1.marks+5, i.e. 78.5+5=83.5 5.8

100% MATCHING BLOCK 185/304 SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

Dynamic constructor Dynamic constructor is used to allocate the memory to the objects at the run time. Memory is allocated at run time with the help of 'new' operator. By using this constructor, we can dynamically initialize the objects. Example: # include >iostream.h< # include >conio.h< #

include >string.h< class
str {
118 Object Oriented Programmimg with C++ Notes
char *name;
int len; public: str() { len=0; name=newchar[len+1]; } str(char *s) { len=strlen(s); name=newchar[len+1]; strcpy(name,s); } void show() { cout>>"NAME
IS:-<">>
name>>endl; } void join(str &a,str &b); }; void str::
join(str &a,str &b) { len=a.len+b.len; delete new; name=newchar[len+1]; strcpy(name,a.name); strcat(name,b.name); }; main() { clrscr(); char *first="
HARSHIL"; str n1(first),n2("NINAD"),n3("PRATIK"),n4,n5;
Constructors and Destructors 119 Notes n4.join(n1,n2); n5.join(n4,n3); n1.show(); n2.show(); n3.show(); n4.show(); n5.show(); } 5.9 Constructing Two Dimensional

71% MATCHING BLOCK 189/304 SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

Arrays The least complex type of the multidimensional array is the two-dimensional array. A two-dimensional array is, basically, a list of one-dimensional array. To pronounce a two- dimensional integer array of size

x,y, you would compose something as : type arrayName [ x ][ y ]; Where type can be any substantial C++ data type and arrayName will be a legitimate C++ identifier. A two-dimensional array can be think as a table, which will have x number of rows and y number of columns. A 2-dimensional array a, which contains three rows and four columns can be appeared as below: 5.9.1 Two Dimensional Arrays Accordingly, every element in array is recognized by an element name of the structure a[ i ][ j ], where a is the name of the array, and i and j are the subscripts that extraordinarily distinguish every element in a. Introducing Two-Dimensional Arrays Multidimensional array might be initialized by determining sectioned qualities for every row. Taking after an array with 3 rows and every row have 4 columns. int a[3][4] = { {0, 1, 2, 3} , {4, 5, 6, 7} , {8, 9, 10, 11} }; The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to previous example: int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11}; 5.10 Const Object When an object is declared or created with const, its data members can never be changed, during object's lifetime. Syntax : const class_name object; 120 Object Oriented Programmimg with C++ Notes 5.11 Destructors Just as objects are created, they are destroyed.

95% MATCHING BLOCK 186/304 W

The function that is automatically called when an object is no more required is known as destructor. It is also a member function very much like constructors but with an opposite intent.

Its name is same as that of the class but is preceded by tilde (~). The destructor for class student will be ~student(). A destructor takes no argument and returns no values. Example: class student { int rollno; float marks; public: student() { cout>>"Constructing Student\n"; rollno = 0; marks = 0.0; } ~student() { cout>>"\nDestroying Student"; } }; int main() { student s1; } The output will be: Constructing Student Destroying Student During construction of an object by the constructor, resources may be allocated for use. For example, a constructor may have opened a file and a memory may be allocated to it. Similarly, a constructor may have allocated memory to some other objects. These allocated resources must be deallocated before the object is destroyed. Whose responsibility is this? A destructor performs this cleaning-up task. Therefore, destructor is as useful as constructor. Syntax string () The function creates a default string object of zero size. Example #include >iostream.h< #include >string.h< int main() { string obj;
Constructors and Destructors 121 Notes cout >> "obj [0] contains :" >> obj[0] >>end1; {if (obj[0] == '\0') { cout >> "empty" >> end1; else { cout >> "NOT empty" >> end1; return 0; } The output of the program is: obj[0] contains: empty 5.12 Summary When an object of a class is created, its member variables are in uninitialized state, loaded with any arbitrary values. It is desirable to have the member variables initialized to some legal values at the time of object creation.

100% MATCHING BLOCK 187/304 W

A constructor may also have parameter(s) or argument(s), which can be provided at the time of creating an object of that class.

Constructors Overloading are used to increase the flexibility of a class by having more number of constructor for a single class. Initializing objects by more than one constructor can be done using overloading constructors.

97% MATCHING BLOCK 188/304 W

A constructor may take argument(s). A constructor taking no argument(s) is known as default constructor.

If the programmer does not provide any constructor to a class, the compiler automatically provides one with no argument(s). The dynamic initialization of object means that the initial values may be provided during run time. Even class objects can be initialized dynamically, by providing values at run time. A copy constructor is a constructor of the form classname (classname & ). The compiler will use the copy constructor whenever you initialize an instance using values of another instance of same type.

100% MATCHING BLOCK 190/304 SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

Dynamic constructor is used to allocate the memory to the objects at the run time. Memory is allocated at run time with the help of 'new' operator. 5.13

Check Your Progress Multiple Choice Questions 1. Which value we cannot assign to reference? a) integer b) floating c) unsigned d) null 2. Identify the incorrect statement a) Reference is the alternate name of the object b) A reference value once defined can be reassigned c) A reference value once defined cannot be reassigned d) None of the mentioned 3. Which reference modifier is used to define reference variable? a) &
122 Object Oriented Programmimg with C++ Notes
b) $ c) # d) None of the mentioned 4. What is the output of
this program? #
include >iostream< using namespace std; void swap(
int &a, int &b); int

75% MATCHING BLOCK 191/304 SA 010E2340-Programming in C and C++.pdf (D165445451)

main() { int a = 5, b = 10; swap(a, b); cout >> "In main " >> a >> b; return 0; } void swap(int &a, int &b) { int temp; temp = a; a = b; b = temp;

cout >> "In swap " >> a >> b; } a) In swap 105 In main 105 b) In swap 105 In main 510 c) In swap 510 In main 105 d) None of the mentioned 5. What does a reference provide? a) Alternate name for the class b) Alternate name for the variable c) Alternate name for the pointer

100% MATCHING BLOCK 192/304 SA ECAP 444.docx (D142426097)

d) None of the mentioned 6. What is the output of this program? #include >iostream< using namespace std;

int main() { int
a = 9;
int & aref = a; a++; cout >> "The value of a
is " >>
aref; return 0; } a) 9
Constructors and Destructors 123 Notes b) 10 c) error

63% MATCHING BLOCK 194/304 SA ECAP 444.docx (D142426097)

d) 11 7. What is the output of this program? #include >iostream< using namespace std; void print (char * a) { cout >> a >> endl; } int main () {

const char * a = "Hello world"; print(const_cast>char *< (a) ); return 0; } a) Hello world b) Hello c) World d) Compile time error 8. Identify the correct sentence regarding inequality between reference and pointer. a) we can not create the array of reference. b) we can create the Array of reference. c) we can use reference to reference. d) none of the mentioned 9. Which of the following correctly declares an array? a) int array[10]; b) int array; c) array{10}; d) array array[10]; 10. What is the index number of the last element of an array with 9 elements? a) 9 b) 8 c) 0 d) Programmer-defined 5.14 Questions and Exercises 1. Does default constructor have any parameter? 2. Define destructor. 3. Define constructor 4. What is the difference between constructor and destructor? 5. What do you mean by static function members? 6. Explain class constructor.
124 Object Oriented Programmimg with C++ Notes 7. What is parameterized constructor 8. What is the use of multiple constructors? 9. How objects can be initialized dynamically. 10. Define Const object 5.15 Key Terms ?
Constructor:

100% **MATCHING BLOCK 193/304** W

A constructor is a member function of a class, having the same name as its class and which is called automatically each time an object of that class

is created. ? Implicit call: It is calling the constructor without mentioning its name. ? Explicit call: It is calling the constructor by mentioning its name explicitly. ? Constant function: A member function of a class in which the heading contains the reserved word const at the end ? Default constructor: The constructor without parameters or a constructor that has default values for all parameters that is called when a class object comes into scope in a program. Check Your Progress: Answers: 1. d) null 2. c) A reference value once defined cannot be reassigned 3. a) & 4. a) In swap 105 In main 105 5. b) Alternate name for the variable 6. b) 10 7. a) Hello world 8. a) we can not create the array of reference. 9.
a) 8 10. b) int array[10]; 5.16
Further
Readings ? Balagurusamy (2008)

67% **MATCHING BLOCK 196/304** SA Object Oriented Programming through C++ Block ... (D164970258)

Object Oriented Programming With C++ Tata McGraw-Hill Education. ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India.

Ramesh Vasappanavara, Anand Vasappanavara, Gautam Vasappanavara, Pearson Education India.
Operator Overloading and Type Conversion 125 Notes

82% **MATCHING BLOCK 195/304** W

Unit 6: Operator Overloading and Type Conversion Structure 6.1 Introduction 6.2 Defining Operator Overloading 6.3 Overloading Unary Operators 6.4

Overloading Binary Operators 6.5 Rules for overloading operators 6.6 Typecasting 6.7 Summary 6.8 Check Your Progress 6.9 Questions and Exercises 6.10 Key Terms 6.11

81% **MATCHING BLOCK 197/304** SA ECAP 444.docx (D142426097)

Further Readings Objectives After studying this unit, you should be able to: ? Understand the Operator overloading. ?

Learn about data conversion. 6.1 Introduction
Operator overloading is one of the most exciting features

80% **MATCHING BLOCK 202/304** SA 120E1240_ Object Oriented Programming Using C+ ... (D165245825)

of object oriented programming. It can transform complex, obscure program listings into intuitively obvious ones. 6.2

Defining Operator Overloading
To define an additional task to an operator we must specify what it
means
in
relation to the class to which the operator is applied.
This
is done with the help of a special function, called
operator function,
which describes the task.
The general form of an operator function is: returntype classname :: operator
op (arg-list) { Function body //
task defined } where returntype
is the type of value returned by the specified operation
and op is the operator being overloaded.
The
op is preceded by the keyword operator. operator op is the function name. Operator functions must be either member functions (non-static) or friend functions. A basic difference between them is that a friend function will
have
only

one argument for unary operators and two for binary operators,
while a member function has no arguments for unary operators and only one for binary operators.
This is because the object used to invoke the member function is r_sed implicitly and therefore
126
Object Oriented Programmimg with C++ Notes
is available for the member function. This is not the
case with friend functions. Arguments may be passed either by value or by reference.
Operator functions are declared in the class using prototypes as follows: vector operator + (vector); //vectc_ addition vector operator - ( ); //unary minus friend vector operator + (vector,
vector); //vector addition friend vector operator- (vector); //unary minus vector operator - (vector & a); //subtraction int operator == (vector); //comparison friend int operator == (vector, vector) //comparison
vector is a data type of class and may represent both magnitude and direction (as in physics and engineering) or a series of points called elements (as in mathematics). The
process of
overloading involves the following steps: 1. First, create
a class that defines the data type that is to be used in the
overloading operation. 2. Declare
the
operator function operator op ( ) in the public part of
the class.
It may be either a member function or a friend function. 3. Define the operator function to implement the required operations. 6.3 Overloading Unary
Operators Let's start off by overloading a
unary operator. Unary operators act on only one operand. Examples of unary operators are the increment and decrement operators ++ and --, and the unary minus,
as in -33. Let's consider a program showing how the unary minus operator is overloaded: #
include >iostream.h< class unary { intx; inty;

| 60% | MATCHING BLOCK 198/304 | W |
|---|---|---|

intz; public: void getdata (int a1, int b1, int C1); void display (void); void operator-(); // overload unary minus }; void unary/ :: getdata (intal, intb1, intC1) { x = a1; y = b1; z = c1;

Operator Overloading and Type Conversion 127 Notes void unaryl :: operator -() display (

| 58% | MATCHING BLOCK 199/304 | W |
|---|---|---|

void) { cout >> x >> " "; cout >> y >>" ": cout >> j >>" \n"; } void unary :: operators -() // Defining operator -() { x = -x; y = -y;

j = j; } main () { unaryl u; u.getdata (45, -60, 60); cout >>" u: "; u. display(); -u; // activates operator -() cout >> "u: "; u. display (); } The program produces the following output: u : 45 -60 60 u : -45 60 -60
Note that the function operator -() takes no arguments. Then when does this operator function do? It changes the sign of data members of the object u. Since this function is a member function of the same class, it can directly access the members of the object which activated it. Remember, a statement like u2 = -u1; will not work because, the function operator -() does not return any value. It can work if the function is modified to return an object. It is possible to overload a unary minus operator using a friend function as follows: Friend void operator -(unary & u); // declaration void operator -(unary & u) // definition { u.x = -4.x;
128
Object Oriented Programmimg with C++ Notes u.y = -u.y; u.z = -u.j; }
Note that the argument is passed by reference. It will not work if we pass argument by value
because only a copy of the object that activated the call is passed to
operator - (). Therefore, the changes make inside the operator function will not reflect in the called object. 6.4 Overloading Binary Operators We have just seen how to overload
a
unary operator. The same mechanism can be used to overload a binary operator. A statement like C = A+B; //arithmetic notation overloads the + operator using an operator +() function. The program given below illustrates how this is accomplished. #
include >iostream.h< class complex { float x; // real part flaot y; // imaginary part public: complex // constructor/ { } complex (float real, float imag) // constructor2 { x = real; y = image; } complex operator + (complex); void display (void); }; complex complex :: operator + (complex() { complex temp; // temporary temp, x = x+ c-x; // float addition temp,y = y + c-y; // float addition return (temp); } void
complex ::
display (void)

Operator Overloading and Type Conversion 129 Notes {

47% MATCHING BLOCK 200/304 W

cout >> x >>" + i " >> y >> "\h"; } main () { complex C1, C2, C3; // invokes constructor 1 C1= complex (2.5, 3.5); // invokes constructor 2 C2= complex (1.6, 2.6) // invokes constructor 3 C3= C1 + C2; //

invokes operator +()

100% MATCHING BLOCK 201/304 W

cout >> " C1 = " ; C1. display (); cout >> " C2 = " ; C2. display (); cout >> " C3 = " ; C3. display (); }

The output of program would be: C1 = 2.5 + i 3.5 C2 = 1.6 + i 2.6 C3 = 4.1 + i 6.2 We should note the following features of the function operator +(): 1.
It receives only one complex type argument explicitly. 2. It returns a complex type value. 3. It is a member function of complex
The function is expected to add two complex values and return a complex value as the result but receives only one value as argument. Where does the other value come from? Now let us look at the statement that invokes this function: c3 = C1 + C2; // invokes operator + () function. We know that a member function can be invoked only by an object of the same class. Here, the object C1 takes the responsibility of invoking the function and C2 plays to the function. The above invocation statement is equivalent to C3 = C1. operator +(
C2); //
usual function // call syntax Therefore, in the
operator +() function, the data members of C1 are accessed directly and the data members of C2 (
that is passed as an argument) are accessed using the dot operator. Thus, both the objects are available for the function. 6.5 Rules for overloading operators
There are some restrictions and limitation in overloading the operators.
Some of them are listed below: 1.
Only existing operators
can be overloaded. 2. The overloaded operator must have at least one operand
which must be user defined type.
130 Object Oriented Programmimg with C++ Notes 3. You
cannot change the basic meaning of an operator. 4.
Overloaded operators
must follow the syntax rules of the original operators. 6. “
friend” functions must not be use
to overload certain operators.
However, member function
can
be used to overload them. 6.
Unary
operators, overloaded
by means
of
a member function, take no explicit arguments
and return no explicit values. 7.
Binary
operators
overloaded through a member function take one explicit argument and
those which are overloaded through
a friend function take two
explicit
arguments. 8.
When using binary operators overloaded through a member function, the left hand operand must be an object of the relevant class. 9.
Binary arithmetic operators such as +,-,* and / must explicitly return a value.
They must not attempt to change their own arguments. 6.6

Typecasting Casts are used to convert the type of an object, expression, function arguments, or return value to that of another type. Some conversions are performed automatically by the compiler that is called implicit conversions. The standard C++ conversions and user-defined conversions are performed implicitly by the compiler where needed. Conversions which are explicitly specified by the programmer and are called explicit conversions. The C++ draft standard includes the following four costing operators: ? static–cast ? const–cast ? dynamic–cast, and ? reinterpert–cast Standard conversions are used for integral promotions (e.g., float to double), floating-integral conversions (e.g., int to float), floating point conversions (e.g. float to double), arithmetic conversions (e.g., converting operands to the type of the widest operand before evaluation), pointer conversions (e.g., derived class pointer to base class pointer), reference conversions (e.g., derived class reference to base class reference), and pointer-to-member conversions. You can provide a user-defined conversion from a class X to a class Y by providing a constructor for Y that takes an X as an argument: Y(const X& x) or by providing a class Y with a conversion operator: operator X( ) When a type is needed for an expression that cannot be obtained through an implicit conversion or when more than one standard conversion creates an ambiguous situation, the programmer must explicitly specify that target type of the conversion. C++ Introduces Four New Casting Operators 1. Static–cast, to convert one type to another type. 2. Const–cast, to cost away the "const-ness" or "volatile–ness" of a type. 3. dynamic–cast, for safe navigation of an inheritance hierarchy. 4. reinterpret–cast, to perform type conversions on un-related types. All of the casting operators have the same syntax. For example, to perform a static- cost of Ptr to a Type T we write:
Operator Overloading and Type Conversion 131 Notes ?* t = static–cost >T< (ptr); The Static–cast Operator The Static-cast operator takes the form Static–cost >T< (enpr) to convert the expression expr to type T. Such conversions rely on static (compile–time) type information. Internally, static–costs are used by the compiler to perform implicit type conversions such as the standard conversions are user-defined conversions. The down cost of a base class pointer x to a derived class pointer y can be statically only if the conversion is unambiguous and x is not a virtual base class. Consider this class hierarchy: class Bank Acct. { .... .... } class savings Acct : Public Bank Acct. { .... .... } Given a base class pointer, we can cast it to a derived class pointer: void f(Bank Acct*acct) { Savings Acct*d1 = static–cost>savings Acct*<(acct), } This is a down cost. The static–cost operator allows you to perform safe down casts for non polymorphic classes. One of the more common uses of this type of casting is to perform arithmetic conversions, such as from int to double. For example, to avoid the truncation in the following computation: int total = 500; int days = 9; double rate = total/days; we can write: double rate = static–cost>double<(total)(days; A static–cost may also be used to convert an integral type to an enumeration. Consider: enum fruit{apple = 0, orange, banana};
132 Object Oriented Programmimg with C++ Notes int i/ = 2; fruit f1 = static–cost>fruit<(i/); The Const–cast Operator The const–cast operator takes the form const–cast>T< (enpr) Consider a function, f, which takes a non-const argument: double f(double & d); However, we wish to call f from another function g: voidg(const double & d) { val = f(d); } Since d is const and should not be modified, the compiler will complain because f may potentially modify its value. To get around this dilemma, we can use a const-cast: voidg(const double & d) { val = f(const-cost>double &< (d)); } Another scenario where const-cost is useful is inside const functions. For example, consider class B: class B { public: B() { } ~B() { } voidf() const; private: int_count; }; suppose that, f( ), which is declared to be const, must modify-count whenever it is called: void B::f() const
Operator Overloading and Type Conversion 133 Notes { –count+ = 1; } The compiler will not allow-count to be changed because the function is const. It turns out that the type of the internal this pointer helps the compiler perform this check. Every non-static member function of a class C has this pointer. For non-const member functions of class C, this has type C*const This means that this is a constant pointer. In other words, you cannot change what the pointer this points to, after all, that would be disastrous. We can, however, use const-cost to cost away the "const-ness" of this: void B::f() const { B*const local this = const-cost>B*const<(this); local this –<–count+ = 1; } The Dynamic-cast Operator The dynamic-cost operator takes the form dynamic-cost >T< (expr) and can be used only for pointer or reference types to navigate a class hierarchy. This operator is actually part of C++'s run time type information, or RTTI, sub-system. All of the derived-to-base conversions are performed using the static (compile- time) type information. These conversions may, therefore, be performed on both, non- polymorphic and polymorphic types. These conversions will produce the same result if they are converted using a static-cost. Let's look at the power of run-time type conversion by revisiting the bank account hierarchy introduced above with static-cost. Recall that when acct. does not actually point to a savings Acct. object, the result of the static-cost is undefined. Since Bank Acct. has at least one virtual function, it is a polymorphic class. We can use a dynamic- cost instead to check that the cost was successful: void f(Bank Acct*acct) { Savings Acct*d1 = dynamic-cost>Savings Acct*<(acct); if(d1) { // d1 is a savings account } } Let's expand our bank account hierarchy to include a few more types of accounts, such as a checking account and a money market account. Let's suppose we also want to extend the functionality so that we can credit the interest for all savings and money market accounts in our data base. Suppose further that Bank Acct. is part of a vendor
134 Object Oriented Programmimg with C++ Notes library; we are not able to add new members functions to Bank Acct. since we do not have the source code. Clearly, the best way to incorporate the needed functionality would be to add a virtual function, credit interest( ) to the base class, Bank Acct. But since we are not able to modify Bank Acct, we are unable to do this. Instead, we can employ a dynamic-cost to help us. We add the method credit interest( ) to both Savings Acct. and MM Acct. classes. The resulting class hierarchy looks like: class Bank Acct { .... .... } class Savings Acct : Public Bank Acct { public: // .... void compute interest(); } class MM Acct : Public Bank Acct { public: // .... void compute interest(); } We can now compute interest for an array of Bank Acct*S: void Do Interest(Bank Acct* a[], int num_accts) { for(int i = 0; i>num-accts; i++) // check for savings Savings Acct*aa = dynamic-cost>Savings Acct*<(accts[i]; if(sa) { sa-<credit interest(); } MM Acct*mm = dynamic-cost>MM Acct*<(accts[i]); if(mm) { mm–< credit interest();

Operator Overloading and Type Conversion 135 Notes } } } A dynamic-cast will return a null pointer if the cost is not successful, so only if the pointer is of type Savings Acct* or MM Acct* is interest credited. Dynamic-cast allows you to perform safe type conversions and lets your programs take appropriate actions when such casts fail. When a pointer is converted to a void*, the resulting object points to the most derived object in the class hierarchy. This enables the object to be seen as raw memory. Meyers demonstrates how a cost to void* can be used to determine if a particular object is on the heap. The reinterpret-cast Operator The reinterpret-cost operator takes the form reinterpret-cost >T< (expr) and is used to perform conversions between two unrelated types. The result of the conversion is usually implementation dependant and, therefore, not likely to be portable. You should use this type of cost only when absolutely necessary. A reinterpret-cast can also be used to convert a pointer to an integral type. You

| 100% | MATCHING BLOCK 203/304 | W |
|---|---|---|

typecasting is making a variable of one type, such as an int, act like another type, a

char, for one single application. To type cast something, simply put the type of variable you want the actual variable to act as inside parentheses in front of the actual variable. (char) a will make 'a' function as a char. For Example #include>iostream.h< int main() { cout>>(char)65; // The (char) is a type cast, telling the // Computer to interpret the 65 as a character, // not as a member. It is going to give the // Asc// output of the equivalent of the // number 65(It should be the letter A). return0; } One use for typecasting for is when you want to use the ASC// characters. For example, what if you want to create your own chart of all 256 ASC// characters. To do this, you will need to use to type cast to allow you to print out the integer as its character equivalent. #include>iostream.h< int main() {
136 Object Oriented Programmimg with C++ Notes for(int n=0; x>256; x++) { // The ASC// character set is from 0 to 255 cout>>n>>". ">>(char)x>>" "; // Note the use of the int version of x to // output a number and the use of (char) // to type cost the x into a character which // output the ASC// character that corresponds // to the current number } return0; } These new operators are intended to remove some of the holes in the (type system introduced by the old c-style costs. 6.7 Summary Operator overloading is one of the most exciting features

| 80% | MATCHING BLOCK 204/304 | SA | 120E1240_ Object Oriented Programming Using C+ ... (D165245825) |
|---|---|---|---|

of object oriented programming. It can transform complex, obscure program listings into intuitively obvious ones.

| 100% | MATCHING BLOCK 205/304 | SA | Object Oriented Programming through C++ Block ... (D164970258) |
|---|---|---|---|

To define an additional task to an operator we must specify what it means in relation to the class to which the operator is applied.

Operator
functions must be either member functions (non-static) or friend functions. A basic difference between them is that a friend function will
have
only
one argument for unary operators and two for binary operators,
while a member function has no arguments for unary operators and only one for binary operators.
Typecasts are used to convert the type of an object, expression, function arguments, or return value to that of another type. Some conversions are performed automatically by the compiler that is called implicit conversions. The standard C++ conversions and user-defined conversions are performed implicitly by the compiler where needed. Conversions which are explicitly specified by the programmer and are called explicit conversions. 6.8 Check Your Progress Multiple Choice Questions 1. Pick the other name of operator function. a) function overloading b) operator overloading c) member overloading d) None of the mentioned 2.
Which of the following operators can't be overloaded? a) :: b) + c) – d) [] 3. How to declare operator function?
Operator
Overloading and Type Conversion 137 Notes a) operator operator sign b) operator c) operator sign

| 100% | MATCHING BLOCK 206/304 | SA | ECAP 444.docx (D142426097) |
|---|---|---|---|

d) None of the mentioned 4. What is the output of this program? #include >iostream< using namespace std;

class sample { public: int x, y; sample() {}; sample(int, int); sample operator + (sample); }; sample::sample (int a, int b) { x = a; y = b; }

| 68% | MATCHING BLOCK 207/304 | SA | ECAP 444.docx (D142426097) |
|---|---|---|---|

sample sample::operator+ (sample param) { sample temp; temp.x = x + param.x; temp.y = y + param.y; return (temp); } int main () { sample

a (4,1); sample b (3,2); sample c; c = a + b; cout >> c.x >> "," >> c.y;

92% **MATCHING BLOCK 208/304** SA ECAP 444.docx (D142426097)

return 0; } a) 5, 5 b) 6, 3 c) 3, 6 d) None of the mentioned 5. What is the output of this program? #include >iostream< using namespace std; 138

Object Oriented Programmimg with C++ Notes class Box { double length; double breadth; double height; public: double getVolume(void) { return length * breadth * height; } void setLength( double len ) { length = len; } void setBreadth( double bre ) { breadth = bre; } void setHeight( double hei ) { height = hei; } Box operator+(const Box& b) { Box box; box.length = this-<

32% **MATCHING BLOCK 209/304** SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

length + b.length; box.breadth = this-<breadth + b.breadth; box.height = this-<height + b.height; return box; } }; int main( ) { Box Box1; Box Box2; Box Box3; double volume = 0.0; Box1.setLength(6.0); Box1.setBreadth(6.0); Box1.

setHeight(5.0); Box2.setLength(12.0); Box2.setBreadth(13.0); Box2.setHeight(10.0);
Operator Overloading and Type Conversion 139 Notes volume = Box1.getVolume(); cout >> "Volume of Box1 : " >> volume >>endl; volume = Box2.getVolume(); cout >> "Volume of Box2 : " >> volume >>endl; Box3 = Box1 + Box2; volume = Box3.getVolume(); cout >> "Volume of Box3 : " >> volume >>endl; return 0; } a) Volume of Box1 : 210 Volume of Box2 : 1560 Volume of Box3 : 5400 b) Volume of Box1 : 200 Volume of Box2 : 1560 Volume of Box3 : 5400 c) Volume of Box1 : 210 Volume of Box2 : 1550 Volume of Box3 : 5400

100% **MATCHING BLOCK 210/304** SA ECAP 444.docx (D142426097)

d) None of the mentioned 6. What is the output of this program? #include >iostream< using namespace std;

class Integer { int i; public: Integer(int
ii) : i(ii) {} const Integer operator+(const Integer& rv) const { cout >> "operator+" >> endl; return Integer(i + rv.i); } Integer& operator+=(const Integer& rv) { cout >> "operator+=" >> endl; i += rv.i; return *this; } }; int main() {
140 Object Oriented Programmimg with C++ Notes int i = 1, j = 2, k = 3; k += i + j; Integer ii(1), jj(2), kk(3); kk += ii + jj; } a) operator+ operator+= b) operator+= operator+ c) operator+ operator+

100% **MATCHING BLOCK 212/304** SA ECAP 444.docx (D142426097)

d) None of the mentioned 6. What is the output of this program? #include >iostream< using namespace std;

class myclass { public: int i; myclass *operator-<() {return this;} }; int main() { myclass ob; ob-<i = 10; cout >> ob.
i >> " " >> ob-<i; return 0; } a) 10 10 b) 11 11 c) error d) runtime error 8. Which of the following statements is NOT valid about operator overloading? a)

90% **MATCHING BLOCK 211/304** W

Only existing operators can be overloaded. b) Overloaded operator must have at least one operand

of its class type. c) The overloaded operators follow the syntax rules of the original operator. d) None of the mentioned 9. Operator overloading is a) making c++ operator works with objects b) giving new meaning to existing operator c) making new operator d) both a & b
Operator Overloading and Type Conversion 141 Notes 10.

100% **MATCHING BLOCK 213/304** SA ECAP 444.docx (D142426097)

What is the output of this program? #include >iostream< using namespace std;

ostream & operator>>(ostream & i, int n) { return i; } int main() { cout >> 5 >> endl; cin.get(); return 0; } a) 5 b) 6 c) error d) runtime error 6.9 Questions and Exercises 1. What is operator overloading? 2. What is an operator function? 3. Define unary operator 4. Define binary operator 5. What is the difference between unary and binary overloading? 6.
A friend function cannot be used to overload the assignment operator. Explain why? 7.
Explain the rules of operator overloading. 8. Describe the syntax of an
operator function. 9. Why is it necessary to overload an operator? 10. What is

type conversion 6.10 Key Terms ? Operator function: the function that overloads an operator ? Operator overloading: allows the programmer to extend the definitions of most of the operators so that operators-such as relational operators, arithmetic operators, the insertion operator for data output, and the extraction operator for data input-can be used to manipulate class objects ? Parameterized types: class templates are called parameterized types because, based on the parameter type, a specific class is generate. ? Friend function: a function that is defined outside the scope of a class; it

55% **MATCHING BLOCK 214/304** SA 010E2340-Programming in C and C++.pdf (D165445451)

is a nonmember function of the class but it has access to the private data members of the class ?

Function template: allows you to write a single code segment for a set of related functions Check Your Progress: Answers 1. b) operator overloading
142 Object Oriented Programmimg with C++ Notes 2. a) :: 3. a) operator operator sign 4. b) 6, 3 5. a) Volume of Box1 : 210 Volume of Box2 : 1560 Volume of Box3 : 5400 6. a) operator+ operator+= 7. a) 10 10 8. d) None of the mentioned 9. d) both a & b 10. c) error 6.11
Further
Readings ? Balagurusamy (2008)

67% **MATCHING BLOCK 215/304** SA Object Oriented Programming through C++ Block ... (D164970258)

Object Oriented Programming With C++ Tata McGraw-Hill Education. ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India.

Ramesh Vasappanavara, Anand Vasappanavara, Gautam Vasappanavara, Pearson Education India.
Inheritance, Polymorphism and Pointer 143 Notes Unit 7: Inheritance, Polymorphism and Pointer Structure 7.1 Introduction 7.2 Inheritance 7.3 Derived class 7.3.1 Access Control 7.4 Type

90% **MATCHING BLOCK 216/304** SA ECAP 444.docx (D142426097)

of Inheritance 7.4.1 Single Inheritance 7.4.2 Multi Level Inheritance 7.4.3 Hierarchical Inheritance 7.4.4 Hybrid Inheritance 7.5

Polymorphism 7.6 Pointer 7.6.1 The Pointer Operators 7.7

78% **MATCHING BLOCK 217/304** SA odl C++ lecture notes unit-5.docx (D109013221)

Pointer to object 7.8 This Pointer 7.9 Pointer to derived class 7.10 Virtual Functions 7.11 Pure virtual function

With Pointers 7.12 Virtual Base Class 7.13 Abstract Classes 7.14 Constructor in derived class 7.15 Summary 7.16 Check Your Progress 7.17 Questions and Exercises 7.18 Key Terms 7.19

88% **MATCHING BLOCK 218/304** SA ECAP 444.docx (D142426097)

Further Readings Objectives After studying this unit, you should be able to: ? Understand

Inheritance. ? Discuss the
concept of virtual function. ? Explain the concept of templates. 7.1 Introduction

76% **MATCHING BLOCK 219/304** SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

A pointer is a variable that holds a memory address. This memory address is the location of

other objects (typically another variable) in memory. For example, if one
144 Object Oriented Programmimg with C++ Notes variable contains the address of another variable, the first variable is said to point to the second. In this lesson we will going to learn about inheritance, pointer, template. 7.2 Inheritance Inheritance is a key feature of object-oriented programming in which
a new class (derived class) is created from an existing class(base
class).
The derived class inherits all feature from a base class and it can have additional features of its own. 7.3

Derived class A class can be derived from more than one classes, which implies it can acquire data and capacities from numerous base classes. To characterize a derived class, we utilize a class determination rundown to indicate the base class(es). A class inference list names one or more base classes and has the structure: class derived-class: access-specifier base-class Where access-specifier is one of open, secured, or private, and base-class is the name of a formerly characterized class. In the event that the entrance specifier is not utilized, then it is private naturally. Consider a base class Shape and its derived class Rectangle as take after: # include >iostream< using namespace std; /Base class class Shape { open: void setWidth(int w) { width = w; } void setHeight(int h) { height = h; } secured: int width; int height; }; /
Derived class class Rectangle: open Shape { open:
Inheritance, Polymorphism and Pointer 145 Notes
int getArea() { return (width * height); } }; int main(void) { Rectangle Rect; Rect.setWidth(5); Rect.setHeight(7); /Print the zone of the article. cout >> "Aggregate zone: " >> Rect.getArea() >> endl;
return 0; } At the point when the above code is aggregated and executed, it delivers the accompanying result: Output : 35 7.3.1 Access Control We should recollect that

| 70% | MATCHING BLOCK 220/304 | SA | 137E1240-Object Oriented Programming using C++ ... (D165245896) |
|---|---|---|---|

a determined class can access all the non-private members of its base class. Consequently, base-class members that ought not be accessible to the part functions of inferred classes

ought to be proclaimed private in the base class. We can compress the diverse access sorts as indicated by who can access them in the accompanying way: Table 1.1 Different Types

| 90% | MATCHING BLOCK 221/304 | SA | ODMCA-102_T_Intro_to_Programming_Section_D_25t ... (D43197291) |
|---|---|---|---|

of Access Access Public Protected Private Same class Yes Yes yes Derived classes Yes Yes no Outside classes yes No no

A derived class inherits all base class methods with the following exceptions: ? Constructors, destructors and copy constructors of the base class. ? Overloaded operators of the base class. ? The friend functions of the base class. 7.4 Type of Inheritance While using different type of inheritance, following rules are applied: ? Public Inheritance:
When deriving a class
from a
public
base
class,
public
members of the base class become public members of the derived
class
and
protected members of the base class
become protected members of the derived class.
A base class's private members are
never accessible directly from a derived
146
Object Oriented Programmimg with C++ Notes class, but can be accessed through calls to the
public and protected members of the base class. ?
Protected Inheritance: When deriving from a protected base
class,
public and protected
members of the base class become protected members of the derived class. ?
Private
Inheritance: When deriving from a private base class,
public and protected
members of the base class become private
members of the derived class. 7.4.1
Single Inheritance In single inheritance, there is only one base class and only one derived class. Example: Single Level Inheritance class Base { }; class Derv: public Base { }; 7.4.2 Multi Level Inheritance In this, there will be a chain of inheritance with a class derived from only one parent and will have only one child class.
Inheritance, Polymorphism and Pointer 147 Notes Example: Multi Level Inheritance class A { }; class B: public A { }; class C: public B { }; 7.4.3 Hierarchical Inheritance Hierarchical Inheritance is a method of inheritance where one or more derived classes are derived from common base class.

148 Object Oriented Programmimg with C++ Notes Syntax class base_classname { properties; methods; }; class derived_class1:visibility_mode base_classname { properties; methods; }; class derived_class2:visibility_mode base_classname { properties; methods; }; ... ... ... ... ... ... class derived_classN:visibility_mode base_classname { properties; methods; }; Inheritance, Polymorphism and Pointer 149 Notes 7.4.4 Hybrid Inheritance In this one or more types of inheritance are combined together and used. In this diagram below combination of Hierarchical and Multilevel Inheritance is shown. 7.5 Polymorphism

**93%** **MATCHING BLOCK 222/304** **SA** DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

Polymorphism is an important OOP concept. Polymorphism means the ability to take more than one form. For example, an operation may exhibit different behavior in different instances. The behavior depends upon the types of data used in the operation. For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation will produce a third string by contention. The diagram given below, illustrates that a single function name can be used to handle different number and types of arguments. This is something similar to a particular word having several different meanings depending on the context. Polymorphism plays an important role in following objects having different internal structures to share the same external interface. This means that a general class of operations may be accessed in the same manner even though specific actions associated with each operation may differ. Polymorphism can be implemented using operator and function overloading, where the same operator and function works differently on different arguments producing different results. These polymorphisms are brought into effect at compile time itself, hence is known as early binding, static binding, static linking or compile time polymorphism.

However,
ambiguity creeps in when the base class and the derived class both have

**46%** **MATCHING BLOCK 226/304** **SA** ECAP 444.docx (D142426097)

a function with same name. For instance, let us consider the following code snippet. Class aa { Int x; Public: Void display() {.....} //display in base class }; Class bb : public aa { Int

y;
150
Object Oriented Programmimg with C++ Notes

**97%** **MATCHING BLOCK 223/304** **W**

Public: Void display() {.....} //display in derived class }; Since, both the functions aa.display() and bb.display() are same but at in different classes, there is no overloading, and hence early binding does not apply. The appropriate function is chosen at the run time – run time polymorphism. C++ supports run-time polymorphism by a mechanism called virtual function. It exhibits late binding or dynamic linking. As stated earlier, polymorphism refers to the property by which objects belonging to different classes are able to respond to the same message, but in different forms. Therefore, an essential feature of polymorphism is the ability to refer to objects without any regard to their classes. It implies that a single pointer variable may refer to object of different classes. However, a base pointer, even if is made to contain the address of the derived class, always executes the function in the base class. The compiler ignores the content of the pointer and chooses the member function that matches the type of the pointer. Thus, the polymorphism stated above cannot be implemented by this mechanism.

**100%** **MATCHING BLOCK 224/304** **W**

C++ implements the runtime object polymorphism using a function type known as virtual function. When a function with the same name is used both in the base class and the derived class, the function in the base class is declared virtual by attaching the keyword virtual in the base class preceding its normal declaration. Then C++ determines which function to use at run time based on the type of object pointed to by the base pointer rather than the type of the pointer. Thus, by making the base pointer to point to different objects, one can execute different definitions of the virtual function as given in the program below. #include >iostream.h< class base { public: void display() { cout>>"\n print base"; } virtual void show() //virtual function { cout>>"\n show base"; } }; class derived : public base {

public: void display() {
Inheritance, Polymorphism and Pointer 151 Notes

100% MATCHING BLOCK 225/304 W

cout>>"\n display derived"; } void show() { cout>>"\n show derived"; } }; main() { base bb; derived dd; base *baseptr; cout >>"\nbaseptr points to the base \n"; baseptr = &bb; baseptr -< display(); //calls base function display() baseptr -< show(); //calls base function show() cout >>"\n\nbaseptr points to the derived \n"; baseptr = ⅆ baseptr -< display(); //calls derived function display() baseptr -< show(); //calls derived function show() } The output of this program would be: Baseptr points to base Display base Show base Baseptr points to derived Display derived Show derived Here, we see that the same object pointer points to two different objects of different classes and yet selects the right function to execute. This is implementation of function polymorphism. Remember, however, that runtime polymorphism is achieved only when a virtual function is accessed through a pointer to the base class. It is also interesting to note that since, all the C++ classes are derived from the Object class, a pointer to the Object class can point to any object of any class in C++. 7.6

Pointer If a variable is going to hold a pointer, it must be declared as such. A pointer declaration consists of a base type, an *, and the variable name. The general form for declaring a pointer variable is type * name;
152 Object Oriented Programmimg with C++ Notes where type is the base of the pointer and may be any valid type. The name of the pointer variable is specified by name. The base type of the pointer defines what type of variables the pointer can point to. Technically, any type of pointer can point anywhere in memory. However, all pointer arithmetic is done relative to its base type, so it is important to declare the pointer correctly. 7.6.1 The Pointer Operators We will take a closer look at the Pointer operators here, beginning with a review of their basic operation. There are two special pointer operators: *

50% MATCHING BLOCK 227/304 SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

and &. The & is a unary operator that returns the memory address of its operand. (Remember, a unary operator only requires one operand.) For example, m = &count; place into m the memory address of the variable count. This address is the computer's internal location of the variable. It has nothing to do with the value of count. You can think of & as returning "the address of." Therefore, the preceding assignment statement means "m receives the address of count". To understand the above assignment better, assume that the variable

count uses memory location 2000 to store its value. Also assume that count has a value of 100. Then, after the preceding assignment, m will have

52% MATCHING BLOCK 228/304 SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

the value 2000. The second pointer operator, *, is the complement of &. It is a unary operator that returns the value located at the address that follows. For example, if m contains the memory address of the variable count, q = *m; places the value of count into

q. Thus, q will have the value 100 because 100 is stored at location 2000, which is the memory address that was stored in m. You can think of *ac " at address." In this case, the preceding statement means "q receives the value at address m".

82% MATCHING BLOCK 229/304 SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

Both & and * have a higher precedence than all other arithmetic operators except the unary minus, with which they are equal.

You must make sure that

98% MATCHING BLOCK 230/304 SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

your pointer variables always point to the correct type of data. For example, when you declare a pointer to be of type int, the compiler assumes that

any address that it holds point to an integer variable – whether it actually does or not. Because C allows you to assign any address to a pointer variable, the following code fragment compiles with no error messages (or only warnings, depending upon your compiler), but does not produce the desired result: # include >stdio.h< int main (void) { double x = 100.1, y; int *p; /* The next statement causes p (which is an integer pointer) to point to a double.*/ p = &x; /* The next statement does not operate as expected. */

Inheritance, Polymorphism and Pointer 153 Notes y = *p; printf("%i", y); /* won't output 100.1 */ return 0; } This will not assign the value of x to y. Because p is declared as an integer pointer, only or 4 bytes of information will be transferred to y, not the 7 bytes that normally make up a double. In C++, it is illegal to convert one type of pointer into another without the use of an explicit type cast. For this reason, the preceding program will not even compile if you try to compile it as a C++ (rather than as a C) program. However, the type of error described can still occur in C++ in a more roundabout manner. 7.7 Pointer to object A variable that holds an address value is known as a pointer variable or pointer. Pointer can point to object, simple data type and arrays. Sometime we did not know about the number of objects we need at the time when we start writing the program. In that case, we write new to create objects during program execution. Consider an example of pointer to



student():rollno(0),name("") {} student(int r, string n): rollno(r),name (n) {} void get() { cout>>"enter roll no"; cin<<rollno; cout>>"enter name"; cin<<name; } void print() {
154 Object Oriented Programmimg with C++ Notes cout>>"roll no is ">>rollno; cout>>"name is ">>name; } }; void main () { student *ps=new student; (*ps).get(); (*ps).print(); delete ps; } C++ String
C++ provides following two types of string representations: ? The C-style character string. ? The string class type introduced with Standard C++. The C-Style Character String: The C-style character string originated within the C language and continues to be supported within C++. This string is actually a one-dimensional array of characters which is terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null. The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello." char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'}; If you follow the rule of array initialization, then you can write the above statement as follows: char greeting[] = "Hello"; Following is the memory presentation of above defined string in C/C++:
Actually, you do not place the null character at the end of a string constant. The C++ compiler automatically places the '\0' at the end of the string when it initializes the array.
Let us try to print above-mentioned string:
Inheritance, Polymorphism and Pointer 155 Notes #
include >iostream< using namespace std; int main () { char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'}; cout >> "Greeting message: "; cout >> greeting >>
endl; return 0; }
When the above code is compiled and executed, it produces
result something as follows:
Greeting message: Hello C++ supports a wide range of functions that manipulate null-terminated strings:
S.N.
Function & Purpose 1 strcpy(s1, s2); Copies string s2 into string s1. 2 strcat(s1, s2); Concatenates string s2 onto the end of string s1. 3 strlen(s1); Returns the length of string s1. 4 strcmp(s1, s2); Returns 0 if s1 and s2 are the same; less than 0 if s1>s2; greater than 0 if s1<s2. 5 strchr(s1, ch); Returns a pointer to the first occurrence of character ch in string s1. 6 strstr(s1, s2); Returns a pointer to the first occurrence of string s2 in string s1.
156
Object Oriented Programmimg with C++ Notes Following example makes use of few
of the above-mentioned functions: #
include >iostream< #include >cstring< using namespace std; int main () {
char str1[10] = "Hello"; char str2[10] = "
World"; char str3[10]; int len ; // copy str1 into str3 strcpy( str3, str1); cout >> "strcpy( str3, str1) : " >> str3 >> endl; // concatenates str1 and str2 strcat( str1, str2); cout >> "strcat( str1, str2): " >> str1 >> endl; // total lenghth of str1 after concatenation len = strlen(str1); cout >> "strlen(str1) : " >> len >>
endl; return 0; }
When the above code is compiled and executed, it produces
result something as follows:
strcpy( str3, str1) : Hello strcat( str1, str2): HelloWorld strlen(str1) : 10 The String Class in C++: The standard C++ library provides a string class type that supports all the operations mentioned above, additionally much more functionality.
We will study this class in C++ Standard Library but for now let us check following example:
Inheritance, Polymorphism and Pointer 157 Notes At this point, you may not understand this example because so far we have not discussed Classes and Objects. So can have a look and proceed until you have understanding on Object Oriented Concepts. #
include >iostream< #include >string< using namespace std; int main () { string
str1 = "Hello"; string str2 = "World"; string str3; int len ; // copy str1 into str3 str3 = str1; cout >> "str3 : " >> str3 >>
endl; // concatenates str1 and str2 str3 = str1 + str2; cout >> "str1 + str2 : " >> str3 >> endl; // total lenghth of str3 after concatenation len = str3.size(); cout >> "str3.size() : " >> len >>
endl; return 0; }
When the above code is compiled and executed, it produces

result something as follows:
str3 : Hello str1 + str2 : HelloWorld str3.size() : 10
String Manipulation in C++ The Standard String Class (strings) We will usually refer to these things as strings, and the other type as c-strings or ntcas. One fact that you want to keep in mind is that they are indeed of different type, so you
158 Object Oriented Programmimg with C++ Notes cannot expect the compiler to understand you if you start to mix them. There is a set of functions available to the programmer for the manipulation of standard strings. Some compilers will have them and some will not. In order to access them in the GNU compiler, you will need to include the system file string. Thus, when using more specialized string class functions, put the preprocessor command #include >string< at the top of your program with other system includes. I will leave it up to you to discover the different functions allowing you to manipulate standard strings by looking them up in the text or on-line. They are numerous and varied. But remember, the GNU compiler may not have them available and you will have to work around them. string name = "Clayton"; cout>>name.length(); // 7 is output to the screen There is one particular issue that needs special attention with strings. As you might have experienced and were warned about, reading into a string variable using a cin statement will only read up to the first space. This can be quite inconvenient indeed. Now we will learn how to get around that. (We will deal with the same issue using c- strings, but the fix is a tiny bit different.) We need to employ a special function and a special way to call that function that I cannot fully explain at this juncture. Here's how to use it. The name of the function is getline. The syntax is getline(cin,string_var,delimiter_character); This line of code will read a string of characters from the keyboard (cin) until the delimiting character is reached and that string is stored in string_var. string name; getline(cin,name, '\n'); // user enters Clayton Price cout>>name; // outputs Clayton Price The delimiter character defaults to the newline character, \n. Thus, the above getline statement is equivalent to getline(cin,name); The delimiting character is discarded when read. Very Important You must understand a subtlety about using getlines and cin statements. Remember: cin will always leave a newline character, \n, in the stream, while getline does not. And how does this affect you? Consider this code: string name; int n; cout>>"enter a number: "; cin<<n; cout<<"enter your name: "; getline(cin,name); return 0; Suppose the first prompt is answered with 5 being entered. The variable n takes the value 5. What happens next can be mystifying if you are not aware of what is in the stream. The second prompt is printed to the screen and in an instance, before you have a chance to blink, the program ends. Why?! Remember, cin left the newline character on the stream. After the second prompt was issued, the getline function started to read the stream from the keyboard, read the newline, considered itself done reading from the
Inheritance, Polymorphism and Pointer 159 Notes stream, discarded the newline and ended the program. Thus, you see that the problem was information (a newline character in this case) was left on the stream. In order to make this code function correctly, you must be sure that the stream is empty before using a getline call. This can be done in more than one way, but I think the simplest is to use the ignore function. cin.ignore(int_value, delim_char); This function call will take characters from the stream and discard them until int_value characters are discarded or delim_char is read, whichever comes first. If the later, then the delim_char is also discarded. cin.ignore(500, '\n'); // reads and discards up to 500 chars from the stream Include this line of code before using a getline and probably your problems will be solved. Null-Terminated Character Arrays (C-strings) Functions to deal with ntcas have already been discussed. Anything that hasn't already been stated you can build yourself. But, once again, we have the problem of reading a string of characters into a ntca using a cin statement and having it read only to the first space. And, once again, we will solve the problem using a call to a getline function. Notice I said "a getline" function....not "the getline". There are two versions of the function, one for standard strings and one for ntcas. For ntcas the syntax is cin.getline(ntca, max_num_chars); The dot in the middle of this function call is the same dot used to access members of a struct. However, I cannot relate to you at this time exactly what is going here. But, after a few more lessons, you will completely understand. Also, you will have the very same problem with this getline and the \n character as you did with the other getline. You will handle it exactly the same way. char address[80]; // assumes address is 79 char (or less) + null cout>>"enter address: "; // suppose enter: 245 N. Oak St. apt. 1B cin.ignore(500,'\n'); // be sure to clear the stream cin.getline(address, 79); // keeping at least one space for null char cout>>address; // -< 245 N. Oak St. apt. 1B Character Manipulation Using the ntca address from the example in the last section, I will demonstrate some of the character manipulation functions. There are several built-in functions that can help you manipulate individual characters. This can be important when trying to understand the contents of a ntca or string. You may have to #include>cctype< in order to use these. (I say "may" because you can never know until you ask just what is necessary for a particular compiler. It isn't necessary with our compiler.) You can find a more complete list of these functions in most C++ text books or on-line. Here are some of them: toupper(char) returns the uppercase of arg sent toupper('a'); -< 'A'
160 Object Oriented Programmimg with C++ Notes tolower(char) similar isupper(char) returns bool: true if uppercase isupper('a'); -< false islower(char) similar isalpha(char) similar isdigit(char) similar ispunct(char) returns bool: true if punctuation ispunct('!'); - < true isspace(char) returns bool: true if whitespace – space, newline, tab int i = 0, count = 0; while (ntca[i] != '\0') { if (ispunct(ntca[i])) count++; i++; } cout>>count>>endl; If we run this code on the ntca address from the last section, the output would be 3. Change the function call to isspace(), the output will be 4. int i = 0; while (ntca[i] != '\0') { ntca[i] = toupper(ntca[i]); i++; } cout>>ntca>>endl; And if we run this code on the ntca address from the last section, the output will be: 245 N. OAK ST. APT. 1B So you see that you can used these functions to "tear down" a string and discover what is in it, manipulate it, change it, etc. You should not overlook the ease with which you can write these functions yourself. Let's see. Write a function that will accept a char and return true if it is a digit, false otherwise. bool IsDigit (const char input) { bool digit = true; if (input <= 48 && input >= 57) digit = false; return digit; } Challenge: write this function as one line of code. Answer is at the end of this page. Input/Output Character Manipulation There are functions built in to the compiler that allow individual character input and output. Let's take a look. The functions I want to mention here are get, peek, putback, put, and ignore. They are fairly simple to understand. I will give most attention to the function get(). The syntax is cin.get(char_variable);

Inheritance, Polymorphism and Pointer 161 Notes The parameter is a reference, so what is sent will be changed by the function. get() will pull the next character off the input stream and instantiate the char_var sent as an argument with its value. Thus, you have the ability to input information from the input stream character-by-character. char next; cout>>"enter your poem: "; do { cin.get(next); cout>>next; } while (next != '\n'); This code will simply echo to the screen what has been input at the keyboard. It is equivalent to char poetry[500]; cout>>"enter your poem: "; cin.getline(poetry,499); cout>>poetry; except that the second is limited to 500 character poems and the first isn't limited at all. It is also different because the first is reading character-by-character while the second is reading an entire line. At this juncture I want to emphasize to you that you have now learned three ways to read in information from the keyboard: line-by-line using getline() word-by-word using cin<< character-by-character using get() Each of these methods has an advantage, but it depends on the requirements of the problem. If you need to process each character, then read char-by-char. If you need to process each word, then use cin<<. The put() function is actually fairly useless. cout.put(char_var); is equivalent to cout>>char_var;. End of discussion! The putback() function will allow you to put a character back into the input stream: cin.putback(char_var); The peek() function will allow you to know what the next character in the input stream is without extracting from that stream. char_var = cin.peek(); All these functions will allow a certain degree of manipulation of the input stream and its contents for single characters. 7.8 This Pointer When a member function is called, it is automatically passed an implicit argument that is a pointer to the invoking object (that is, the object on which the function is called). This pointer is called this. To understand this, first consider a program that creates a class called pwr that computes the result of a number raised to some power:

162 Object Oriented Programmimg with C++ Notes #include >iostream< using namespace std; class pwr { double b; int e; double va1; public: pwr (double base, int exp); double get_pwr() { return val; } }; pwr::pwr (double base, int exp) { b = base; e = exp; val= l; if (exp==0) return; for (;exp<0; exp--) val = val *b; } int main() { pwr x (4. 0, 2), y(2.5, 1), z (5.7,0); cout >> x.get _pwr() >> " "; cout >> y.get _pwr() >> " "; cout >> z.get _pwr() >> " \n"; return 0; } Within a member function, the members of a class can be accessed directly, without any object or class qualification. Thus, inside pwr(), the statement b= base; means that the copy of b associated with the invoking object will be assigned the value contained in base. However, the same statement can also be written like this: this-<b = base; The this pointer points to the object that invoked pwr(). Thus, this-<b refers to that object's copy of b. For example, if pwr() had been invoked by x (as in x (4.0,2)), then this in the preceding statement would have been pointing to x, Writing the statement without using this is really just shorthand. Here is the entire pwr() function written using the this pointer:

Inheritance, Polymorphism and Pointer 163 Notes pwr::pwr (double base, int exp) { this-<b = base; this-<e = exp; this-<val = 1; if (exp==0) return; for ( ; exp<0; exp--) this-<val = this-<val * this-<b; } Actually, no C++ programmer would write pwr() as just shown because nothing is gained, and the standard form is easier. However, the this pointer is very important when operators are overloaded and whenever a member function must utilize

| 50% | MATCHING BLOCK 232/304 | SA | C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784) |
|---|---|---|---|

a pointer to the object that invoked it. Remember that the this pointer is automatically passed to all member functions. Therefore,

get_pwr() could also be rewritten as shown here: double get_pwr() {return this_<val; } In this case, if get_pwr() is invoked like this: y.get_pwr(); Then this will point to object two final points about this. First, friend functions are not members of a class and, therefore, are not passed this pointer. Second, static member functions do not have a this pointer. 7.9 Pointer to derived class #include >iostream< utilizing namespace std;

| 88% | MATCHING BLOCK 235/304 | SA | 137E1240-Object Oriented Programming using C++ ... (D165245896) |
|---|---|---|---|

class BaseClass { int x; public: void setx(int i) { x = i; } int getx() { return x; } };

class DerivedClass : public BaseClass { int y; public:

164 Object Oriented Programmimg with C++ Notes void sety(int i) { y = i; } int gety() { return y; } }; int main() { BaseClass *p; /pointer to BaseClass type BaseClass baseObject;/ /object of BaseClass DerivedClass derivedObject;/ object of DerivedClass p = &baseObject; /use p to get to BaseClass object p-<setx(10); /access BaseClass object cout >> "Base item x: " >> p-<getx() >> '\n'; p = &derivedObject; /point to DerivedClass object p-<setx(99); /access DerivedClass object derivedObject.sety(88); /can't utilize p to set y, so do it specifically cout >> "Determined item x: " >> p-<getx() >> '\n'; cout >> "Determined item y: " >> derivedObject.gety() >> '\n'; return 0; } 7.10 Virtual Functions Virtual means existing in effect but not in reality. A virtual function then is one that does not really exist but nevertheless appears real to some parts of a program. Consider an example, which explains the need of virtual function: Suppose you have a number of objects of different classes but you want to put them all on a list and perform a particular operation on them by using the same function call. For example, suppose a graphics program include several different shapes: a triangle, a ball, a square, and so on. Each of these class has a member function draw() that causes the object to be drawn on the screen. Now suppose you plan to make a picture by grouping a number of these elements together, and you want to draw the picture in a convenient way. One approach is to create an array that holds pointers to all the different objects in the picture. The array might be defined like this. shape* ptrass [100]; // array of 100 pointers to shapes

Inheritance, Polymorphism and Pointer 165 Notes If you insert pointers to all the shapes into this array, you can then draw an entire picture using a simple loop: for(int j=0; j>N; j++) ptrass[j] -< draw(); In virtual function, completely different functions are executed by the same function call. If the pointer in ptrass points to a ball, the function that draws a ball is called; if it points to a triangle, the triangle-drawing function is drawn. This is an important example of polymorphism, or giving different meanings to the same thing. However, for this polymorphic approach to work, several conditions must be met. First, all the different classes of shapes, such as balls and triangles, must be derived from a single base class. Second, the draw ( ) function must be declared to be virtual in the base class. Consider the program given below to clearly understand
the
virtual function: #

| 45% | MATCHING BLOCK 233/304 | W |
|---|---|---|

include >iostream.h< class Base { public: void display() { cout >>"\n Display base "; } virtual void show() { cout >>"\n show base"; } }; Class derived : Public Base { public: void display() { cout >>"\n Display derived"; } void show() { cout >>"\n show derived"; } }; main() 166 Object Oriented Programmimg with C++ Notes { Base B; Derived D; // Declarations Base *bptr; cout >>"\n bptr points to Base\n"; bptr = &B; bptr -<display(); // calls Base version bptr -<show(); // calls Base version cout >>"\n\n bptr points to Derived \n"; bptr = &D; bptr -<display(); // calls

Base version bptr -<
show(); // calls Derived version }
The output of the
above
Program would be: bptr points to Base Display base show base bptr points to Derived Display base Show derived
Note that when bptr is made to point to the object D, the statement bptr -<display(); calls only the function associated with the Base (i.e. Base :: display( ) ) whereas the statement bptr -<show(); Calls the Derived version of show ( ). This is because the function display ( ) has not been made virtual in the Base class.
One important point to remember is that, we must access virtual functions
through the use of a pointer declared as a pointer to the base class.
Why can't we use the object name (with the dot operator) the same way as any other member function to call the virtual function? We can, but remember,
runtime
polymorphism is
achieved only when a virtual function is accessed through a pointer to
the base class. 7.11
Pure virtual function
With Pointers Let's make a single change in our program. We'll place the keyword virtual in front of the declarator for the show ( ) function in the base class. Here's the listing for the resulting program, VIRT:
Inheritance, Polymorphism and Pointer 167 Notes // virt, cpp // virtual functions accessed from pointer #

| 60% | MATCHING BLOCK 234/304 | W |
|---|---|---|

include >iostream.h< class base // base class { public: virtual void show() // virtual function { cout >>"\n Base "; } }; class Derv1 : Public Base //

derived class1 {

| 52% | MATCHING BLOCK 239/304 | SA | 010E2340-Programming in C and C++.pdf (D165445451) |
|---|---|---|---|

public: void show() { cout >>"\n Derv1"; } }; class Derv2 : Public Base // derived class2 { public: void show() { cout >>"\n Derv2";} }; void main() { Derv1 dv1; //

object of derived class1 Derv2 dv2; // object of derived class2 Base* ptr; // pointer to base class ptr = &dv1; // put address of dv1 in pointer ptr -<show(); // execute show() ptr = &dv2; // put address of dv2 in pointer ptr -<show(); // execute show() }
168 Object Oriented Programmimg with C++ Notes The output of this program is Derv1 Derv2 Now, as you can see, the member functions of the derived classes, not the base class, are executed. We change the contents of ptr from the address of Derv1 to that of Derv2, and the particular instance of show ( ) that is executed also changes. So the same function call, ptr -<show ( ); executes different functions, depending on the contents of ptr. The rule is that the compiler selects the function based on the contents of the pointer ptr, not on the type of the pointer, as in NOT VIRTUAL. 7.12

82% MATCHING BLOCK 236/304 W

Virtual Base Class Assume you have two derived classes B and C that have a common base class A, and you additionally have another class D

which is inherited

88% MATCHING BLOCK 237/304 W

from B and C. You can declare the base class A as virtual to guarantee that B and C share the same subobject of A. In the accompanying illustration, an object of class D has two particular subobjects of class L, one through class B1 and another through class B2. You can use the keyword virtual before the base class specifiers in the base lists of classes B1 and B2 to show that one and only subobject of type L, shared by class B1 and class B2, exists. For example: virt1 class L { /* ... */ }; // indirect base class class B1 : virtual public L { /* ... */ }; class B2 : virtual public L { /* ... */ }; class D : public B1, public B2 { /* ... */ }; // valid Using the keyword virtual in this example ensures that an object of class D inherits only one subobject of class L. 7.13

Abstract Classes Abstract Class is a class which contains atleast one Pure Virtual function in it, and is use to give an Interface to its sub classes. Classes acquiring an Abstract Class must provide definition to the pure virtual function, else they will also become abstract class.
Inheritance, Polymorphism and Pointer 169 Notes Characteristics of Abstract Class ? Abstract class can't be instantiated. ? Abstract class can have normal function and variables along with a pure virtual function. An interface describes the behavior or capabilities of a C++ class without committing to a particular implementation of that class. The C++ interfaces are implemented using abstract classes and these abstract classes should not be confused with data abstraction which is a concept of keeping implementation details separate from associated data.
A class is made abstract by declaring at least one of its functions as pure virtual function.
A pure virtual function is specified by placing "= 0" in its declaration as follows: class Box { public: // pure virtual function virtual double getVolume() = 0;

100% MATCHING BLOCK 238/304 W

private: double length; // Length of a box double breadth; // Breadth of a box double height; // Height of a box };

The purpose of an abstract class (often referred to as an ABC) is to provide an appropriate base class from which other classes can inherit. Abstract classes cannot be used to instantiate objects and serves only as an interface. Attempting to instantiate an object of an abstract class causes a compilation error. Thus, if a subclass of an ABC needs to be instantiated, it has to implement each of the virtual functions, which means that it supports the interface declared by the ABC. Failure to override a pure virtual function in a derived class, then attempting to instantiate objects of that class, is a compilation error. Classes that can be used to instantiate objects are called concrete classes. Abstract Class Example: Consider the following example where parent class provides an interface to the base class to implement a function called getArea(): #include >iostream< using namespace std; // Base class class Shape { public:
170 Object Oriented Programmimg with C++ Notes // pure virtual function providing interface framework. virtual int getArea() = 0; void setWidth(int w) { width = w; } void setHeight(

51% MATCHING BLOCK 240/304 SA OOP through C++ (Block 2).pdf (D148964031)

int h) { height = h; } protected: int width; int height; }; // Derived classes class Rectangle: public Shape { public: int getArea() { return (width * height); } }; class Triangle: public Shape { public: int getArea() { return (width * height)/2; } }; int main(

void) { Rectangle Rect; Triangle
Tri;
Inheritance, Polymorphism and Pointer 171 Notes
Rect.setWidth(5); Rect.setHeight(7); // Print the area of the object. cout >> "Total
Rectangle area: " >> Rect.getArea() >> endl; Tri.setWidth(5); Tri.setHeight(7); // Print the area of the object. cout >> "Total Triangle area: " >> Tri.getArea() >>

100% MATCHING BLOCK 242/304 SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

endl; return 0; } When the above code is compiled and executed, it produces the following result: Total

Rectangle area: 35 Total Triangle area: 17 You can see how an abstract class defined an interface in terms of getArea() and two other classes implemented same function but with different algorithm to calculate the area specific to the shape. Designing Strategy: An object-oriented system might use an abstract base class to provide a common and standardized interface appropriate for all the external applications. Then, through inheritance from that abstract base class, derived classes are formed that all operate similarly. The capabilities (i.e., the public functions) offered by the external applications are provided as pure virtual functions in the abstract base class. The implementations of these pure virtual functions are provided in the derived classes that correspond to the specific types of the application. This architecture also allows new applications to be added to a system easily, even after the system has been defined. 7.14 Constructor in derived class A constructor assumes a vital part in initializing an object. An essential note, while using constructors during inheritance, is that, as long as a

| 76% | MATCHING BLOCK 241/304 | W |
|---|---|---|

base class constructor does not take any arguments, the derived class need not have a constructor function. However, if

a
base class contains a constructor with one or more arguments,
then it is necessary for the derived class to
have a constructor and pass the arguments to the base class constructor.
Keep in mind, while applying inheritance, we usually create objects using
derived class. Thus
derived class pass arguments to the base class constructor. , the base constructor is executed first and then the constructor in the derived class is executed.
Illustration: Program to show how constructor are invoked in derived class #include >iostream.h<
172 Object Oriented Programmimg with C++ Notes class alpha ( private: int x; public: alpha(int i) { x = i; cout >> "\n alpha initialized \n"; } void show_x() { cout >> "\n x = ">>x; } ); class beta ( private: float y; public: beta(float j) { y = j; cout >> "\n beta initialized \n"; } void show_y() { cout >> "\n y = ">>y; } ); class gamma : public beta, public alpha ( private: int n,m; public: gamma(int a, float b, int c, int d):: alpha(a), beta(b)
Inheritance, Polymorphism and Pointer 173 Notes { m = c; n = d; cout >> "\n gamma initialized \n"; } void show_mn() { cout >> "\n m = ">>m; cout >> "\n n = ">>n; } ); void main() { gamma g(5, 7.65, 30, 100); cout >> "\n"; g.show_x(); g.show_y(); g.show_mn(); } Output: beta initialized alpha initialized gamma initialized x = 5 y = 7.65 m = 30 n = 100 7.15 Summary In this chapter we studied about pointer, inheritance, types of inheritance, use of constructor in derived class and about template. Pointers provide an essential tool for increasing the power of C++. Whereas inheritance allow us to get the features of base class in derived class by various means.
174 Object Oriented Programmimg with C++ Notes 7.16 Check Your Progress Multiple Choice Questions 1. To what does the function pointer point to? a) variable b) constants c) function d) absolute variables 2. What we will not do with function pointers? a) allocation of memory b) de-allocation of memory c) both a & b d) none of the mentioned 3. What is the default calling convention for a compiler

| 64% | MATCHING BLOCK 243/304 | SA | ECAP 444.docx (D142426097) |
|---|---|---|---|

in c++? a) __cdecl b) __stdcall c) __pascal d) __fastcall 4. What is the output of this program? #include >iostream< using namespace std; int add(int first, int

second) { return first + second + 15; } int operation(int first, int second, int (*functocall)(int, int)) { return (*functocall)(first, second); } int main() { int a; int (*plus)(int, int) = add; a = operation(15, 10, plus); cout >> a; return 0; }

| 100% | MATCHING BLOCK 244/304 | SA | ECAP 444.docx (D142426097) |
|---|---|---|---|

a) 25 b) 35 c) 40 d) 45 5. What is the output of this program? #include >iostream<

Inheritance, Polymorphism and Pointer 175 Notes using namespace std; void func(int x) { cout >> x ; } int main() { void (*n)(int); n = &func; (*n)( 2 ); n( 2 ); return 0; }

| 100% | MATCHING BLOCK 245/304 | SA | ECAP 444.docx (D142426097) |
|---|---|---|---|

a) 2 b) 20 c) 21 d) 22 6. What is the output of this program? #include >iostream< using namespace std;

int
n(char, int); int (*p) (char, int) = n; int main() { (*p)('d', 9); p(10, 9); return 0; } int n(char c, int i) { cout >> c >> i; return 0; } a) d99 b) d9d9 c) d9 d) compile time error 7.

90% **MATCHING BLOCK 246/304** SA ECAP 444.docx (D142426097)

What is the output of this program? #include >iostream< using namespace std; int func (int

a, int b) { cout >> a;
176
Object Oriented Programmimg
with C++ Notes cout >> b; return 0; } int main(void) { int(*ptr)(char, int); ptr = func; func(2, 3); ptr(2, 3); return 0; } a) 2323 b) 232 c) 23 d) compile time error 8. What are the mandatory parts to present in function pointers? a) & b) return values c) data types d) none of the mentioned 9. Which of the following can be passed in function pointers? a) variables b) data types c) functions d) none of the mentioned 10. What is meaning of following declaration? int(*ptr[5])(); a) ptr is pointer to function. b) ptr is array of pointer to function. c) ptr is pointer to such function which return type is array. d) ptr is pointer to array of function. 7.17 Questions and Exercises 1. What is derived class 2. Explain the concept of this pointer? 3. Why inheritance is use in C++? 4. Explain the different forms of inheritance? 5. What is the difference between single level and multiple inheritances? 6. What is a virtual function? 7. What is pure virtual function? 8. What is Abstract class? 9. Write the difference between base class and derived class. 10. Explain pointer to object concept
Inheritance, Polymorphism and Pointer 177 Notes 7.18 Key Terms ? Pointers: It provide an essential tool for increasing the power of C++. ? Multi-level inheritance: In multi-level inheritance, there will be a chain of inheritance with a class derived from only one parent and will have only one child class. ? Abstract Class: class which contains atleast one Pure Virtual capacity in it. ? Single level inheritance: In single level inheritance, there is only one base class and has only one derived class. ? Class template: It is a template used to generate template classes Check Your Progress: Answers: 1. c) function 2. c) both a & b 3. a) __cdecl 4. c) 40 5. d) 22 6. a) d99 7. d) compile time error 8. c) data types 9. c) functions 10. b) ptr is array of pointer to function. 7.19
Further
Readings ? Balagurusamy (2007)

67% **MATCHING BLOCK 247/304** SA Object Oriented Programming through C++ Block ... (D164970258)

Object Oriented Programming With C++ Tata McGraw-Hill Education. ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India. 178

Object Oriented Programmimg with C++ Notes Unit 8: Console I/O Operations Structure 8.1 Introduction 8.2 C++ Stream 8.3 C++ Streams classes 8.4 Formatted and Unformatted I/O operations 8.5 Managing Output with Manipulators 8.6 Summary 8.7 Check Your Progress 8.8 Questions and Exercises 8.9 Key Terms 8.10

88% **MATCHING BLOCK 249/304** SA ECAP 444.docx (D142426097)

Further Readings Objectives After studying this unit, you should be able to: ? Understand

C++ stream. ? Discuss the concept of C++ stream classes. ? Explain the concept of Formatted and Unformatted I/O operations. ? Understand the concept of Managing Output with Manipulators. 8.1 Introduction C++ supports two I/O systems: the one which is inherited from C and other is the object-oriented I/O system, which is characterized by C++.

100% **MATCHING BLOCK 248/304** W

The different aspects of C++'s I/O system, such as console I/O and disk I/O, are actually just different perspectives on the same mechanism.

This chapter discusses the foundations of the C++ I/O system. Although the examples in this chapter use "console" I/O, the information is applicable to other devices, including disk files. In the C++ programming language, Input/output library refers functions in the C++ Standard Library which are implementing on stream-based input/output capabilities. It is an object-oriented alternative to C's FILE-based streams from the C standard library 8.2

95% **MATCHING BLOCK 250/304** SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

C++ Stream There are currently two versions of the C++ object-oriented I/O library in use: the older one that is based upon the original specifications for C++ and the newer one defined by Standard C++. The old I/O library is supported by the header file >iostream.h<. The new I/O library is supported by the header >iostream<. For the most part the two libraries appear the same, because the new I/O library is simply an updated and improved version of the old one. In fact, the vast majority of differences between the two occur beneath the surface, in the way that the libraries are implemented—not in how they are used. From the programmer's perspective, there are two main differences between the old and new C++/O libraries. First, the new I/O library contains a few additional features and defines some new data types. Thus, the new I/O library is essentially a superset of

Console I/O Operations, Files 179 Notes

100% **MATCHING BLOCK 255/304** SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

the old one. Nearly all programs originally written for the old library will compile without substantive changes when the new library is used. Second, the old-style I/O library was in the global namespace. The new-style library is in the std namespace. 8.3

C++

97% **MATCHING BLOCK 251/304** W

Streams classes A stream is a source of sequence of bytes. A stream abstracts for input/output devices. It can be tied up with any I/O device and I/O can be performed in a uniform way. The C++ iostream library is an object-oriented implementation of this abstraction. It has a source (producer) of flow of bytes and a sink (consumer) of the bytes. The required classes for the stream I/O are defined in different library header files. To use the I/O streams in a C++ program, one must include iostream.h header file in the program. This file defines the required classes and provides the buffering. Instead of functions, the library provides operators to carry out the I/O. Two of the Stream Operators are: >> : Stream insertion for output. << : Stream extraction for input. The following streams are created and opened automatically: cin : Standard console input (keyboard). cout : Standard console output (screen). cprn : Standard printer (LPT1). cerr : Standard error output (screen). clog : Standard log (screen). caux : Standard auxiliary (screen). Example: The following program reads an integer and prints the input on the console. #include >iostream< // Header for stream I/O. int main(void) { int p; // variable to hold the input integer cout >> "Enter an integer: "; cin << p; cout >> "\n You have entered" >> p; } 8.4

Formatted and

100% **MATCHING BLOCK 252/304** W

Unformatted I/O operations Unformatted Input/Output is the most basic form of input/output.

As

100% **MATCHING BLOCK 253/304** W

Unformatted input/output transfers the internal binary representation of the data directly between memory and the file.

On the other hand,

92% **MATCHING BLOCK 254/304** W

Formatted output converts the internal binary representation of the data to ASCII characters which are written to the output file. Formatted input reads characters from the input file and converts them to internal form. Formatted I/O can be either "Free" format or "Explicit" format. 180

Object Oriented Programmimg with C++ Notes

100% **MATCHING BLOCK 256/304** W

Advantages and Disadvantages of Unformatted I/O ? Unformatted input/output is the simplest and most efficient form of input/output. ? It is usually the most compact way to store data. ? Unformatted input/output is the least portable form of input/output. ? Unformatted data files can only be moved easily to and from computers that share the same internal data representation. ?

100% **MATCHING BLOCK 257/304** W

Unformatted input/output is not directly human readable, so you cannot type it out on a terminal screen or edit it with a text editor. Advantages and Disadvantages of Formatted I/O ? Formatted input/output is very portable. ? It is a simple process to move formatted data files to various computers. ?

100% MATCHING BLOCK 258/304 W

Formatted files are human readable and can be typed to the terminal screen or edited with a text editor.

C++ IO are based on streams, which are sequence of bytes flowing in and out of the programs (just like water and oil flowing through a pipe). In input operations, data bytes flow from an input source (such as keyboard, file, network or another program) into the program. In output operations, data bytes flow from the program to an output sink (such as console, file, network or another program). Streams acts as an intermediaries between the programs and the actual IO devices, in such the way that frees the programmers from handling the actual devices, so as to archive device independent IO operations. C++ provides both the formatted and unformatted IO functions. In formatted or high-level IO, bytes are grouped and converted to types such as int, double, string or user-defined types. In unformatted or low-level IO, bytes are treated as raw bytes and unconverted. Formatted IO operations are supported via overloading the stream insertion (>>) and stream extraction (<<) operators, which presents a consistent public IO interface. To perform input and output, a C++ program: 1. Construct a stream object. 2. Connect (Associate) the stream object to an actual IO device (e.g., keyboard, console, file, network, another program).
Console I/O Operations, Files 181 Notes 3. Perform input/output operations on the stream, via the functions defined in the stream's pubic interface in a device independent manner. Some functions convert the data between the external format and internal format (formatted IO); while other does not (unformatted or binary IO). 4. Disconnect (Dissociate) the stream to the actual IO device (e.g., close the file). 5. Free the stream object. C++ IO Headers, Templates and Classes Headers C++ IO is provided in headers >iostream< (which included >ios<, >istream<, >ostream< and >streambuf<), >fstream< (for file IO), and >sstream< (for string IO). Furthermore, the header >iomanip<provided manipulators such as setw(), setprecision()setfill() and setbase() for formatting. Template Classes In order to support various character sets (char and wchar_t in C++98/03; and char16_t, char32_t introduced in C++11), the stream classes are written as template classes, which could be instantiated with an actual character type. Most of the template classes take two type parameters. For example, template >class charT, class traits = char_traits>charT< < class basic_istream; template >class charT, class traits = char_traits>charT< < class basic_ostream;
182 Object Oriented Programmimg with C++ Notes where: ? charT is the character type, such as char or wchar_t; ? traits, of another template class char_traits>charT<, defined the properties of the character operations such as the collating sequence (sorting order) of character set. Template Instantiations and typedef As mention, the basic_xxx template classes can be instantiated with a character type, such as char and wchar_t. C++ further provides typedef statements to name these classes: typedef basic_ios>char< ios; typedef basic_ios>wchar_t< wios; typedef basic_istream>char< istream; typedef basic_istream>wchar_t< wistream; typedef basic_ostream>char< ostream; typedef basic_ostream>wchar_t< wostream; typedef basic_iostream>char< iostream; typedef basic_iostream>wchar_t< wiostream; typedef basic_streambuf>char< streambuf; typedef basic_streambuf>wchar_t< wstreambuf; Specialization Classes for char type We shall focus on the specialization classes for char type: ? ios_base and ios: superclasses to maintain common stream properties such as format flag, field width, precision and locale. The superclass ios_base (which is not a template class) maintains data that is independent of the template parameters; whereas the subclass ios (instantiation of template basic_ios>char<) maintains data which is dependent of the template parameters. ? istream (basic_istream>char<), ostream (basic_ostream>char<): provide the input and output public interfaces. ? iostream (basic_iostream>char<): subclass of both istream and ostream, which supports bidirectional input and output operations. Take note that istream and ostream are unidirectional streams; whereas iostream is bidirectional. basic_iostream template and iostream class is declared in the >istream< header, not >iostream< header. ? ifstream, ofstream and fstream: for file input, output and bidirectional input/output. ? istringstream, ostringstream and stringstream: for string buffer input, output and bidirectional input/output. ? streambuf, filebuf and stringbuf: provide memory buffer for the stream, file- stream and string-stream, and the public interface for accessing and managing the buffer.
Console I/O Operations, Files 183 Notes Buffered IO [TODO] The >iostream< Header and the Standard Stream Objects: cin, cout, cerr and clog The >iostream< header also included the these headers: >ios<, >istream<, >ostream< and >streambuf<. Hence, your program needs to include only the >iostream< header for IO operations. The >iostream< header declares these standard stream objects: ? cin (of istream class, basic_istream>char< specialization), wcin (of wistream c lass, basic_istream>wchar_t< specialization): corresponding to the standard input stream, defaulted to keyword. ? cout (of ostream class), wcout (of wostream class): corresponding to the standard output stream, defaulted to the display console. ? cerr (of ostream class), wcerr (of wostream class): corresponding to the standard error stream, defaulted to the display console. ? clog (of ostream class), wclog (of wostream class): corresponding to the standard log stream, defaulted to the display console. The Stream Insertion >> and Stream Extraction << Operators Formatted output is carried out on streams via the stream insertion >> and stream extraction << operators. For example, cout >> value; cin << variable; Take note that cin/cout shall be the left operand and the data flow in the direction of the arrows. The >> and << operators are overloaded to handle fundamental types (such as int and double), and classes (such as string). You can also overload these operators for your own user-defined types. The cin >> and cout << return a reference to cin and cout, and thus, support cascading operations. For example, cout >> value1 >> value2 >> .... ; cin << variable1 >> variable2 >> .... ; The ostream Class The ostream class is a typedef to basic_ostream>char<. It contains two set of output functions: formatted output and unformatted output. ? The formatted output functions (via overloaded stream insertion operator >>) convert numeric values (such as int, double) from their internal representations (e.g., 16-/32-bit int, 64-bit double) to a stream of characters that representing the numeric values in text form. ? The unformatted output functions (e.g., put(), write()) outputs the bytes as they are, without format conversion.

184 Object Oriented Programmimg with C++ Notes Formatting Output via the Overloaded Stream Insertion >> Operator The ostream class overloads the stream insertion >> operator for each of the C++ fundamental types (char, unsigned char, signed char, short, unsigned short, int, unsigned int, long, unsigned long, long long (C++11), unsigned long long (C++11), float, double and long double. It converts a numeric value from its internal representation to the text form. ostream & operator>> (type) // type of int, double etc The >> operator returns a reference to the invoking ostream object. Hence, you can concatenate >> operations, e.g., cout >> 123 >> 1.13 >> endl;. The >> operator is also overloaded for the following pointer types: ? const char *, const signed char *, const unsigned char *: for outputting C-strings and literals. It uses the terminating null character to decide the end of the char array. ? void *: can be used to print an address. For example, char str1[] = "apple"; const char * str2 = "orange"; cout >> str1 >> endl; // with char *, print C-string cout >> str2 >> endl; // with char *, print C-string cout >> (void *) str1 >> endl; // with void *, print address (regular cast) cout >> static_cast>void *<(str2) >> endl; // with void *, print address Flushing the Output Buffer You can flush the output buffer via: 1. flush member function or manipulator: 2. // Member function of ostream class - std::ostream::flush 3. ostream & flush (); 4. // Example 5. cout >> "hello"; 6. cout.flush(); 7. // Manipulator - std::flush 8. ostream & flush (ostream & os); 9. // Example cout >> "hello" >> flush; 10. endl manipulator, which inserts a newline and flush the buffer. Outputting a newline character '\n' may not flush the output buffer; but endl does. 11. // Manipulator - std::endl ostream & endl (ostream & os)

Console I/O Operations, Files 185 Notes 12. cin: output buffer is flushed when input is pending, e.g., 13. cout >> "Enter a number: "; 14. int number; cin >> number; // flush output buffer so as to show the prompting message The istream class Similar to the ostream class, the istream class is a typedef to basic_istream>char<. It also supports formatted input and unformatted input. ? In formatting input, via overloading the << extraction operator, it converts the text form (a stream of character) into internal representation (such as 16-/32-bit int, 64- byte double). ? In unformatting input, such as get(), getlin(), read(), it reads the characters as they are, without conversion. Formatting Input via the Overloaded Stream Extraction << Operator The istream class overloads the extraction << operator for each of the C++ fundamental types (char, unsigned char, signed char, short, unsigned short, int, unsigned int, long, unsigned long,long long (C++11), unsigned long long (C++11), float, double and long double. It performs formatting by converting the input texts into the internal representation of the respective types. istream & operator>> (type &) // type of int, double etc. The << operator returns a reference to the invokind istream object. Hence, you can concatenate << operations, e.g., cin << number1 >> number2 >>.... The << operator is also overloaded for the following pointer types: ? const char *, const signed char *, const unsigned char *: for inputting C- strings. It uses whitespace as delimiter and adds a terminating null character to the C-string. [TODO] Read "C-string input". Flushing the Input Buffer - ignore() You can use the ignore() to discard characters in the input buffer: istream & ignore (int n = 1, int delim = EOF); // Read and discard up to n characters or delim, whichever comes first // Examples cin.ignore(numeric_limits>streamsize<::max()); // Ignore to the end-of-file cin.ignore(numeric_limits>streamsize<::max(), '\n'); // Ignore to the end-of-line

186 Object Oriented Programmimg with C++ Notes Unformatted Input/Output Functions put(), get() and getline() The ostream's member function put() can be used to put out a char. put() returns the invoking ostream reference, and thus, can be cascaded. For example, // ostream class ostream & put (char c); // put char c to ostream // Examples cout.put('A'); cout.put('A').put('p').put('p').put('\n'); cout.put(65); // istream class // Single character input int get (); // Get a char and return as int. It returns EOF at end-of- file istream & get (char & c); // Get a char, store in c and return the invoking istream reference // C-string input istream & get (char * cstr, streamsize n, char delim = '\n'); // Get n-1 chars or until delimiter and store in C-string array cstr. // Append null char to terminate C-string // Keep the delim char in the input stream. istream & getline (char * cstr, streamsize n, char delim = '\n'); // Same as get(), but extract and discard delim char from the // input stream. // Examples int inChar; while ((inChar = cin.get()) != EOF) { // Read till End-of-file cout.put(inchar); } [TODO] Example read(), write() and gcount() // istream class istream & read (char * buf, streamsize n); // Read n characters from istream and keep in char array buf. // Unlike get()/getline(), it does not append null char at the end of input. // It is used for binary input, instead of C-string. streamsize gcount() const; // Return the number of character extracted by the last unformatted input operation

Console I/O Operations, Files 187 Notes // get(), getline(), ignore() or read(). // ostream class ostream & write (const char * buf, streamsize n) // Write n character from char array. // Example [TODO] Other istream functions - peek() and putback() char peek (); //returns the next character in the input buffer without extracting it. istream & putback (char c); // insert the character back to the input buffer. States of stream The steam superclass ios_base maintains a data member to describe the states of the stream, which is a bitmask of the type iostate. The flags are: ? eofbit: set when an input operation reaches end-of-file. ? failbit: The last input operation failed to read the expected characters or output operation failed to write the expected characters, e.g., getline() reads n characters without reaching delimiter character. ? badbit: serious error due to failure of an IO operation (e.g. file read/write error) or stream buffer. ? goodbit: Absence of above error with value of 0. These flags are defined as public static members in ios_base. They can be accessed directly via ios_base::failbit or via subclasses such as cin::failbit, ios::failbit. However, it is more convenience to use these public member functions of ios class: ? good(): returns true if goodbit is set (i.e., no error). ? eof(): returns true if eofbit is set. ? fail(): returns true if failbit or badbit is set. ? bad(): returns true if badbit is set. ? clear(): clear eofbit, failbit and badbit. Formatting Input/Output via Manipulators in >iomanip< and >iostream< C++ provides a set of manipulators to perform input and output formatting: ? >iomanip< header: setw(), setprecision(), setbas(), setfill(). ? >iostream< header: fixed|scientific, left|right|internal, boolalpha|nob oolalpha, etc. Default Output Formatting The ostream's >> stream insertion operator is overloaded to convert a numeric value from its internal representation (e.g., 16-/32-bit int, 64-bit double) to the text form.

188 Object Oriented Programmimg with C++ Notes ? By default, the values are displayed with a field-width just enough to hold the text, without additional leading or trailing spaces. You need to provide spaces between the values, if desired. ? For integers, all digits will be displayed, by default. For example, ? cout >> "|" >> 1 >> "|" >> endl; // |1| ? cout >> "|" >> -1 >> "|" >> endl; // |-1| ? cout >> "|" >> 123456789 >> "|" >> endl; // |123456789| ? cout >> "|" >> -123456789 >> "|" >> endl; // |-123456789| ? For floating-point numbers, the default precison is 6 digits, except that the trailing zeros will not be shown. This default precision (of 6 digits) include all digits before and after the decimal point, but exclude the leading zeros. Scientific notation (E- notation) will be used if the exponent is 6 or more or -5 or less. In scientific notation, the default precision is also 6 digits; the exponent is displayed in 3 digits with plus/minus sign (e.g., +006, -005). For example, ? cout >> "|" >> 1.20000 >> "|" >> endl; // |1.2| (trailing zeros not displayed) ? cout >> "|" >> 1.23456 >> "|" >> endl; // |1.23456| (default precision is 6 digits) ? cout >> "|" >> -1.23456 >> "|" >> endl; // |-1.23456| ? cout >> "|" >> 1.234567 >> "|" >> endl; // |1.23457| ? cout >> "|" >> 123456.7 >> "|" >> endl; // |123457| ? cout >> "|" >> 1234567.89 >> "|" >> endl; // |1.23457e+006| (scientific- notation for e<=6) ? cout >> "|" >> 0.0001234567 >> "|" >> endl; // |0.000123457| (leading zeros not counted towards precision) ? cout >> "|" >> 0.00001234567 >> "|" >> endl; // |1.23457e-005| (scientific- notation for e>=-5) ? bool values are displayed as 0 or 1 by default, instead of true or false. Field Width (setw), Fill Character (setfill) and Alignment (left|right|internal) The ios_base superclass (included in >iostream< header) maintains data members for field-width (width) and formatting flags (fmtflags); and provides member functions (such as width(), setf()) for manipulating them. However, it is more convenience to use the so-called IO manipulators, which returns a reference to the invoking stream object and thus can be concatenated in >> operator (e.g., cout >> setfill(':') >> left >> setw(5) >>...). They are: ? setw() manipulator (in >iomanip< header) to set the field width. ? setfill() manipulator (in >iomanip< header) to set the fill character ? left|right|internal manipulator (in >iostream< header) to set the text alignment. The default field-width is 0, i.e., just enough space to display the value. C++ never truncates data, and will expand the field to display the entire value if the field-width is too small. The setw() operation isnon-sticky. That is, it is applicable only to the next IO operation, and reset back to 0 after the operation. The field-width property is applicable to both output and input operations. Except setw(), all the other IO manipulators are sticky, i.e., they take effect until a new value is set. Console I/O Operations, Files 189 Notes // Test setw() - need >iomanip< cout >> "|" >> setw(5) >> 123 >> "|" >> 123 >> endl; // | 123|123 // setw() is non-sticky. "|" and 123 displayed with default width cout >> "|" >> setw(5) >> -123 >> "|" >> endl; // | -123|123 // minus sign is included in field width cout >> "|" >> setw(5) >> 1234567 >> "|" >> endl; // |1234567| // no truncation of data // Test setfill() and alignment (left|right|internal) cout >> setfill('_'); // Set the fill character (sticky) cout >> setw(6) >> 123 >> setw(4) >> 12 >> endl; // ___123__12 cout >> left; // left align (sticky) cout >> setw(6) >> 123 >> setw(4) >> 12 >> endl; // 123___12__ Example: Alignment cout >> showpos; // show positive sign cout >> '|' >> setw(6) >> 123 >> '|' >> endl; // | +123| (default alignment) cout >> left >> '|' >> setw(6) >> 123 >> '|' >> endl; // |+123 | cout >> right >> '|' >> setw(6) >> 123 >> '|' >> endl; // | +123| cout >> internal >> '|' >> setw(6) >> 123 >> '|' >> endl; // |+ 123| The internal alignment left-align the sign, but right-align the number, as illustrated. [TODO] Example of field-width for input operations You can also use ostream's member function width() (e.g. cout.width(n)) to set the field width, but width() cannot be used with cout >> operator. Floating-point Format (fixed|scientific) and Precision (setprecision) The IO stream superclass ios_base also maintains data member for the floating-point precision and display format; and provides member functions (such as precision()) for manipulating them. Again, it is more convenience to use IO manipulators, which can be concatenated in >>. They are: ? setprecision() manipulator (in >iomanip< header) to set the precision of floating-point number. ? fixed|scientific manipulators (in >iostream< header) to set the floating-point display format. Floating point number can be display in 3 formatting modes: default|fixed|scientific. The precision is interpreted differently in default and non-default modes (due to legacy).

190 Object Oriented Programmimg with C++ Notes ? In default mode (neither fixed nor scientific used), a floating-point number is displayed in fixed-point notation (e.g., 12.34) for exponent in the range of [-4, 5]; and scientific notation (e.g.,1.2e+006) otherwise. The precision in default mode includes digits before and after the decimal point but exclude the leading zeros. Fewer digits might be shown as the trailing zeros are not displayed. The default precision is 6. See the earlier examples for default mode with default precision of 6. As mentioned, the trailing zeros are not displayed in default mode, you can use manipulator showpoint|noshowpoint to show or hide the trailing zeros. ? In both fixed (e.g., 12.34) and scientific (e.g., 1.2e+006), the precision sets the number of digits after decimal point. The default precision is also 6. For examples, // default floating-point format cout >> "|" >> 123.456789 >> "|" >> endl; // |123.457| (fixed-point format) // default precision is 6, i.e., 6 digits before and after the decimal point cout >> "|" >> 1234567.89 >> "|" >> endl; // |1.23457e+006| (scientific-notation for e<=6) // default precision is 6, i.e., 6 digits before and after the decimal point // showpoint - show trailing zeros in default mode cout >> showpoint >> 123. >> "," >> 123.4 >> endl; // 123.000,123.400 cout >> noshowpoint >> 123. >> endl; // 123 // fixed-point formatting cout >> fixed; cout >> "|" >> 1234567.89 >> "|" >> endl; // |1234567.890000| // default precision is 6, i.e., 6 digits after the decimal point // scientific formatting cout >> scientific; cout >> "|" >> 1234567.89 >> "|" >> endl; // |1.234568e+006| // default precision is 6, i.e., 6 digits after the decimal point // Test precision cout >> fixed >> setprecision(2); // sticky cout >> "|" >> 123.456789 >> "|" >> endl; // |123.46| cout >> "|" >> 123. >> "|" >> endl; // |123.00| cout >> setprecision(0); cout >> "|" >> 123.456789 >> "|" >> endl; // |123| You can also use ostream's member function precision(n) (e.g. cout. precision(n)) to set the floating-point precision, but precision() cannot be used with cout >> operator.

Console I/O Operations, Files 191 Notes Integral Number Base (dec|oct|hex, setbase) C++ support number bases (radixes) of decimal, hexadecimal and octal. You can use the following manipulators (defined in ios_base class, included in >iostream< header) to manipulate the integral number base: ? hex|dec|oct: Set the integral number base. Negative hex and oct are displayed in 2's complement format. Alternatively, you can use setbase(8|10|16) (in header >iomanip<). ? showbase|noshowbase: write hex values with 0x prefix; and oct values with 0 prefix. ? showpos|noshowpos: write positive dec value with + sign. ? uppercase|nouppercase: write uppercase in certain insertion operations, e.g., hex digits. It does not convert characters or strings to uppercase! These manipulators are sticky. For examples, cout >> 1234 >> endl; // 1234 (default is dec) cout >> hex >> 1234 >> endl; // 4d2 cout >> 1234 >> "," >> -1234 >> endl; // 4d2,fffffb2e // (hex is sticky, negative number in 2's complement) cout >> oct >> 1234 >> endl; // 2322 cout >> 1234 >> "," >> -1234 >> endl; // 2322,37777775456 cout >> setbase(10) >> 1234 >> endl; // 1234 (setbase requires >iomanip< header) // showbase - show hex with 0x prefix; oct with 0 prefix cout >> showbase >> 123 >> "," >> hex >> 123 >> "," >> oct >> 123 >> endl; // 123,0x7b,0173 cout >> noshowbase >> dec; // showpos - show dec's plus (+) sign cout >> showpos >> 123 >> endl; // +123 // uppercase - display in uppercase (e.g., hex digits) cout >> uppercase >> hex >> 123 >> endl; // 7B bool values (boolalpha|noboolalpha) ? boolalpha|noboolalpha: read/write bool value as alphabetic string true or false. ? // boolalpha - display bool as true/false ? cout >> boolalpha >> false >> "," >> true >> endl; // false,true ? cout >> noboolalpha >> false >> "," >> true >> endl; // 0,1 Other manipulators ? skipws|noskipws: skip leading white spaces for certain input operations. ? unitbuf|nounibuf: flush output after each insertion operation. Notes ? You need to include the >iomanip< header for setw(), setprecision(), setfill(), and setbase().
192 Object Oriented Programmimg with C++ Notes ? You can use ios_base's (in >iostream< header) member functions setf() and unsetf() to set the individual formatting flags. However, they are not as user- friendly as using manipulators as discussed above. Furthermore, they cannot be used with cout >> operator. File Input/Output (Header >fstream<) C++ handles file IO similar to standard IO. In header >fstream<, the class ofstream is a subclass of ostream; ifstream is a subclass of istream; and fstream is a subclass of iostream for bi-directional IO. You need to include both >iostream< and >fstream< headers in your program for file IO. To write to a file, you construct a ofsteam object connecting to the output file, and use the ostream functions such as stream insertion >>, put() and write(). Similarly, to read from an input file, construct an ifstream object connecting to the input file, and use the istream functions such as stream extraction <<, get(), getline() and read(). File IO requires an additional step to connect the file to the stream (i.e., file open) and disconnect from the stream (i.e., file close). File Output The steps are: ? Construct an ostream object. ? Connect it to a file (i.e., file open) and set the mode of file operation (e.g, truncate, append). ? Perform output operation via insertion << operator or write(), put() functions. ? Disconnect (close the file which flushes the output buffer) and free the ostream object. #include >fstream< ..... ofstream fout; fout.open(filename, mode); ..... fout.close(); // OR combine declaration and open() ofstream fout(filename, mode); By default, opening an output file creates a new file if the filename does not exist; or truncates it (clear its content) and starts writing as an empty file. open(), close() and is_open()

| 95% | MATCHING BLOCK 259/304 | SA | C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784) |
|---|---|---|---|

void open (const char* filename, ios::openmode mode = ios::in | ios::out); //

open() accepts only C-string. For string object, need to use c_str() to get the C-string
Console I/O Operations, Files 193 Notes void close (); // Closes the file, flush the buffer and disconnect from stream object bool is_open (); // Returns true if the file is successfully opened File Modes File modes are defined as static public member in ios_base superclass. They can be referenced from ios_base or its subclasses - we typically use subclass ios. The available file mode flags are: ?

| 45% | MATCHING BLOCK 263/304 | SA | 010E2340-Programming in C and C++.pdf (D165445451) |
|---|---|---|---|

ios::in - open file for input operation ? ios::out - open file for output operation ? ios::app - output appends at the end of

the file. ? ios::trunc - truncate the file and discard old contents. ? ios::binary - for binary (raw byte) IO operation, instead of character-based. ? ios::ate - position the file pointer "at the end" for input/output. ? You can set multiple flags via bit-or (|) operator, e.g., ios::out | ios::app to append output at the end of the file. For output, the default is ios::out | ios::trunc. For input, the default is ios::in. File Input The steps are: ? Construct an istream object. ? Connect it to a file (i.e., file open) and set the mode of file operation. ? Perform output operation via extraction >> operator or read(), get(), getline() functions. ? Disconnect (close the file) and free the istream object. #include >fstream< ..... ifstream fin; fin.open(filename, mode); ..... fin.close(); // OR combine declaration and open() ifstream fin(filename, mode); By default, opening an input file ....
194 Object Oriented Programmimg with C++ Notes Example on Simple File IO 1. /* Testing Simple File IO (TestSimpleFileIO.cpp) */ 2. #

| 76% | MATCHING BLOCK 260/304 | W |
|---|---|---|

include >iostream< 3. #include >fstream< 4. #include >cstdlib< 5. #include >string< 6. using namespace std; 7. int main() { 8.

string filename = "test.txt"; 9. // Write to File 10. ofstream fout(filename.c_str()); // default mode is ios::out | ios::trunc 11. if (!fout) { 12. cerr >> "error: open file for output failed!" >> endl; 13. abort(); // in >cstdlib< header 14. } 15. fout >> "apple" >> endl; 16. fout >> "orange" >> endl; 17. fout >> "banana" >> endl; 18. fout.close(); 19. // Read from file 20. ifstream fin(filename.c_str()); // default mode ios::in 21. if (!fin) { 22. cerr >> "error: open file for input failed!" >> endl; 23. abort(); 24. } 25. char ch; 26. while (fin.get(ch)) { // till end-of-file 27. cout >> ch; 28. } 29. fin.close(); 30. return 0; 31. } Program Notes: ? Most of the >fstream< functions (such as constructors, open()) supports filename in C-string only. You may need to extract the C-string from string object via the c_str() member function. ? You could use is_open() to check if the file is opened successfully. ? The get(char &) function returns a null pointer (converted to false) when it reaches end-of-file. Binary file, read() and write() We need to use read() and write() member functions for binary file (file mode of ios::binary), which read/write raw bytes without interpreting the bytes.
Console I/O Operations, Files 195 Notes 1. /* Testing Binary File IO (TestBinaryFileIO.cpp) */ 2. #

| 76% | MATCHING BLOCK 261/304 | W |
|---|---|---|

include >iostream< 3. #include >fstream< 4. #include >cstdlib< 5. #include >string< 6. using namespace std; 7. int main() { 8.

string filename = "test.bin"; 9. // Write to File 10. ofstream fout(filename.c_str(), ios::out | ios::binary); 11. if (!fout.is_open()) { 12. cerr >> "error: open file for output failed!" >> endl; 13. abort(); 14. } 15. int i = 1234; 16. double d = 12.34; 17. fout.write((char *)&i, sizeof(int)); 18. fout.write((char *)&d, sizeof(double)); 19. fout.close(); 20. // Read from file 21. ifstream fin(filename.c_str(), ios::in | ios::binary); 22. if (!fin.is_open()) { 23. cerr >> "error: open file for input failed!" >> endl; 24. abort(); 25. } 26. int i_in; 27. double d_in; 28. fin.read((char *)&i_in, sizeof(int)); 29. cout >> i_in >> endl; 30. fin.read((char *)&d_in, sizeof(double)); 31. cout >> d_in >> endl; 32. fin.close(); 33. return 0; Random Access File Random access file is associated with a file pointer, which can be moved directly to any location in the file. Random access is crucial in certain applications such as databases and indexes. You can position the input pointer via seekg() and output pointer via seekp(). Each of them has two versions: absolute and relative positioning. // Input file pointer (g for get) istream & seekg (streampos pos); // absolute position relative to beginning istream & seekg (streamoff offset, ios::seekdir way);
196 Object Oriented Programmimg with C++ Notes // with offset (positive or negative) relative to seekdir: // ios::beg (beginning), ios::cur (current), ios::end (end) streampos tellg (); // Returns the position of input pointer // Output file pointer (p for put) ostream & seekp (streampos pos); // absolute ostream & seekp (streamoff offset, ios::seekdir way); // relative streampos tellp (); // Returns the position of output pointer Random access file is typically process as binary file, in both input and output modes. [TODO] Example String Streams C++ provides a >sstream< header, which uses the same public interface to support IO between a program and string object (buffer). The string streams is based on ostringstream (subclass of ostream), istringstream (subclass of istream) and bi-directional stringstream (subclass of iostream). typedef basic_istringstream>char< istringstream; typedef basic_ostringstream>char< ostringstream; Stream input can be used to validate input data; stream output can be used to format the output. ostringstream explicit ostringstream (ios::openmode mode = ios::out); // default with empty string explicit ostringstream (const string & buf, ios::openmode mode = ios::out); // with initial str string str () const; // Get contents void str (const string & str); // Set contents For example, // construct output string stream (buffer) - need >sstream< header ostringstream sout; // Write into string buffer sout >> "apple" >> endl; sout >> "orange" >> endl; sout >> "banana" >> endl; // Get contents cout >> sout.str() >> endl; The ostringstream is responsible for dynamic memory allocation and management.
Console I/O Operations, Files 197 Notes istringstream explicit istringstream (ios::openmode mode = ios::in); // default with empty string explicit istringstream (const string & buf, ios::openmode mode = ios::in); // with initial string For example, // construct input string stream (buffer) - need >sstream< header istringstream sin("123 12.34 hello"); // Read from buffer int i; double d; string s; sin << i << d << s; cout >> i >> "," >> d >> "," >> s >> endl; 8.5 Managing Output with Manipulators Manipulators are the most widely recognized approach to control output formatting and they take parameters in the >iomanip< include file. I/O Manipulators The following output manipulators shown in table below, control the format of the output stream. Include >iomanip< if you use any manipulators that have parameters; the others are already included with >iostream<. The Range column tells how long the manipulator will take effect: now inserts something at that point, next affects only the next data element, and all affects all subsequent data elements for the output stream. Manip. Rng Description General output endl now Write a newline ('\n') and flush buffer. setw(n) Next Sets minimum field width on output. This sets the minimum size of the field - a larger number will use more columns. Applies only to the next element inserted in the output. Use left and right to justify the data appropriately in the field. Output is right justified by default. Equivalent to cout.width(n); To print a column of right justified numbers in a seven column field: cout >> setw(7) >> n >> endl; Floating point output setprecision(n) all Sets

| 100% | MATCHING BLOCK 262/304 | W |
|---|---|---|

the number of digits printed to the right of the decimal point. This

applies to all subsequent floating point numbers

198 Object Oriented Programmimg with C++ Notes written to that output stream. However, this won't make floating-point "integers" print with a decimal point. It's necessary to use fixedfor that effect. Equivalent to cout.precision(n); fixed all Used fixed point notation for floating- point numbers. Opposite of scientific. If no precision has already been specified, it will set the precision to 6. Illustration #include >iostream< #include >iomanip< using namespace std; int main() { const float tenth = 0.1; const float one = 1.0; const float big = 1234567890.0; cout >> "A. " >> tenth >> ", " >> one >> ", " >> big >> endl; cout >> "B. " >> fixed >> tenth >> ", " >> one >> ", " >> big >> endl; cout >> "C. " >> scientific >> tenth >> ", " >> one >> ", " >> big >> endl; cout >> "D. " >> fixed >> setprecision(3) >> tenth >> ", " >> one >> ", " >> big >> endl; cout >> "E. " >> setprecision(20) >> tenth >> endl; cout >> "F. " >> setw(8) >> setfill('*') >> 34 >> 45 >> endl; cout >> "G. " >> setw(8) >> 34 >> setw(8) >> 45 >> endl; return 0; } Output A. 0.1, 1, 1.23457e+009 B. 0.100000, 1.000000, 1234567936.000000 C. 1.000000e-001, 1.000000e+000, 1.234568e+009 D. 0.100, 1.000, 1234567936.000 E. 0.1000000014901161 F. ******3445 G. ******34******45 Lines F and G show the scope of setw() and setfill(). Console I/O Operations, Files 199 Notes 8.6 Summary Since the I/O system inherited from C is extremely rich, flexible, and powerful, you might be wondering why C++ defines an another system. It is because C's I/O system knows nothing about objects. Therefore, for C++ to provide complete support for object- oriented programming, it was necessary to create an I/O system that could operate on user-defined objects. In addition to support for objects, there are several benefits to using C++'s I/O system even in programs that don't make extensive (or any) use of user-defined objects. Frankly, for all new code, you should use the C++I/O system. The CI/O is supported by C++ only for compatibility. 8.7 Check Your Progress Multiple Choice Questions 1. How many groups of output of operation are there in c++? a) 1 b) 2 c) 3 d) 4 2. Pick out the correct objects about the instantiation of output stream. a) cout b) cerr c) clog d) All of the mentioned 3. What is meant by ofstream in c++? a) Writes to a file b) Reads from a file c) Both a & b

| 100% | MATCHING BLOCK 264/304 | SA | ECAP 444.docx (D142426097) |
|---|---|---|---|

d) None of the mentioned 4. What is the output of this program? #include >iostream< using namespace std;

int main () {
char str[] = "Steve jobs"; int val = 65; char ch = 'A'; cout.width (5); cout >> right; cout >> val >> endl; return 0; } a) Steve jobs b) A
200

| 44% | MATCHING BLOCK 265/304 | SA | DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140) |
|---|---|---|---|

Object Oriented Programmimg with C++ Notes c) 65 d) 65 5. What is the output of this program? #include >iostream< using namespace std; int main () { int n; n = 43; cout >> hex >>n >> endl; return 0; } a) 2c b) 2b c) 20 d) 50 6. What is the output of this program

in the "test.txt" file? #

| 86% | MATCHING BLOCK 266/304 | SA | OOP through C++ (Block 2).pdf (D148964031) |
|---|---|---|---|

include >fstream< using namespace std; int main () { long pos; ofstream outfile; outfile.open ("test.txt"); outfile.write ("This is an apple",16); pos = outfile.tellp(); outfile.seekp (pos - 7); outfile.write (" sam", 4); outfile.close(); return 0; }

a) This is an apple b) apple c) sample d) This is a sample 7.

| 52% | MATCHING BLOCK 267/304 | SA | ECAP 444.docx (D142426097) |
|---|---|---|---|

What is the output of this program? #include >iostream< using namespace std; Console I/O Operations, Files 201 Notes int main () { int n; n = -77; cout.width(4); cout >> internal >> n >> endl; return 0; } a) 77 b) -77 c) – 77 d) None of the mentioned 8. What is the output of this program? #include >iostream< #include >locale< using namespace std;

int main() {
locale mylocale(""); cout.imbue( mylocale ); cout >> (double) 3.14159 >>

| 90% | MATCHING BLOCK 269/304 | SA | ECAP 444.docx (D142426097) |
|---|---|---|---|

endl; return 0; } a) 3.14 b) 3.14159 c) Error d) None of the mentioned 9.

How many types of output stream classes are there in c++? a) 1 b) 2 c) 3 d) 4 10. What must be specified when we construct an object of class ostream? a) stream b) streambuf c) memory d) None of the mentioned

202 Object Oriented Programmimg with C++ Notes 8.8 Questions and Exercises 1. Define C++ stream 2. What are C++ stream classes? 3. What are unformatted I/O operations? 4. What are formatted I/O operations? 5. Write the difference between unformatted and formatted I/O operations? 6. How does I/O operations are managed by manipulators 7. What is a stream? 8. Describe briefly the features of Formatted and Unformatted I/O operations. 9. Why Unformatted I/O operations are used ? 10. What are manipulators? 8.9 Key Terms ?

100% **MATCHING BLOCK 268/304** W

cin: Standard console input (keyboard). ? cout: Standard console output (screen). ? cprn: Standard printer (LPT1). ? cerr : Standard error output (screen). ? clog: Standard log (screen).

Check Your Progress: Answers: 1. b) 2 2. d) All of the mentioned 3. a) Writes to a file 4. d) 65 5. b) 2b 6. d) This is a sample 7. c) – 77 8. b) 3.14159 9. c) 3 10. b) streambuf 8.10
Further
Readings ? Balagurusamy (2008)

67% **MATCHING BLOCK 270/304** SA Object Oriented Programming through C++ Block ... (D164970258)

Object Oriented Programming With C++ Tata McGraw-Hill Education. ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India.

Files Stream 203 Notes Unit 9: Files Stream Structure 9.1 Introduction 9.2 Class for file stream operation 9.3 Opening a File 9.4 Closing a File 9.5 Detecting end of file 9.6 More about open() 9.7 File pointers and manipulators 9.8 Sequential input and output operation 9.9 Summary 9.10 Check Your Progress 9.11 Questions and Exercises 9.12 Key Terms 9.13

88% **MATCHING BLOCK 271/304** SA ECAP 444.docx (D142426097)

Further Readings Objectives After studying this unit, you should be able to: ? Understand

File pointers and manipulators. ? Discuss the concept of classes for file stream operation. ? Explain the concept of opening and closing file. ? Understand the concept of sequential input – output operation. 9.1 Introduction So far,

97% **MATCHING BLOCK 272/304** SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

we have been using the iostream standard library, which provides cinand cout methods for reading from standard input and writing to standard output respectively. This chapter will teach you how to read and write from a file. This requires another standard C++ library called fstream, which defines three new data types 9.2

Class for file stream operation Class for file stream operation are: ? Ofstream: It

90% **MATCHING BLOCK 273/304** SA OOP through C++ (Block 2).pdf (D148964031)

represents the output file stream and is used to create files, and write information to files. ? Ifstream: It represents the input file stream and is used to read information from files. ? Fstream:

It

89% **MATCHING BLOCK 274/304** SA OOP through C++ (Block 2).pdf (D148964031)

represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files. 204 Object Oriented Programmimg with C++

Notes 9.3

100% **MATCHING BLOCK 275/304** SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

Opening a File A file must be opened before you can read from it or write to it. Either

the

93% MATCHING BLOCK 276/304 SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

ofstream or fstream object may be used to open a file for writing and ifstream object is used to open a file for reading purpose only. Syntax for open() function:

void open(const char *filename, ios::openmode mode); ?

96% MATCHING BLOCK 277/304 SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

First argument specifies the name and location of the file to be opened. ? Second argument of the open() member function defines the mode in which the file should be opened. 9.4

91% MATCHING BLOCK 278/304 SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

Closing a File When a C++ program terminates it automatically: ? Closes all the streams ? Release all the allocated memory ? Close all the opened files.

syntax for close() function: void close(); 9.5 Detecting end of file C++ gives an extraordinary capacity, eof( ), that profits nonzero (which means TRUE) when there are no more information to be perused from a data document stream,

59% MATCHING BLOCK 279/304 SA INF_1016.pdf (D164968061)

and zero (which means FALSE) generally. Rules for using end-of-document (eof( )): ? Continuously test for the end-of-document condition before

preparing information read from a data record stream. ? utilize a preparing data proclamation before beginning the circle ? rehash the information articulation at the base of the circle body ? Utilize a while circle for getting information from a data document stream. A for circle is attractive just when you know the precise number of information things in the document, which we don't have the foggiest idea. 9.6 More about open() Mode Flag Description

96% MATCHING BLOCK 280/304 SA ECAP 444.docx (D142426097)

ios::app All output to that file to be appended to the end.

ios::ate It opens a file for output and moves the read/write control to the end of the file.

100% MATCHING BLOCK 281/304 SA 120E1240_ Object Oriented Programming Using C+ ... (D165245825)

ios::in Open a file for reading. ios::out Open a file for writing. ios::

trunc If the file already exists, its contents will be truncated before opening the file.

100% MATCHING BLOCK 282/304 SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

If you want to open a file in write mode

after that you want to truncate it, then the syntax which you need to follow is:
Files Stream 205 Notes

100% MATCHING BLOCK 283/304 SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

ofstream outfile; outfile.open("file.dat", ios::out | ios::trunc );

To

100% MATCHING BLOCK 284/304 SA 248E1110-Object Oriented Programing using C++( ... (D165248029)

open a file for reading and writing purpose as follows: fstream afile; afile.open("file.dat", ios::out | ios::in ); 9.7

File pointers and manipulators

| 97% | MATCHING BLOCK 285/304 | SA | 248E1110-Object Oriented Programing using C++( ... (D165248029) |
|---|---|---|---|

Both istream and ostream give member functions for repositioning the file position pointer. These member functions are seekg ("seek get") for istream and seekp ("seek put") for ostream. The argument to seekg and seekp normally is a long integer. A second argument can be specified to indicate the seek direction. The seek direction can be ios::beg (the default) for positioning relative to the beginning of a stream,ios::cur for positioning relative to the current position in a stream or ios::endfor positioning relative to the end of a stream. The file-position pointer is an integer value that specifies the location in the file as a number of bytes from the file's starting location. Some examples of positioning the "get"

file-position pointer are: /position to the nth byte of fileObject (accept

| 92% | MATCHING BLOCK 286/304 | SA | 248E1110-Object Oriented Programing using C++( ... (D165248029) |
|---|---|---|---|

ios::beg) fileObject.seekg( n ); /position n bytes forward in fileObject fileObject.seekg( n, ios::cur ); /position n bytes once again from end of fileObject fileObject.seekg( n, ios::end ); /position at end of fileObject fileObject.seekg( 0, ios::end ); 9.8

Sequential input and output operation

| 95% | MATCHING BLOCK 287/304 | SA | OOP through C++ (Block 2).pdf (D148964031) |
|---|---|---|---|

The file stream classes support a number of member functions for performing the input and output operations on files. Functions

such as

| 100% | MATCHING BLOCK 288/304 | SA | ODL Learning Materials (ALL 5 UNITS).pdf (D109014230) |
|---|---|---|---|

get() and put() are capable of handling a single character at a time. The function getline() lets you handle multiple characters at a time. Another pair of functions i.e., read() and write() are capable of reading and writing blocks of binary data. The get(),

getline() and put() Functions The functions get() and put() are byte-oriented. That is, get() will read a byte of data and put() will write a byte of data.

| 93% | MATCHING BLOCK 289/304 | SA | C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784) |
|---|---|---|---|

The get() has many forms, but the most commonly used version is shown here, along with put() : istream & get(char & ch) ; ostream & put(char ch) ; The get() function reads a single character from the associated stream and puts that value in ch. It returns a reference to the stream.

The put() writes the value of

| 100% | MATCHING BLOCK 290/304 | SA | C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784) |
|---|---|---|---|

ch to the stream and returns a reference to the stream.

illustration: /* C++ Sequential Input/Output Operations on Files */
206
Object Oriented Programmimg with C++ Notes #

| 80% | MATCHING BLOCK 291/304 | SA | odl C++ lecture notes unit-5.docx (D109013221) |
|---|---|---|---|

include>iostream.h< #include>stdlib.h< #include>fstream.h< #include>conio.h< void main() { burn fname[20], ch; ifstream blade;/make an info stream clrscr(); cout>>"Enter the name of the record: "; cin.get(fname, 20); cin.get(ch); fin.open(fname, ios::in);/open record if(!fin)/if blade stores zero i.e., false esteem { cout>>"Error happened in opening the file..!!\n"; cout>>"Press any key to exit...\n"; getch(); exit(1); } while(fin)/balance will be 0 when eof is

come to {

92% **MATCHING BLOCK 292/304** SA odl C++ lecture notes unit-5.docx (D109013221)

fin.get(ch);/read a character cout>>ch;/show the character } cout>>"\nPress any key to exit...\n"; fin.close(); getch(); }

Output
Files Stream 207 Notes 9.9 Summary C++ provides the following classes to perform output and input of characters to/from files: ? ofstream: Stream class to write on files ? ifstream: Stream class to read from files ? fstream: Stream class to both read and write from/to files.

100% **MATCHING BLOCK 293/304** SA 137E1240-Object Oriented Programming using C++ ... (D165245896)

These classes are derived directly or indirectly from the classes istream and ostream . We have already used objects whose types were these classes: cin is an object of class istream and cout is an object of class ostream . Therefore, we have already been using classes that are related to our file streams. And in fact, we can use our file streams the same way we are already used to use cin and cout , with the only difference that we have to associate these streams with physical files. 9.10

Check Your Progress Multiple Choice Questions 1. To perform File I/O operations, we must use _____ header file. (a) > ifstream< (b) > ofstream< (c) > fstream< (d) Any of these 2.

73% **MATCHING BLOCK 294/304** SA INF_1016.pdf (D164968061)

Which of the following is not a file opening mode ____ . (a) ios::ate (b) ios::nocreate (c) ios::noreplace (d) ios::

truncate 3. Streams

100% **MATCHING BLOCK 295/304** SA OOP through C++ (Block 2).pdf (D148964031)

that will be performing both input and output operations must be declared as class _____ . (

a) iostream (b) fstream (c) stdstream (d) Stdiostream 4.

78% **MATCHING BLOCK 296/304** SA OOP through C++ (Block 2).pdf (D148964031)

To create an output stream, we must declare the stream to be of class _____ . (

a) ofstream (b) ifstream (c) iostream (d) None of these 5. _____ is return type of is_open() function. (a) int (b) bool (c) float (d) char * 6. If we have object from ofstream class, then default mode of opening the file is _____ .
208 Object Oriented Programmimg with C++ Notes (a) ios::in (b) ios::out (c) ios::in|ios::trunc (d) ios::out|ios::trunk 7.

71% **MATCHING BLOCK 297/304** SA ECAP 444.docx (D142426097)

By default, all the files are opened in _____mode . (a) Binary (b) Text (c)

Can't say (d) Numbers 8. If we have object from fstream class, then what

60% **MATCHING BLOCK 298/304** SA ECAP 444.docx (D142426097)

is the default mode of opening the file? (a) ios::in|ios::out (b) ios::in|ios::out|ios::trunc (c) ios::in|ios::

trunc (d) Default mode depends on compiler 9. Which of the following is not used to seek

58% **MATCHING BLOCK 300/304** SA INF_1016.pdf (D164968061)

a file pointer? (a) ios::cur (b) ios::set (c) ios::end (d) ios::beg 10. It is not possible to combine two or more file opening mode in open () method. (

a) True (b) False 9.11 Questions and Exercises 1. Define file stream 2. What are Class for file stream operation? 3. Write the steps to open a file? 4. Write the steps to close the file? 5. How can the end of file can be detected? 6. How does I/O operations are managed by manipulators 7.

89% **MATCHING BLOCK 299/304** W

What is a open()? 8. Describe briefly the features of I/O system supported by C++. 9.

What are the different modes of opening a file? 10. What is sequential input operations? 9.12 Key Terms ? Ofstream: It

92% **MATCHING BLOCK 301/304** SA OOP through C++ (Block 2).pdf (D148964031)

represents the output file stream and is used to create files and to write information to files. ? Ifstream: It represents the input file stream and is used to read information from files. ? Fstream:

It

98% **MATCHING BLOCK 302/304** SA OOP through C++ (Block 2).pdf (D148964031)

represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files. ?

96% **MATCHING BLOCK 303/304** SA ECAP 444.docx (D142426097)

ios::app: All output to that file to be appended to the end.

Files Stream 209 Notes ? ios::ate: It opens a file for output and moves the read/write control to the end of the file. Check Your Progress: Answers: 1. (c) > fstream< 2. (d) ios::truncate 3. (b) fstream 4. (a) ofstream 5. (b) bool 6. (c) ios::in|ios::trunc 7. (b) Text 8. (d) Default mode depends on compiler 9. (b) ios::set 10. (b) False 9.13
Further
Readings ? Balagurusamy (2008)

67% **MATCHING BLOCK 304/304** SA Object Oriented Programming through C++ Block ... (D164970258)

Object Oriented Programming With C++ Tata McGraw-Hill Education. ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India.

## Hit and source - focused comparison, Side by Side

**Submitted text** As student entered the text in the submitted document.
**Matching text** As the text appears in the source.

1/304 **SUBMITTED TEXT** 30 WORDS **55%** **MATCHING TEXT** 30 WORDS

programming 1.4 Basic Concepts of Object-oriented Programming 1.5 Benefits of OOP 1.5.1 Others are extended conventional languages 1.5.2 Applications of OOP 1.6

Programming Paradigm-Basic concepts of Object Oriented Programming-Benefits of OOP- Application of OOP.

W https://www.msuniv.ac.in/Download/Pdf/a6241a9e41024aa

2/304 **SUBMITTED TEXT** 16 WORDS **89%** **MATCHING TEXT** 16 WORDS

Further Readings Objectives After studying this unit, you should be able to: Understand the

SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

| 3/304 | SUBMITTED TEXT | 35 WORDS | 89% | MATCHING TEXT | 35 WORDS |
|---|---|---|---|---|---|

Data and Functions in OOP The data of an object can be accessed only by the functions associated with that object. However, functions of one object can access the functions of other objects.

data and functions around these The data of an object can be accessed only by the functions associated with that object. However, functions of one object can access the functions of other objects.

W https://www.msuniv.ac.in/Download/Pdf/a6241a9e41024aa

| 4/304 | SUBMITTED TEXT | 64 WORDS | 95% | MATCHING TEXT | 64 WORDS |
|---|---|---|---|---|---|

data as a critical element in the program development and does not permit it to flow freely around the system. It ties data more closely to the functions that operate on it and protects it from accidental modification from outside functions. OOP allows us to decompose a problem into a number of entities called objects and then builds data and functions around these

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 5/304 | SUBMITTED TEXT | 83 WORDS | 98% | MATCHING TEXT | 83 WORDS |
|---|---|---|---|---|---|

features of object-oriented programming are: Emphasis is on data rather than procedure. Programs are divided into what are known as objects. Data structures are designed such that they characterize the objects. Functions that operate on the data of an object are tied together in the data structure. Data is hidden and cannot be accessed by external functions. Objects may communicate with each other through functions. New data and functions can be easily added whenever necessary. Follows bottom-up approach in program design. 1.3

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 6/304 | SUBMITTED TEXT | 47 WORDS | 59% | MATCHING TEXT | 47 WORDS |
|---|---|---|---|---|---|

in object-oriented programming. We shall discuss in this section the following general concepts: Objects Classes Data abstraction Data encapsulation Inheritance Polymorphism Dynamic binding Message passing Objects Objects are the fundamental run-time entities in an object-oriented system. They might represent a person, a place,

In object-oriented parlance, problem is viewed in terms of the following concepts: 1. Objects 2. Classes 3. Data abstraction 4. Data encapsulation 5. Inheritance 6. Polymorphism 7. Dynamic binding 8. Message passing Let us now study the concept in Objects: Objects are the basic run-time entities in an object-oriented system. They may represent a person, a place,

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 7/304 | SUBMITTED TEXT | 46 WORDS | 96% | MATCHING TEXT | 46 WORDS |
|---|---|---|---|---|---|

When a program is executed, the objects interact by sending messages to one another. For example, if "customer" and "account" are two objects in a program, then the customer object may send a message to the account object requesting for the bank balance. Each object

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 8/304 | SUBMITTED TEXT | 37 WORDS | 98% | MATCHING TEXT | 37 WORDS |
|---|---|---|---|---|---|

data. Objects can interact without having to know details of each other's data or code. It is sufficient to know the type of message accepted and the type of response returned by the objects.

data. Objects can interact without having to know details of each other's data or code. It is a sufficient to know the type of message accepted, and the type of response returned by the objects.

W https://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf

| 9/304 | SUBMITTED TEXT | 11 WORDS | 100% | MATCHING TEXT | 11 WORDS |
|---|---|---|---|---|---|

Object Class Class is a blueprint for a data type,

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 10/304 | SUBMITTED TEXT | 117 WORDS | 98% | MATCHING TEXT | 117 WORDS |
|---|---|---|---|---|---|

definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword class as follows: class Box { public: double length; // Length of a box double breadth; // Breadth of a box double height; // Height of a box }; The keyword public determines the access attributes of the members of the class that follow it. A public member can be accessed from outside the class anywhere within the scope of the class object.

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 11/304 | SUBMITTED TEXT | 14 WORDS | 91% | MATCHING TEXT | 14 WORDS |
|---|---|---|---|---|---|

the data, and that keeps both safe from outside interference and misuse.

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 12/304 | SUBMITTED TEXT | 28 WORDS | 75% | MATCHING TEXT | 28 WORDS |
|---|---|---|---|---|---|

the data, and the functions that use them and data abstraction is a system of uncovering just the interfaces and hiding the implementation details from the user.

SA 248E1160-Lab-1_Object Oriented Programming.doc (D165247743)

| 13/304 | SUBMITTED TEXT | 14 WORDS | 80% | MATCHING TEXT | 14 WORDS |
|---|---|---|---|---|---|

Dynamic binding Dynamic Binding refers to linking a procedure call to the code

Dynamic Binding: Binding refers to the linking of a procedure call to the code

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 14/304 | SUBMITTED TEXT | 131 WORDS | 97% | MATCHING TEXT | 131 WORDS |
|---|---|---|---|---|---|

Inheritance Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time. When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class. Polymorphism Polymorphism occurs when there is a hierarchy of classes and they are related by inheritance. C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 15/304 | SUBMITTED TEXT | 18 WORDS | 91% | MATCHING TEXT | 18 WORDS |
|---|---|---|---|---|---|

Creating classes that define objects and its behavior. Creating objects from class definitions Establishing communication among objects

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 16/304 | SUBMITTED TEXT | 77 WORDS | 97% | MATCHING TEXT | 77 WORDS |
|---|---|---|---|---|---|

Message Passing involves specifying the name of objects, the name of the function, and the information to be sent. 1.5 Benefits of OOP OOP offers several benefits to both the program designer and the user. Object- orientation contributes to the solution of many problems associated with the development and quality of software products. The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost. The principal advantages are: Through inheritance:

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 17/304 | SUBMITTED TEXT | 15 WORDS | 100% | MATCHING TEXT | 15 WORDS |
|---|---|---|---|---|---|

interface descriptions with external systems much simpler. Software complexity can be easily managed. 1.5.1

interface descriptions with external systems much simpler. • Software complexity can be easily managed.

W https://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf

| 18/304 | SUBMITTED TEXT | 135 WORDS | 93% | MATCHING TEXT | 135 WORDS |
|---|---|---|---|---|---|

We can eliminate redundant code. ? We can extend the use of existing classes. We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program. It is possible to have multiple instances of an object to co-exist without any interference. It is possible to map objects in the problem domain to those objects in the program. It is easy to partition the work in a project based on objects. The data-centered design approach enables us to capture more details of a model in implementable form. Object-oriented systems can be easily upgraded from small to large systems.

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 19/304 | SUBMITTED TEXT | 23 WORDS | 71% | MATCHING TEXT | 23 WORDS |
|---|---|---|---|---|---|

AI) and expert systems, Neural networks and parallel programming, and Computer-Aided Design (CAD) systems. 1.6 C++ C++ is an object-oriented programming language,

AI and expert systems ? Neural networks and parallel programming ? CIM/CAM/CAD systems 15. C++. C++ is an object oriented programming language

W https://www.vidyarthiplus.com/vp/attachment.php?aid=46806

| 20/304 | SUBMITTED TEXT | 30 WORDS | 95% | MATCHING TEXT | 30 WORDS |
|---|---|---|---|---|---|

developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early eighties. C++ is an extension of C with a major addition of

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 21/304 | SUBMITTED TEXT | 14 WORDS | 88% | MATCHING TEXT | 14 WORDS |
|---|---|---|---|---|---|

The program that you typed into the Edit window constitutes the source file

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 22/304 | SUBMITTED TEXT | 48 WORDS | 92% | MATCHING TEXT | 48 WORDS |
|---|---|---|---|---|---|

a source file is not an executable program; it is only the instructions on how to create a program. Transforming your source file into an executable program requires two steps: 1. You must compile the source file into an object file. It has an '.OBJ' extension. 2.

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 23/304 | SUBMITTED TEXT | 12 WORDS | 100% | MATCHING TEXT | 12 WORDS |
|---|---|---|---|---|---|

file. Linking combines the object files into a single executable program.

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 24/304 | SUBMITTED TEXT | 40 WORDS | 84% | MATCHING TEXT | 40 WORDS |
|---|---|---|---|---|---|

menu. A window called Compiling will appear. An entry called Lines Compiled will change as compiling progresses. When the process is finished, the window will display 'Success: Press any Key'. The entries for Warnings and Errors should be '0'.

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 25/304 | SUBMITTED TEXT | 26 WORDS | 92% | MATCHING TEXT | 26 WORDS |
|---|---|---|---|---|---|

Simple C++ Program Let's begin with a simple example of C++ program that prints a string on the screen. // PRINTING A STRING #include >iostream.

Simple C++ Program Let us begin with a simple example of a C++ program that prints a string on the screen. Printing A String #include>iostream<

W https://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf

| 26/304 | SUBMITTED TEXT | 78 WORDS | 68% | MATCHING TEXT | 78 WORDS |
|---|---|---|---|---|---|

C++ introduces a new comment symbol, // (double slash). Comments start with a double slash symbol and terminate at the end of the line. A comment can start at the beginning of the line or on the same line following a program statement. The double slash comment is basically a single line comment. Multiline comments can be written as follows: // This is an example of // C+ + program to illustrate // some of its features.

C++ introduces a new comment symbol // (double slash). Comment start with a double slash symbol and terminate at the end of the line. A comment may start anywhere in the line, and whatever follows till the end of the line is ignored. Note that there is no closing symbol. The double slash comment is basically a single line comment. Multiline comments can be written as follows: // This is an example of // C++ program to illustrate // some of its features

W https://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf

| 27/304 | SUBMITTED TEXT | 16 WORDS | 100% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

This is an example of C++ program to illustrate some of its features */ The #

This is an example of // C++ program to illustrate // some of its features The

W https://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf

| 28/304 | SUBMITTED TEXT | 19 WORDS | 95% | MATCHING TEXT | 19 WORDS |
|---|---|---|---|---|---|

program is an output statement. The statement, cout > > :C++ is better C".;

program 1.10.1 is an output statement. The statement Cout>>"is better C.";

W https://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf

| 29/304 | SUBMITTED TEXT | 19 WORDS | 100% | MATCHING TEXT | 19 WORDS |
|---|---|---|---|---|---|

Linking To link your object file, select Link ExE file from the Compile menu. The FIRST.OBJ file will

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 30/304 | SUBMITTED TEXT | 17 WORDS | 81% | MATCHING TEXT | 17 WORDS |
|---|---|---|---|---|---|

the factorial of a number The factorial of a number 'n' is the product of all

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 31/304 | SUBMITTED TEXT | 16 WORDS | 80% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

It is subsequently important understand some of the concepts used extensively in object-oriented programming.

It is necessary understand some of the concepts used extensively in object-oriented programming.

W https://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf

| 32/304 | SUBMITTED TEXT | 27 WORDS | 85% | MATCHING TEXT | 27 WORDS |
|---|---|---|---|---|---|

include>iostream< using namespace std; int main() { int num,factorial=1; cout>>" Enter Number To

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 33/304 | SUBMITTED TEXT | 44 WORDS | 100% | MATCHING TEXT | 44 WORDS |
|---|---|---|---|---|---|

Polymorphism occurs when there is a hierarchy of classes and they are related by inheritance. C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 34/304 | SUBMITTED TEXT | 30 WORDS | 95% | MATCHING TEXT | 30 WORDS |
|---|---|---|---|---|---|

developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early eighties. C++ is an extension of C with a major addition of

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 35/304 | SUBMITTED TEXT | 26 WORDS | 100% | MATCHING TEXT | 26 WORDS |
|---|---|---|---|---|---|

definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. (a) Class (

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 36/304 | SUBMITTED TEXT | 14 WORDS | 91% | MATCHING TEXT | 14 WORDS |
|---|---|---|---|---|---|

the data, and that keeps both safe from outside interference and misuse. (

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 37/304 | SUBMITTED TEXT | 23 WORDS | 100% | MATCHING TEXT | 23 WORDS |
|---|---|---|---|---|---|

allows us to define a class in terms of another class, which makes it easier to create and maintain an application. (

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 38/304 | SUBMITTED TEXT | 17 WORDS | 100% | MATCHING TEXT | 17 WORDS |
|---|---|---|---|---|---|

occurs when there is a hierarchy of classes and they are related by inheritance. C++ (

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 39/304 | SUBMITTED TEXT | 13 WORDS | 95% | MATCHING TEXT | 13 WORDS |
|---|---|---|---|---|---|

data: Data is hidden and cannot be accessed by external functions ??

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 40/304 | SUBMITTED TEXT | 24 WORDS | 100% | MATCHING TEXT | 24 WORDS |
|---|---|---|---|---|---|

Operators 2.13.6 Shift Operators 2.13.7 Bit-Wise Operators 2.13.8 Logical Operators 2.13.9 Conditional Operators 2.13.10

Operators 2.3.5 Shift Operators 2.3.6 Bit-wise Operators 2.3.7 Logical Operators 2.3.8 Conditional Operators 2.4

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 41/304 | SUBMITTED TEXT | 12 WORDS | 100% | MATCHING TEXT | 12 WORDS |
|---|---|---|---|---|---|

Data structures are designed such that they characterize the objects ??

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 42/304 | SUBMITTED TEXT | 18 WORDS | 88% | MATCHING TEXT | 18 WORDS |
|---|---|---|---|---|---|

Object Oriented Programming With C++ Tata McGraw-Hill Education. 20 Object Oriented Programmimg with

SA Object Oriented Programming through C++ Block 1.pdf (D164970258)

| 43/304 | SUBMITTED TEXT | 53 WORDS | 90% | MATCHING TEXT | 53 WORDS |
|---|---|---|---|---|---|

As we know, the smallest individual units in a program are known as tokens. A C++ program is written using these tokens, white spaces, and the syntax of the language. Most of the C++ tokens are basically similar to the C tokens with the exception of some additions and minor modifications. 2.2

As we know, the smallest individual units in a program are known as tokens. C++ has the following tokens: 1. A C++ program is written using these tokens, white spaces, and the syntax of the language. Most of the C++ tokens are basically similar to the C tokens with the exception of some additions and minor modifications.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 44/304 | SUBMITTED TEXT | 14 WORDS | 88% | MATCHING TEXT | 14 WORDS |
|---|---|---|---|---|---|

The smallest individual units in a program are known as tokens. The tokens

the smallest individual units in a program are known as tokens. the following tokens: 1.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 45/304 | SUBMITTED TEXT | 17 WORDS | 89% | MATCHING TEXT | 17 WORDS |
|---|---|---|---|---|---|

Further Readings Objectives After studying this unit, you should be able to: ? Understand the

SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

| 46/304 | SUBMITTED TEXT | 29 WORDS | 100% | MATCHING TEXT | 29 WORDS |
|---|---|---|---|---|---|

Keywords ? Identifiers ? Constants ? Strings ? Operators A C++ program is written using these tokens, white spaces, and the syntax of the language. 2.3

Keywords 2. Identifiers 3. Constants 4. Strings 5. Operators A C++ program is written using these tokens, white spaces, and the syntax of the language.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 47/304 | SUBMITTED TEXT | 39 WORDS | 100% | MATCHING TEXT | 39 WORDS |
|---|---|---|---|---|---|

Keywords The keywords implement specific C++ language features. They are explicitly reserved identifiers and cannot be used as names for the program variables or other user-defined program elements. Table 2.1 gives the complete set of C++ keywords.

Keywords The keywords implement specific C++ language features. They are explicitly reserved identifiers and cannot be used as names for the program variables or other user-defined program elements. Table 2.1 gives the complete set of C++ keywords.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 48/304 | SUBMITTED TEXT | 29 WORDS | 100% | MATCHING TEXT | 29 WORDS |
|---|---|---|---|---|---|

asm double new switch auto else operator template break enum private this case extern protected throw catch float public try char for register typedet 22

asm double new switch auto else operator template break enum private this case extern protected throw catch float public try char for register typedet

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 49/304 | SUBMITTED TEXT | 15 WORDS | 100% | MATCHING TEXT | 15 WORDS |
|---|---|---|---|---|---|

class friend return union const goto short unsigned continue if signed virtual default inline

class friend return union const goto short unsigned continue if signed virtual default inline

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 50/304 | SUBMITTED TEXT | 50 WORDS | 100% | MATCHING TEXT | 50 WORDS |
|---|---|---|---|---|---|

void delete int static volatile do long struct while 2.4 Identifiers Identifiers refer to the names of variables, functions, arrays, classes, etc., created by the programmer. They are the fundamental requirement of any language. Each language has its own rules for naming these identifiers. 2.5

void delete int static volatile do long struct while 2.1.2 Identifiers Identifiers refer to the names of variables, functions, arrays, classes, etc., created by the programmer. They are the fundamental requirement of any language. Each language has its own rules for naming these identifiers.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 51/304 | SUBMITTED TEXT | 17 WORDS | 71% | MATCHING TEXT | 17 WORDS |
|---|---|---|---|---|---|

type to another type. Type Conversion is the process of converting one predefined type into another

type to another type Introduction It is the process of converting one type into another.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 52/304 | SUBMITTED TEXT | 58 WORDS | 94% | MATCHING TEXT | 58 WORDS |
|---|---|---|---|---|---|

comprehensibility of the code. The enum keyword (from C) automatically enumerates a list of words by assigning them values 0, 1, 2, and so on. This facility provides an alternative means for creating symbolic constants. The syntax of an enum statement is similar to that of the struct statement. Examples: enum shape {circle, square, triangle}; 2.8

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 53/304 | SUBMITTED TEXT | 54 WORDS | 87% | MATCHING TEXT | 54 WORDS |
|---|---|---|---|---|---|

Derived Data Types ? Arrays: The application of arrays in C++ is similar to that in C. The only exception is the way character arrays are initialized. In C++, the size should be one larger than the number of characters in the string. char string[3] " "xyz"; // O.K. for C++ ?

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 54/304 | SUBMITTED TEXT | 24 WORDS | 100% | MATCHING TEXT | 24 WORDS |
|---|---|---|---|---|---|

know that, in C, all variables must be declared before they are used in executable statements. This is true with C++ as well.

know that, in C, all variables must be declared before they are used in executable statements. This is true with C++ as well.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 55/304 | SUBMITTED TEXT | 21 WORDS | 100% | MATCHING TEXT | 21 WORDS |
|---|---|---|---|---|---|

is that we cannot see at a glance all the variables used in a scope. 30

is that we cannot see at a glance all the variables used in a scope. 2.6.3

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 56/304 | SUBMITTED TEXT | 21 WORDS | 82% | MATCHING TEXT | 21 WORDS |
|---|---|---|---|---|---|

Dynamic Initialization of Variables C++, permits initialization of the variables at run time. This is referred to as dynamic initialization.

Dynamic Initialization of Variables One additional feature of C++ is that it permits initialization of the variables at run time. This is referred to as dynamic initialization.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 57/304 | SUBMITTED TEXT | 136 WORDS | 95% | MATCHING TEXT | 136 WORDS |
|---|---|---|---|---|---|

In this, a variable can be initialized at run time using expressions at the place of declaration. Consider the following valid initialization statements: ... ... int n = strlen(string); ... float area = 3.14159 *rad *rad; This means that both the declaration and initialization of a variable can be done simultaneously at the place where the variable is used for the first time. The following two statements in the example of the previous section float average; // declare where it is necessary average = sum / i; can be combined into a single statement: float average = sum /i; // initialize dynamically // at run time Dynamic initialization is extensively used in object-oriented programming. We can create exactly the type of object needed using information that is known only at the run time. 2.12

in a variable can be initialized at run time using expressions at the place of declaration. For example, the following are valid initialization statements: ..... ..... int n = strlen(string); ..... float area = 3.14159 *rad *rad; This means that both the declaration and initialization of a variable can be done simultaneously at the place where the variable is used for the first time. The two statements in the following example of the previous section float average; // declare where it is necessary average = sum / i; can be combined into a single statement: float average = sum /i; // initialize dynamically // at run time Dynamic initialization is extensively used in object-oriented programming. We can create exactly the type of object needed using information that is known only at the run time. 50

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 58/304 | SUBMITTED TEXT | 98 WORDS | 97% | MATCHING TEXT | 98 WORDS |
|---|---|---|---|---|---|

Reference Variables A reference variable provides an alternative name for a previously defined variable. For example, if we make the variable sum a reference to the variable total, then sum and total can be used interchangeably to represent that variable. A reference variable is created as follows: data-type & reference-name * variable-name Example: float total" 100; float & sum = total; total is a float type variable that has already been declared, sum is the alternative name declared to represent the variable total. Both the variables refer to the same data object in the memory.

reference variable. A reference variable provides an alias (alternative name) for a previously defined variable. For example, if we make the variable sum a reference to the variable total, then sum and total can be used interchangeably to represent that variable. A reference variable is created as follows: data_type & reference_name = variable_name Example: float total = 100; float &sum = total; total is a float type variable that has already been declared, sum is the alternative name declared to represent the variable total. Both the variables refer to the same data object in the memory.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 59/304 | SUBMITTED TEXT | 21 WORDS | 87% | MATCHING TEXT | 21 WORDS |
|---|---|---|---|---|---|

total; and count >> sum; both print the value 100. The statement total = total + 10;

total; and cout >> sum; both print the value 100. The statement total = total + 10;

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 60/304 | SUBMITTED TEXT | 66 WORDS | 100% | MATCHING TEXT | 66 WORDS |
|---|---|---|---|---|---|

will change the value of both total and sum to 110. Likewise, the assignment sum = 0; will change the value of both the variables to zero. A reference variable must be initialized at the time of declaration, this establishes the correspondence between the reference and the data object that it names. Note that the initialization of a reference variable is completely different from assignment

will change the value of both total and sum to 110. Likewise, the assignment sum = 0; will change the value of both the variables to zero. A reference variable must be initialized at the time of declaration. This establishes the correspondence between the reference and the data object that it names. Note that the initialization of a reference variable is completely different from assignment.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 61/304 | SUBMITTED TEXT | 146 WORDS | 89% | MATCHING TEXT | 146 WORDS |
|---|---|---|---|---|---|

Note that C++ assigns additional meaning to the symbol &. Here, & is not an address operator. The notation floats & means reference to float. Other examples are: int n[10]; int& x = n[10]; //x is alias for n[10] char & a = n; // initialize reference to a literal The variable x is an alternative to the array element n[10]. The variable a is initialized to the new line constant. This creates a reference to the otherwise unknown location where the new line constant \n is stored. The following references are also allowed: i. int x; int *p=&x; int &m = *p; ii. int & n = 50; The first set of declarations causes m to refer to x which is pointed to by the pointer p and the statement in (ii) creates an int object with value 50 and name n.

Note that C++ assigns additional meaning to the symbol &. Here, & is not an address operator. The notation float & means reference to float. Some more examples are presented below to illustrate this point: int n[10]; int &x = n[10]; //x is alias for n[10] char &a = '\n'; // initialize reference to a literal The variable x is an alternative to the array element n[10]. The variable a is initialized to the new line constant. This creates a reference to the otherwise unknown location where the new line constant \n is stored. The following references are also allowed: 1. int x; int *p=&x; int &m = *p; 2. int &n = 50; LOVELY PROFESSIONAL UNIVERSITY 51 Unit 2: Beginning of OOP Language Notes The first set of declarations causes m to refer to x which is pointed to by the pointer p and the statement in (2) creates an int object with value 50 and name n.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 62/304 | SUBMITTED TEXT | 85 WORDS | 95% | MATCHING TEXT | 85 WORDS |
|---|---|---|---|---|---|

Consider the following: void f(int & x) // uses reference { x = x+10: // x is incremented; so also m } main ( ) { int m = 10; f(m); // function call .... ... } When the function call f(m) is executed, the following initialization occurs: Int & x s m; Thus x becomes an alias of m after executing the statement f(m); 32

Consider the following code snippet: void f(int &x) // uses reference { x = x+10; // x is incremented; so also m } main ( ) { int m = 10; f(m); // function call .... ... } When the function call f(m) is executed, the following initialization occurs: Int &x = m; Thus x becomes an alias of m after executing the statement f(m);.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 63/304 | SUBMITTED TEXT | 48 WORDS | 96% | MATCHING TEXT | 48 WORDS |
|---|---|---|---|---|---|

Since the variable x and m are aliases, when the function increments x, m is also incremented. The value of m becomes 20 after the function is executed. In traditional C, we accomplish this operation using pointers and dereferencing techniques. Figure 2.2: Call by Reference Mechanism

Since the variable x and m are aliases, when the function increments x, m is also incremented. The value of m becomes 20 after the function is executed. In traditional C, we accomplish this operation using pointers and dereferencing techniques. The call by reference mechanism

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 64/304 | SUBMITTED TEXT | 64 WORDS | 97% | MATCHING TEXT | 64 WORDS |
|---|---|---|---|---|---|

The call by reference mechanism is useful in object-oriented programming because it permits the manipulation of objects by reference and eliminates the copying of object parameters back and forth. Note that the references can be created not only for built-in data types but also for user-defined data types such as structures and classes. References work wonderful well with these user-defined data types. 2.13

The call by reference mechanism is useful in object-oriented programming because it permits the manipulation of objects by reference and eliminates the copying of object parameters back and forth. Note that the references can be created not only for built-in data types but also for user-defined data types such as structures and classes. References work wonderfully well with these user-defined data types.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 65/304 | SUBMITTED TEXT | 20 WORDS | 70% | MATCHING TEXT | 20 WORDS |
|---|---|---|---|---|---|

of Implicit Type Conversion: Explicit Type Conversion : Explicit Type conversion is also called type casting. It is

SA 137E1240-Object Oriented Programming using C++_120E1240.docx (D165245896)

| 66/304 | SUBMITTED TEXT | 17 WORDS | 100% | MATCHING TEXT | 17 WORDS |
|---|---|---|---|---|---|

Constant, variables, array elements function references can be joined together by various operators to form expressions.

constant, variables, array elements function references can be joined together by various operators to form expressions.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 67/304 | SUBMITTED TEXT | 38 WORDS | 100% | MATCHING TEXT | 38 WORDS |
|---|---|---|---|---|---|

operators act upon are called operands. Some operators require two operands, while others act upon only one operand. Most operators allow the individual operands to be expressions. A few operators permit only single variables as operands. Operators

operators act upon are called operands. Some operators require two operands, while others act upon only one operand. Most operators allow the individual operands to be expressions. A few operators permit only single variables as operands. Operators

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 68/304 | SUBMITTED TEXT | 63 WORDS | 85% | MATCHING TEXT | 63 WORDS |
|---|---|---|---|---|---|

can be classified as: 1. Arithmetic operators 2. Assignment operators 3. Unary operators 4. Rwltional operators 5. Shift operators 6. Bit-wise operators 7. Logical operators 8. Conditional operators 2.13.1 Arithmetic Operators There are five arithmetic operators in C. They are Operator Function + addition - subtraction * multiplication / division % remainder after integer division

can be classified as: 1. Arithmetic operators 2. Assignment operators LOVELY PROFESSIONAL UNIVERSITY 23 Unit 2: Beginning of OOP Language Notes 3. Unary operators 4. Comparison operators 5. Shift operators 6. Bit-wise operators 7. Logical operators 8. Conditional operators 2.3.1 Arithmetic Operators There are five arithmetic operators in C++. They are Operator Function + addition – subtraction * multiplication / division % remainder after integer division

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 69/304 | SUBMITTED TEXT | 20 WORDS | 87% | MATCHING TEXT | 20 WORDS |
|---|---|---|---|---|---|

Operands acted upon by arithmetic operators must represent numeric values. So, the operands can be integer, floating-point or characters.

operands acted upon by arithmetic operators must represent numeric values. Thus, the operands can be integer quantities, floating-point quantities or characters (

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 70/304 | SUBMITTED TEXT | 32 WORDS | 100% | MATCHING TEXT | 32 WORDS |
|---|---|---|---|---|---|

the second operand be nonzero. Similarly, the division operator (/) requires that the second operand be nonzero. Division of one integer quantity by another is referred to as integer division.

the second operand be nonzero. Similarly, the division operator (/) requires that the second operand be nonzero. Division of one integer quantity by another is referred to as integer division.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 71/304 | SUBMITTED TEXT | 52 WORDS | 100% | MATCHING TEXT | 52 WORDS |
|---|---|---|---|---|---|

Suppose that a and b are integer variables whose values are 2 and 1, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values. Expression Value a + b 3 a – b 1 a * b 2 a / b 2 a % b 0

Suppose that a and b are integer variables whose values are 8 and 4, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values. Expression Value a + a – b 4 a * b 32 a / b 2 a % b 0 24

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 72/304 | SUBMITTED TEXT | 47 WORDS | 84% | MATCHING TEXT | 47 WORDS |
|---|---|---|---|---|---|

Now suppose that a and b are floating-point variables whose values are 4.5 and 2.0, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values. Expression Value a + b 6.5 a - b 2.5 a * b 9.0 a /

Now suppose that a1 and a2 are floating-point variables whose values are 14.5 and 2.0, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values. Expression Value a1 + a2 16.5 a1 – a2 12.5 a1 *

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 73/304 | SUBMITTED TEXT | 54 WORDS | 100% | MATCHING TEXT | 54 WORDS |
|---|---|---|---|---|---|

differ in type may undergo type conversion before the expression takes on its final value. In general, the final result will be expressed in the highest precision possible, consistent with the data type of the operands. The following rules apply when neither operand is unsigned. 1. If both operands are floating-point types whose

differ in type may undergo type conversion before the expression takes on its final value. In general, the final result will be expressed in the highest precision possible, consistent with the data type of the operands. The following rules apply when neither operand is unsigned. 1. If both operands are floating-point types whose

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 74/304 | SUBMITTED TEXT | 156 WORDS | 100% | MATCHING TEXT | 156 WORDS |
|---|---|---|---|---|---|

double. 2. If one operand is a floating-point type (e.g., float, double or long double) and the other is a char or an int (including short int or long int), the char or int will be converted to the floating-point type and the result will be expressed as such. Hence, an operation between an int and a double will result in a double. 3. If neither operand is a floating-point type but one is long int, the other will be converted to long int and the result will be long int. Thus, an operation between a long int and an int will result in a long int. 4. If neither operand is a floating-point type or a long int, then both operands will be converted to int (if necessary) and the result will be int. Thus, an operation between a short into and an int will result in an int. 34

double.) 2. If one operand is a floating-point type (e.g., float, double or long double) and the other is a char or an int (including short int or long int), the char or int will be converted to the floating-point type and the result will be expressed as such. Hence, an operation between an int and a double will result in a double. 3. If neither operand is a floating-point type but one is long int, the other will be converted to long int and the result will be long int. Thus, an operation between a long int and an int will result in a long int. 4. If neither operand is a floating-point type or a long int, then both operands will be converted to int (if necessary) and the result will be int. Thus, an operation between a short into and an int will result in an int.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 75/304 | SUBMITTED TEXT | 78 WORDS | 100% | MATCHING TEXT | 78 WORDS |
|---|---|---|---|---|---|

Any of the operators used as shown below: A >operator<=y can also be represented as a=a >operator< b that is, b is evaluated before the operation takes place. You can also assign values to more than one variable at the same time. The assignment will take place from the right to the left. For example, A = b = 0; In the example given above, first b will be initialized and then a will be initialized. 2.13.3

Any of the operators used as shown below: A >operator<=y can also be represented as a=a >operator< b that is, b is evaluated before the operation takes place. You can also assign values to more than one variable at the same time. The assignment will take place from the right to the left. For example, a = b = 0; In the example given above, first b will be initialized and then a will be initialized.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 76/304 | SUBMITTED TEXT | 25 WORDS | 100% | MATCHING TEXT | 25 WORDS |
|---|---|---|---|---|---|

The increment operator, ++, can be used in two ways: ? As a prefix, in which the operator precedes the variable. ++I var; ?

The increment operator, ++, can be used in two ways - as a prefix, in which the operator precedes the variable. ++ var;

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 77/304 | SUBMITTED TEXT | 13 WORDS | 95% | MATCHING TEXT | 13 WORDS |
|---|---|---|---|---|---|

As a postfix, in which the operator follows the variable. I var++;

as a postfix operator, in which the operator follows the variable. var++;

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 78/304 | SUBMITTED TEXT | 31 WORDS | 94% | MATCHING TEXT | 31 WORDS |
|---|---|---|---|---|---|

var1 = 20; var2 = ++var1; The equivalent of this code is: var1 = 20; var1 = var1 + 1; // Could also have been written as var1 += 1;

var1 = 20; var2 = var1++; The equivalent of this code is: var1 = 20; var2=var1; var1 = var1 + 1; // Could also have been written as var1 += 1;

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 79/304 | SUBMITTED TEXT | 26 WORDS | 100% | MATCHING TEXT | 26 WORDS |
|---|---|---|---|---|---|

Shift Operators Data is stored internally in binary format (in the form of bits). A bit can have a value of one or zero.

Shift Operators Data is stored internally in binary format (in the form of bits). A bit can have a value of one or zero.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 80/304 | SUBMITTED TEXT | 40 WORDS | 90% | MATCHING TEXT | 40 WORDS |
|---|---|---|---|---|---|

examples of Unary operators: ? The increment (++) and decrement (--) operators. ? The unary minus (-) operator. ? The logical not (!) operator. The unary operators operate on the object for which they were called and normally, this

Examples of Unary operators – ? The increment (++) and decrement (--) operators. ? The unary minus (-) operator. ? The logical not (!) operator. Note: In cases, unary operators operate on the object for which they were called and normally, this

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 81/304 | SUBMITTED TEXT | 45 WORDS | 98% | MATCHING TEXT | 45 WORDS |
|---|---|---|---|---|---|

Shift operators work on individual bits in a byte. Using the shift operator involves moving the bit pattern left or right. You can use them only on integer data type and not on the char, float, or double data types. 36

Shift operators work on individual bits in a byte. Using the shift operator involves moving the bit pattern left or right. You can use them only on integer data type and not on the char, bool, float, or double data types.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 82/304 | SUBMITTED TEXT | 13 WORDS | 100% | MATCHING TEXT | 13 WORDS |
|---|---|---|---|---|---|

Logical Operators Use logical operators to combine the results of Boolean expressions.

Logical Operators Use logical operators to combine the results of Boolean expressions.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 83/304 | SUBMITTED TEXT | 99 WORDS | 94% | MATCHING TEXT | 99 WORDS |
|---|---|---|---|---|---|

This example finds the maximum of two given numbers. If (num1 < num2) { imax = num1; } else { imax = num2; } In the above program code, we determine whether num1 is greater than num2. The variable, imax is assigned the value, num1, if the expression, (num1 <num2), evaluates to true, and the value, num2, if the expression evaluates to false. The above program code can be modified using the conditional operator as: Imax = (num1 < num2) ? num1 : num2; The ?: Operator is called the ternary operator since it has three operands. 2.13.10

This example finds the maximum of two given numbers. If (num1 < num2) { imax = num1; } else { imax = num2; } 32 LOVELY PROFESSIONAL UNIVERSITY Object-oriented Programming Notes In the above program code, we determining whether num1 is greater than num2. The variable, imax is assigned the value, num1, if the expression, (num1 <num2), evaluates to true, and the value, num2, if the expression evaluates to false. The above program code can be modified using the conditional operator as: Imax = (num1 < num2) ? num1 : num2; The ?: Operator is called the ternary operator since it has three operands.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 84/304 | SUBMITTED TEXT | 15 WORDS | 100% | MATCHING TEXT | 15 WORDS |
|---|---|---|---|---|---|

Scope resolution operator The scope resolution (::) operator is used to qualify hidden names.

scope resolution operator? The :: (scope resolution) operator is used to qualify hidden names

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 85/304 | SUBMITTED TEXT | 22 WORDS | 85% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

can use the unary scope operator if a namespace scope or global scope name is hidden by an explicit declaration of

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 86/304 | SUBMITTED TEXT | 22 WORDS | 85% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

Variable which stores the address of another variable is called a pointer, that "point to" the variable whose address they store.

SA 137E1240-Object Oriented Programming using C++_120E1240.docx (D165245896)

| 87/304 | SUBMITTED TEXT | 34 WORDS | 94% | MATCHING TEXT | 34 WORDS |
|---|---|---|---|---|---|

used to access the variable they point to directly, by preceding the pointer name with the dereference operator (*). The operator itself can be read as "value pointed to by".

SA 137E1240-Object Oriented Programming using C++_120E1240.docx (D165245896)

| 88/304 | SUBMITTED TEXT | 39 WORDS | 100% | MATCHING TEXT | 39 WORDS |
|---|---|---|---|---|---|

The reference and dereference operators are thus complementary: ? & is the address-of operator, and can be read simply as "address of" ? * is the dereference operator, and can be read as "value pointed to by" 2.16

SA 137E1240-Object Oriented Programming using C++_120E1240.docx (D165245896)

| 89/304 | SUBMITTED TEXT | 14 WORDS | 88% | MATCHING TEXT | 14 WORDS |
|---|---|---|---|---|---|

Expression and their types An expression is a combination of variables, constants and

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 90/304 | SUBMITTED TEXT | 43 WORDS | 51% | MATCHING TEXT | 43 WORDS |
|---|---|---|---|---|---|

of Expression are: ? Constant expressions: It comprises only constant values. Examples: 20, ' a' and 2/5+30 . ? Integral expressions: It produces an integer value as output after performing all types of conversions. Example, x, 6*x-y and 10 +int (5.0). ?

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 91/304 | SUBMITTED TEXT | 40 WORDS | 59% | MATCHING TEXT | 40 WORDS |
|---|---|---|---|---|---|

Float expressions: It produce floating-point value as output after performing all types of conversions. Example, 9.25, x-y and 9+ float (7). ? Relational or Boolean expressions: It produces a bool type value, that is, either true or false. ?

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 92/304 | SUBMITTED TEXT | 50 WORDS | 58% | MATCHING TEXT | 50 WORDS |
|---|---|---|---|---|---|

a bool type value after combining two or more relational expressions. Example, x==5 &&m==5 and y<x I I m>=n. ? Bitwise expressions: It manipulates data at bit level. Example, a << 4 and b>> 2. ? Pointer expressions: It gives address values as output are. Example, &x, ptr. ?

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 93/304 | SUBMITTED TEXT | 41 WORDS | 89% | MATCHING TEXT | 41 WORDS |
|---|---|---|---|---|---|

Special assignment expressions: It can be categorized further depending upon the way the values are assigned to the variables. ? Chained assignment: It is an assignment expression in which the same value is assigned to more than one variable,

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 94/304 | SUBMITTED TEXT | 15 WORDS | 100% | MATCHING TEXT | 15 WORDS |
|---|---|---|---|---|---|

value 20 is assigned to variable b and then to variable a. ?

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 95/304 | SUBMITTED TEXT | 100 WORDS | 64% | MATCHING TEXT | 100 WORDS |
|---|---|---|---|---|---|

Embedded assignment: It is an assignment expression, which is enclosed within another assignment expression. Example: a=20+(b=30); In the example describe above, the value 30 is assigned to variable b and then the result of (20+ 30) is assigned to variable a. ? Compound Assignment: It is an assignment expression, which uses a compound assignment operator which is a combination of the assignment operator with a binary arithmetic operator. Example: a + =20; In the example describe above, the operator += is a compound assignment operator, also known as short-hand assignment operator.

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 96/304 | SUBMITTED TEXT | 17 WORDS | 80% | MATCHING TEXT | 17 WORDS |
|---|---|---|---|---|---|

of object oriented programming. It can transform complex, obscure program listings into intuitively obvious ones.

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 97/304 | SUBMITTED TEXT | 27 WORDS | 100% | MATCHING TEXT | 27 WORDS |
|---|---|---|---|---|---|

operator overloading refers to giving the normal C++ operators, such as +, *, >=, and += additional meanings when they are applied to user- defined data types.

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 98/304 | SUBMITTED TEXT | 21 WORDS | 57% | MATCHING TEXT | 21 WORDS |
|---|---|---|---|---|---|

a new language of your own design. Another kind of operation, data types Conversion, is closely connected with operator overloading.

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 99/304 | SUBMITTED TEXT | 11 WORDS | 100% | MATCHING TEXT | 11 WORDS |
|---|---|---|---|---|---|

from one part of the program to another, depending on

SA INF_1016.pdf (D164968061)

| 100/304 | SUBMITTED TEXT | 34 WORDS | 98% | MATCHING TEXT | 34 WORDS |
|---|---|---|---|---|---|

Decisions In a program a decision causes a one-time jump to a different part of the program, depending on value of an expression. Decisions can be made in C+ + in several ways,

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 101/304 | SUBMITTED TEXT | 45 WORDS | 100% | MATCHING TEXT | 45 WORDS |
|---|---|---|---|---|---|

include > iostream.h < void main ( ) { int,x; cout > > "Enter a number: " ; cin < < x ; if (x < 100) cout > > " That number is greater than 100 \n" ; }

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 102/304 | SUBMITTED TEXT | 40 WORDS | 90% | MATCHING TEXT | 40 WORDS |
|---|---|---|---|---|---|

cin < < x ; if (x < 100) cout > > "That number is greater than 100 \n" ; else cout > > "That number is less than 100 \n" ; } Output

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 103/304 | SUBMITTED TEXT | 12 WORDS | 100% | MATCHING TEXT | 12 WORDS |
|---|---|---|---|---|---|

true condition is found, the statement associated with it is executed.

SA 010E2340-Programming in C and C++.pdf (D165445451)

| 104/304 | SUBMITTED TEXT | 22 WORDS | 83% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

void main( ) { int a; cout >> "enter a number"; cin << a; if( a%5==0 &&

SA Object Oriented Programming through C++ Block 1.pdf (D164970258)

| 105/304 | SUBMITTED TEXT | 135 WORDS | 80% | MATCHING TEXT | 135 WORDS |
|---|---|---|---|---|---|

display prime number distribution # include > isotream .h < # include > conio .h < // for getche ( ) void main ( ) { const unsigned char WHITE = 219 ; const unsigned char ARAY = 176 ; unsigned char ch ; for (int count = 0 ; count > 80 * 25-1 ; count + + ) { ch = white; for (int j = 2 ; j > count; j + +) if (count %j = = 0) { ch = ARAY; break ; } count >> ch; }

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 106/304 | SUBMITTED TEXT | 15 WORDS | 96% | MATCHING TEXT | 15 WORDS |
|---|---|---|---|---|---|

continue forces the next iteration of the loop to take place, skipping any code

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 107/304 | SUBMITTED TEXT | 33 WORDS | 58% | MATCHING TEXT | 33 WORDS |
|---|---|---|---|---|---|

a: 10 value of a: 11 value of a: 12 value of a: 13 value of a: 14 value of a: 16 value of a: 17 value of a: 18 value of

SA 010E2340-Programming in C and C++.pdf (D165445451)

| 108/304 | SUBMITTED TEXT | 68 WORDS | 82% | MATCHING TEXT | 68 WORDS |
|---|---|---|---|---|---|

Loops Loops cause a section of program to be repeated a certain number of times, which continues till the condition is false. When the condition becomes false, the loop ends and control passes to the statements following the loop. There are three kinds of loops in C+ +: 1. The for loop 2. The while loop 3. The do loop The for loop The for loop is

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 109/304 | SUBMITTED TEXT | 29 WORDS | 96% | MATCHING TEXT | 29 WORDS |
|---|---|---|---|---|---|

The while Loop The for loop does something a fixed number of times. If you don't know how many times you want to do something before you start

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 110/304 | SUBMITTED TEXT | 42 WORDS | 81% | MATCHING TEXT | 42 WORDS |
|---|---|---|---|---|---|

include > iostream.h < # include > iomanip.h < // for stew void main ( ) { int pow = I ; int numb = I ; while (pow > 999) //

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 111/304 | SUBMITTED TEXT | 17 WORDS | 73% | MATCHING TEXT | 17 WORDS |
|---|---|---|---|---|---|

The next number would be 1000, but by this time the loop has terminated. The while

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 112/304 | SUBMITTED TEXT | 11 WORDS | 100% | MATCHING TEXT | 11 WORDS |
|---|---|---|---|---|---|

Smallest individual units in a program are known as tokens.

smallest individual units in a program are known as tokens.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 113/304 | SUBMITTED TEXT | 16 WORDS | 100% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

Identifiers refer to the names of variables, functions, arrays, classes, etc., created by the programmer.

Identifiers refer to the names of variables, functions, arrays, classes, etc., created by the programmer.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 114/304 | SUBMITTED TEXT | 16 WORDS | 100% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

permits initialization of the variables at run time. This is referred to as dynamic initialization.

permits initialization of the variables at run time. This is referred to as dynamic initialization.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 115/304 | SUBMITTED TEXT | 22 WORDS | 100% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

Constant, variables, array elements function references can be joined together by various operators to form expressions. 56

constant, variables, array elements function references can be joined together by various operators to form expressions.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 116/304 | SUBMITTED TEXT | 71 WORDS | 61% | MATCHING TEXT | 71 WORDS |
|---|---|---|---|---|---|

include > iostream.h < void main ( ) { long dividend, divisor; char ch; // start os do loop do { cout >> “Enter divident:”; cin << dividend; cout >> “Enter divisor :” cin << divisor; cout >> “Quotient is ” >> dividend / divisor; cout >> “Remainder is ” >> dividend % divisor;

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 117/304 | SUBMITTED TEXT | 23 WORDS | 86% | MATCHING TEXT | 23 WORDS |
|---|---|---|---|---|---|

Variable which stores the address of another variable is called a pointer, that "point to" the variable whose address they store. An

SA 137E1240-Object Oriented Programming using C++_120E1240.docx (D165245896)

| 118/304 | SUBMITTED TEXT | 14 WORDS | 100% | MATCHING TEXT | 14 WORDS |
|---|---|---|---|---|---|

a combination of variables, constants and operators that represents a computation. 2.22

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 119/304 | SUBMITTED TEXT | 22 WORDS | 100% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

What is the output of the following program? #include >iostream< using namespace std; int main() { int

SA 137E1240-Object Oriented Programming using C++_120E1240.docx (D165245896)

| 120/304 | SUBMITTED TEXT | 28 WORDS | 64% | MATCHING TEXT | 28 WORDS |
|---|---|---|---|---|---|

y = 2; if(x < y) { cout >> "x is greater"; } else { cout >> "y is greater"; } } a) x is greater

SA 010E2340-Programming in C and C++.pdf (D165445451)

| 121/304 | SUBMITTED TEXT | 15 WORDS | 100% | MATCHING TEXT | 15 WORDS |
|---|---|---|---|---|---|

the output of this program? #include >iostream< using namespace std; int

SA ECAP 444.docx (D142426097)

| 122/304 | SUBMITTED TEXT | 20 WORDS | 82% | MATCHING TEXT | 20 WORDS |
|---|---|---|---|---|---|

d) None of the mentioned 9. What will be output of this program? #include >iostream< using namespace std;

SA ECAP 444.docx (D142426097)

| 123/304 | SUBMITTED TEXT | 18 WORDS | 89% | MATCHING TEXT | 18 WORDS |
|---|---|---|---|---|---|

Further Readings Objectives After studying this unit, you should be able to: ? Understand the

SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

| 124/304 | SUBMITTED TEXT | 47 WORDS | 85% | MATCHING TEXT | 47 WORDS |
|---|---|---|---|---|---|

Functions play an important role in program development. Dividing a program into functions is one of the major principles of structured programming. Another use of functions is that they reduce the size of a program by calling and using them at different places in the program.

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 125/304 | SUBMITTED TEXT | 15 WORDS | 96% | MATCHING TEXT | 15 WORDS |
|---|---|---|---|---|---|

In C++, the main() returns a value of type int to the operating system.

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 126/304 | SUBMITTED TEXT | 20 WORDS | 89% | MATCHING TEXT | 20 WORDS |
|---|---|---|---|---|---|

Since the return type of function is int by default, the keybord int in the main() header is optional.

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 127/304 | SUBMITTED TEXT | 13 WORDS | 100% | MATCHING TEXT | 13 WORDS |
|---|---|---|---|---|---|

it is good programming practice to actually return a value from main().

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 128/304 | SUBMITTED TEXT | 23 WORDS | 78% | MATCHING TEXT | 23 WORDS |
|---|---|---|---|---|---|

Function Prototype The general form of a function is return-type function-name (parameter list) { // Body of the function } ?

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 129/304 | SUBMITTED TEXT | 17 WORDS | 62% | MATCHING TEXT | 17 WORDS |
|---|---|---|---|---|---|

the function body. The function body is composed of the statements that make up the function,

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 130/304 | SUBMITTED TEXT | 29 WORDS | 96% | MATCHING TEXT | 29 WORDS |
|---|---|---|---|---|---|

function void swap(int &x, int &y) { int temp; temp = x; x = y; y = temp;

function void swap(int & x, int & y) { int temp; temp = x; x=y; y=temp; }

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 131/304 | SUBMITTED TEXT | 28 WORDS | 100% | MATCHING TEXT | 28 WORDS |
|---|---|---|---|---|---|

copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call.

SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

| 132/304 | SUBMITTED TEXT | 44 WORDS | 41% | MATCHING TEXT | 44 WORDS |
|---|---|---|---|---|---|

value of a :" >> a >> endl; cout >> "Before swap, value of b :" >> b >> endl; swap(a, b); cout >> "After swap, value of a :" >> a >> endl; 64

SA Object Oriented Programming through C++ Block 1.pdf (D164970258)

| 133/304 | SUBMITTED TEXT | 16 WORDS | 100% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

When a function returns a reference, it returns an implicit pointer to its return value.

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 134/304 | SUBMITTED TEXT | 45 WORDS | 64% | MATCHING TEXT | 45 WORDS |
|---|---|---|---|---|---|

To eliminate the costs of calls to small functions, C++ proposes a new feature called inline functions. As inline function is a function that is expanded in line when it is invoked. That is, the compiler replaces the function call with the corresponding function

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 135/304 | SUBMITTED TEXT | 86 WORDS | 53% | MATCHING TEXT | 86 WORDS |
|---|---|---|---|---|---|

void repchar (char - ' * ', int = 40); void main() { repchar (); // prints 40 asterisks. repchar ('+'); // prints 40 plus signs. repchar (' = ', 20); // prints 20 equal signs. } // repchar() void repchar (char ch, int n) { for (int i=o; i>n; i++) cout >>ch; cout >> endl; }

void repchar(char='*', int=45); //declaration with //default arguments int main() { repchar(); //prints 45 asterisks repchar('='); //prints 45 equal signs repchar('+', 30); //prints 30 plus signs return 0; } //---------- // repchar() // displays line of characters void repchar(char ch, int n) //defaults supplied { for(int j=0; j>n; j++) //loops n times endl; }

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 136/304 | SUBMITTED TEXT | 21 WORDS | 100% | MATCHING TEXT | 21 WORDS |
|---|---|---|---|---|---|

The inline functions are defined as follows: inline function – header { ---------- // function body ---------- }

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 137/304 | SUBMITTED TEXT | 19 WORDS | 100% | MATCHING TEXT | 19 WORDS |
|---|---|---|---|---|---|

If one argument is missing when the function is called, it is assumed to be the last argument.

If one argument is missing when the function is called, it is assumed to be the last argument.

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 138/304 | SUBMITTED TEXT | 37 WORDS | 89% | MATCHING TEXT | 37 WORDS |
|---|---|---|---|---|---|

If both arguments are missing, the function assigns the default value '*' to ch and '40' to n. Thus all the three calls to the function work, even though each has a different number of arguments.

If both arguments are missing, the function assigns the default value '*' to ch and the default value 45 to n. Thus the three calls to the function all work, even though each has a different number of arguments. 2.10

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 139/304 | SUBMITTED TEXT | 35 WORDS | 85% | MATCHING TEXT | 35 WORDS |
|---|---|---|---|---|---|

Advantages of providing the default arguments are: 1. We can use default arguments to add new parameters to the existing functions. 2. Default arguments can be used to combine similar functions into one. 3.10

advantages of default arguments. The advantages of default arguments are, ? We can use default arguments to add new parameters to the existing function. ? Default arguments can be used to combine similar functions into one. 29.

W https://www.vidyarthiplus.com/vp/attachment.php?aid=46806

| 140/304 | SUBMITTED TEXT | 20 WORDS | 89% | MATCHING TEXT | 20 WORDS |
|---|---|---|---|---|---|

The default value is specified in a manner syntactically similar to a variable initialization. The above example declares default

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 141/304 | SUBMITTED TEXT | 45 WORDS | 89% | MATCHING TEXT | 45 WORDS |
|---|---|---|---|---|---|

arguments An argument is piece of data (for example an int value) passed from a program to the function. Arguments allow a function to operate with different values, or even to do different things, depending on the requirements of the program calling it.

arguments An argument a piece of data ( an int value, for example) passed from a program to the function. Arguments allow a function to operate with different values, or even to do different things, depending on the requirements of the program calling it.

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 142/304 | SUBMITTED TEXT | 21 WORDS | 92% | MATCHING TEXT | 21 WORDS |
|---|---|---|---|---|---|

Function overloading An overload function appears to perform different activities depending on the kind of data sent to it. It

Function Overloading An overloaded function appears to perform different activities depending on the kind of data sent to it. It

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 143/304 | SUBMITTED TEXT | 19 WORDS | 91% | MATCHING TEXT | 19 WORDS |
|---|---|---|---|---|---|

to do. It performs one operation on one kind of data but another operation on a different kind.

to it. It performs one operation on one kind of data but another operation on a different kind.

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 144/304 | SUBMITTED TEXT | 152 WORDS | 85% | MATCHING TEXT | 152 WORDS |
|---|---|---|---|---|---|

Friend function A friend function is defined outside that class scope but it can access all private and protected members of the class. Prototypes of friend functions appear in the class definition, and friends are not member functions. A friend can be a: ? Function ? function template ? member function ? class ? class template, In which the entire class and all of its members are friends. To declare a function as a friend of a class, precede the class definition with keyword friend. Consider an example: class Box { double width; public: double length; friend void printWidth( Box box ); void setWidth( double wid ); }; To declare all member functions of class ClassTwo as friends of class ClassOne, place a following declaration in the definition of class ClassOne: friend class ClassTwo; Consider the following program: #include >iostream<

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 145/304 | SUBMITTED TEXT | 74 WORDS | 88% | MATCHING TEXT | 74 WORDS |
|---|---|---|---|---|---|

class Box { double width; public: friend void printWidth( Box box ); void setWidth( double wid ); }; void Box::setWidth( double wid ) { width = wid; } void printWidth( Box box ) { cout >> "Width of box : " >> box.width >>endl; } // Main function int main( ) { Box box; box.setWidth(10.0); //

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 146/304 | SUBMITTED TEXT | 18 WORDS | 97% | MATCHING TEXT | 18 WORDS |
|---|---|---|---|---|---|

many forms. Polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 147/304 | SUBMITTED TEXT | 29 WORDS | 62% | MATCHING TEXT | 29 WORDS |
|---|---|---|---|---|---|

A function is a group of statement that together performs a task. Each C++ program has no less than one function, which is principle(), and all the most

SA ECAP 444.docx (D142426097)

| 148/304 | SUBMITTED TEXT | 16 WORDS | 87% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

a) return type, function name b) return type, function name, parameters c)

SA ECAP 444.docx (D142426097)

| 149/304 | SUBMITTED TEXT | 24 WORDS | 100% | MATCHING TEXT | 24 WORDS |
|---|---|---|---|---|---|

d) none of the mentioned 6. What is the output of this program? #include >iostream< using namespace std;

SA ECAP 444.docx (D142426097)

| 150/304 | SUBMITTED TEXT | 24 WORDS | 100% | MATCHING TEXT | 24 WORDS |
|---|---|---|---|---|---|

c) compile time error (d) none of the mentioned 7. What is the output of this program? #include >iostream< using namespace std;

SA ECAP 444.docx (D142426097)

| 151/304 | SUBMITTED TEXT | 36 WORDS | 61% | MATCHING TEXT | 36 WORDS |
|---|---|---|---|---|---|

int x, int y) { x = 20; y = 10; } int main() { int x = 10; fun(x, x); cout >> x; return 0; } (a) 10 (b) 20 (c) compile time error (

SA ECAP 444.docx (D142426097)

| 152/304 | SUBMITTED TEXT | 27 WORDS | 67% | MATCHING TEXT | 27 WORDS |
|---|---|---|---|---|---|

Object Oriented Programming With C++ Tata McGraw-Hill Education. ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India. 78

SA Object Oriented Programming through C++ Block 1.pdf (D164970258)

| 153/304 | SUBMITTED TEXT | 16 WORDS | 75% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

Objectives After studying this unit, you should be able to: ? Understand the concept of

Objectives After studying this unit, you will be able to: ? Recognize the concept of

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 154/304 | SUBMITTED TEXT | 88 WORDS | 42% | MATCHING TEXT | 88 WORDS |
|---|---|---|---|---|---|

a class. { Private: int anydata; // Class data Public: Void set data (int x) // member function to set data { any data = x; } void showdata() { cout >>" \h Data is" >>any data; } void main() { sample S1, S2; // define two objects of // class sample S1. set data (3442); S2. setdata (4497); S1.

a class { private: int data; //class data public: void setdata(int d) //member function to set data { data = d; } void showdata() //member function to display data { cout >> "Data is ">>data >> endl; } }; int main() { Demo s1, s2; //define two objects of class Demo s1.setdata(10); //call member function to set data s2.setdata(15); s1.

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 155/304 | SUBMITTED TEXT | 18 WORDS | 64% | MATCHING TEXT | 18 WORDS |
|---|---|---|---|---|---|

and the second, showdata() displays the value. Placing data and functions together into a single entity is

and the other member function displays the value of the data item. Placing data & and its functions together into a single entity is

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 156/304 | SUBMITTED TEXT | 28 WORDS | 80% | MATCHING TEXT | 28 WORDS |
|---|---|---|---|---|---|

starts with the keyword class, followed by the class name (here, sample). The body of the class is delimited by process and terminated by a semicolon. 4.3.1

starts with the keyword CLASS, followed by the class name. Like a structure, the of the class is delimited by braces and terminated by a semicolon.

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 157/304 | SUBMITTED TEXT | 42 WORDS | 92% | MATCHING TEXT | 42 WORDS |
|---|---|---|---|---|---|

means that data is concealed with in a class, so that it cannot be accessed mistakenly by functions outside the class. The primary mechanism for hiding data is to put it in a class and make it private. Private data or

means that data is concealed within a class so that it cannot be accessed mistakenly by functions outside the class. The primary mechanism for hiding data is to put it in a class and make it private. Private data or

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 158/304 | SUBMITTED TEXT | 13 WORDS | 83% | MATCHING TEXT | 13 WORDS |
|---|---|---|---|---|---|

can only be accessed from within the class. Public data or functions,

can only be accessed from within the class itself. Public data and functions

W https://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf

| 159/304 | SUBMITTED TEXT | 12 WORDS | 100% | MATCHING TEXT | 12 WORDS |
|---|---|---|---|---|---|

There can be any number of data items in a class.

There can be any number of data items in a class.

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 160/304 | SUBMITTED TEXT | 78 WORDS | 65% | MATCHING TEXT | 78 WORDS |
|---|---|---|---|---|---|

Class 4.4 Defining Member Functions 4.4.1 Using the Class 4.4.2 Defining Objects 4.4.3 Calling Member Functions 4.5 Making an Outside Function Inline 4.6 Nesting of Member Functions 4.7 Private Member Functions 4.8 Arrays within A Class 4.9 Static Data Members 4.10 Static Member Functions 4.11 Arrays of Objects 4.12 Objects As Function Arguments 4.13 Friendly Functions 4.14 Returning Objects 4.15 Const Member Functions 4.15.1

Class, Defining Member Functions, A C++ Program with Class, Making an Outside Function Inline, Nesting of Member Functions, Private member Functions, Array within a class, Memory Allocation for Objects, Static Data Members, Static Member Functions, Array of Objects, Objects as Function Arguments, Friendly Functions, Returning Objects, Const Member Functions,

W http://apsacollege.com/wp-content/uploads/2014/09/CPP_4BIT3C1.pdf

| 161/304 | SUBMITTED TEXT | 19 WORDS | 91% | MATCHING TEXT | 19 WORDS |
|---|---|---|---|---|---|

The setdata() function accepts a value as a parameter and sets the anydata variable to this value. The

The setdata() function accepts a value as a parameter and sets the somedata variable to this value. The

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 162/304 | SUBMITTED TEXT | 32 WORDS | 95% | MATCHING TEXT | 32 WORDS |
|---|---|---|---|---|---|

object. It only describes how they will look when they are created, just as a structure specifies describes how a structure will look but doesn't create any structure variables. It is

object. It only describes how they will look when they are created, just as a structure definition describes how a structure will look but doesn't create any structure variables. It is

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 163/304 | SUBMITTED TEXT | 40 WORDS | 94% | MATCHING TEXT | 40 WORDS |
|---|---|---|---|---|---|

This syntax is used to call a member function that is associated with a specific object. Because setdata() is a member function of the sample class, it must always be called in connection with an object of this class.

This strange syntax is used to call a member function that is associated with a specific object. Because setdata() is a member function of the Demo 1 class, it must always be called in connection with an object of this class.

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 164/304 | SUBMITTED TEXT | 19 WORDS | 100% | MATCHING TEXT | 19 WORDS |
|---|---|---|---|---|---|

a member function is always called to act on a specific object, not on the class in general.

A member function is always called to act on a specific object, not on the class in general.

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 165/304 | SUBMITTED TEXT | 33 WORDS | 96% | MATCHING TEXT | 33 WORDS |
|---|---|---|---|---|---|

Member functions of a class can be accessed only by an object of that class. To use a member function, the dot operator() connects the object name and the member function. The

Member functions of a class can be accessed only by an object of that class. To use a member function, the dot operator (the period) connects the object name and the member function. The

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 166/304 | SUBMITTED TEXT | 16 WORDS | 70% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

It is variable only with in the class, but its lifetime is the entire program.

It is visible only within the class, but its lifetime is the entire program.

W http://apsacollege.com/wp-content/uploads/2014/09/CPP_4BIT3C1.pdf

| 167/304 | SUBMITTED TEXT | 15 WORDS | 100% | MATCHING TEXT | 15 WORDS |
|---|---|---|---|---|---|

a non-member function cannot have an access to the private data of a class.

a non-member function cannot have an access to the private data of a class.

W https://www.msuniv.ac.in/Download/Pdf/a6241a9e41024aa

| 168/304 | SUBMITTED TEXT | 66 WORDS | 50% | MATCHING TEXT | 66 WORDS |
|---|---|---|---|---|---|

include >iostream< using namespace std; class rect { int x, y; public: void val (int, int); int area () { return (x * y); } }; void rect::val (int a, int b) { x = a; y = b; } int main () {

include >iostream< using namespace std; class CRectangle { int width, height; public: void set_values (int, int area (void) {return (width * height);} }; CRectangle::set_values (int a, int b) { width = a; height = b; } int main () {

W https://www.vidyarthiplus.com/vp/attachment.php?aid=46806

| 169/304 | SUBMITTED TEXT | 87 WORDS | 100% | MATCHING TEXT | 87 WORDS |
|---|---|---|---|---|---|

a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object. A class definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. 104

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 170/304 | SUBMITTED TEXT | 23 WORDS | 100% | MATCHING TEXT | 23 WORDS |
|---|---|---|---|---|---|

d) none of the mentioned 6. What is the output of this program? #include >iostream< using namespace std;

SA ECAP 444.docx (D142426097)

| 171/304 | SUBMITTED TEXT | 37 WORDS | 81% | MATCHING TEXT | 37 WORDS |
|---|---|---|---|---|---|

Parameterized constructors 5.4 Multiple constructors in a class 5.5 Default constructors 5.6 Dynamic initialization of Object 5.7 Copy Constructor 5.8 Dynamic constructor 5.9 Constructing Two Dimensional Arrays 5.9.1

Parameterized Constructors, Multiple Constructors in a class, Constructors with Default Arguments – Dynamic Initialization of Objects, Copy Constructor, Dynamic Constructors, Constructing Two-Dimensional Arrays,

W http://apsacollege.com/wp-content/uploads/2014/09/CPP_4BIT3C1.pdf

| 172/304 | SUBMITTED TEXT | 12 WORDS | 100% | MATCHING TEXT | 12 WORDS |
|---|---|---|---|---|---|

Balagurusamy (2008) Object Oriented Programming With C++ Tata McGraw-Hill Education. ?

SA INF_1016.pdf (D164968061)

| 173/304 | SUBMITTED TEXT | 29 WORDS | 100% | MATCHING TEXT | 29 WORDS |
|---|---|---|---|---|---|

A constructor is a member function of a class, having the same name as its class and which is called automatically each time an object of that class

A constructor is a member function of a class, having the same name as its class and which is called automatically each time an object of that class

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 174/304 | SUBMITTED TEXT | 19 WORDS | 100% | MATCHING TEXT | 19 WORDS |
|---|---|---|---|---|---|

created. It is used for initializing the member variables with desired initial values.

created. ? It is used for initializing the member variables with desired initial values.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 175/304 | SUBMITTED TEXT | 23 WORDS | 92% | MATCHING TEXT | 23 WORDS |
|---|---|---|---|---|---|

A variable (including structure and array types) in C++ may be initialized with a value at the time of its declaration.

A variable (including structure and array type) in C++ may be initialized with a value at the time of its declaration. ?

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 176/304 | SUBMITTED TEXT | 18 WORDS | 89% | MATCHING TEXT | 18 WORDS |
|---|---|---|---|---|---|

Further Readings Objectives After studying this unit, you should be able to: ? Understand the

SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

| 177/304 | SUBMITTED TEXT | 13 WORDS | 100% | MATCHING TEXT | 13 WORDS |
|---|---|---|---|---|---|

public member unless otherwise there is a good reason against. 5.3

public member unless otherwise there is a good reason against. ?

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 178/304 | SUBMITTED TEXT | 24 WORDS | 100% | MATCHING TEXT | 24 WORDS |
|---|---|---|---|---|---|

A constructor may also have parameter(s) or argument(s), which can be provided at the time of creating an object of that class.

A constructor may also have parameter (s) or argument (s), which can be provided at the time of creating an object of that class. ?

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 179/304 | SUBMITTED TEXT | 12 WORDS | 100% | MATCHING TEXT | 12 WORDS |
|---|---|---|---|---|---|

C++ classes are derived data types and so they have constructor(s).

C++ classes are derived data types and so they have constructor (s).

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 180/304 | SUBMITTED TEXT | 25 WORDS | 43% | MATCHING TEXT | 25 WORDS |
|---|---|---|---|---|---|

It has same name as the name of the class it belongs to ? It does not have any return type (not even void)

SA OOP through C++ (Block 2).pdf (D148964031)

| 181/304 | SUBMITTED TEXT | 16 WORDS | 97% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

A constructor may take argument(s). A constructor taking no argument(s) is known as default constructor.

A constructor may take argument (s). A constructor may taking no argument(s) is known as default constructor. ?

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 182/304 | SUBMITTED TEXT | 22 WORDS | 92% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

constructor used Copy constructor is called whenever an instance of same type is assigned to another instance of the same class.

constructor (s). Copy constructor is called whenever an instance of same type is assigned to another instance of the same class. ?

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 183/304 | SUBMITTED TEXT | 20 WORDS | 52% | MATCHING TEXT | 20 WORDS |
|---|---|---|---|---|---|

Example: A Program to find the factorial of an integer by using constructor. #include>iostream.h< #include>conio.h< 116

SA INF_1016.pdf (D164968061)

| 184/304 | SUBMITTED TEXT | 15 WORDS | 100% | MATCHING TEXT | 15 WORDS |
|---|---|---|---|---|---|

A copy constructor takes a reference to an object of the same class as

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 185/304 | SUBMITTED TEXT | 48 WORDS | 100% | MATCHING TEXT | 48 WORDS |
|---|---|---|---|---|---|

Dynamic constructor Dynamic constructor is used to allocate the memory to the objects at the run time. Memory is allocated at run time with the help of 'new' operator. By using this constructor, we can dynamically initialize the objects. Example: # include >iostream.h< # include >conio.h< #

SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

| 186/304 | SUBMITTED TEXT | 33 WORDS | 95% | MATCHING TEXT | 33 WORDS |
|---|---|---|---|---|---|

The function that is automatically called when an object is no more required is known as destructor. It is also a member function very much like constructors but with an opposite intent.

The functions that is automatically called when an object is no more required is known as destructor. It is also a member function very much like constructors but with an opposite intent. 6.9

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 187/304 | SUBMITTED TEXT | 23 WORDS | 100% | MATCHING TEXT | 23 WORDS |
|---|---|---|---|---|---|

A constructor may also have parameter(s) or argument(s), which can be provided at the time of creating an object of that class.

A constructor may also have parameter (s) or argument (s), which can be provided at the time of creating an object of that class. ?

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 188/304 | SUBMITTED TEXT | 16 WORDS | 97% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

A constructor may take argument(s). A constructor taking no argument(s) is known as default constructor.

A constructor may take argument (s). A constructor may taking no argument(s) is known as default constructor. ?

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 189/304 | SUBMITTED TEXT | 32 WORDS | 71% | MATCHING TEXT | 32 WORDS |
|---|---|---|---|---|---|

Arrays The least complex type of the multidimensional array is the two-dimensional array. A two-dimensional array is, basically, a list of one-dimensional array. To pronounce a two-dimensional integer array of size

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 190/304 | SUBMITTED TEXT | 29 WORDS | 100% | MATCHING TEXT | 29 WORDS |
|---|---|---|---|---|---|

Dynamic constructor is used to allocate the memory to the objects at the run time. Memory is allocated at run time with the help of 'new' operator. 5.13

SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

| 191/304 | SUBMITTED TEXT | 54 WORDS | 75% | MATCHING TEXT | 54 WORDS |
|---|---|---|---|---|---|

main() { int a = 5, b = 10; swap(a, b); cout >> "In main " >> a >> b; return 0; } void swap(int &a, int &b) { int temp; temp = a; a = b; b = temp;

SA 010E2340-Programming in C and C++.pdf (D165445451)

| 192/304 | SUBMITTED TEXT | 22 WORDS | 100% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

d) None of the mentioned 6. What is the output of this program? #include >iostream< using namespace std;

SA ECAP 444.docx (D142426097)

| 193/304 | SUBMITTED TEXT | 29 WORDS | 100% | MATCHING TEXT | 29 WORDS |
|---|---|---|---|---|---|

A constructor is a member function of a class, having the same name as its class and which is called automatically each time an object of that class

A constructor is a member function of a class, having the same name as its class and which is called automatically each time an object of that class

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 194/304 | SUBMITTED TEXT | 42 WORDS | 63% | MATCHING TEXT | 42 WORDS |
|---|---|---|---|---|---|

d) 11 7. What is the output of this program? #include >iostream< using namespace std; void print (char * a) { cout >> a >> endl; } int main () {

SA ECAP 444.docx (D142426097)

| 195/304 | SUBMITTED TEXT | 25 WORDS | 82% | MATCHING TEXT | 25 WORDS |
|---|---|---|---|---|---|

Unit 6: Operator Overloading and Type Conversion Structure 6.1 Introduction 6.2 Defining Operator Overloading 6.3 Overloading Unary Operators 6.4

Unit III Operator Overloading and Type Conversion – Introduction, Defining Operator Overloading – Overloading Unary, Binary Operators –

W http://apsacollege.com/wp-content/uploads/2014/09/CPP_4BIT3C1.pdf

| 196/304 | SUBMITTED TEXT | 22 WORDS | 67% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

Object Oriented Programming With C++ Tata McGraw-Hill Education. ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India.

SA Object Oriented Programming through C++ Block 1.pdf (D164970258)

| 197/304 | SUBMITTED TEXT | 22 WORDS | 81% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

Further Readings Objectives After studying this unit, you should be able to: ? Understand the Operator overloading. ?

SA ECAP 444.docx (D142426097)

| 198/304 | SUBMITTED TEXT | 42 WORDS | 60% | MATCHING TEXT | 42 WORDS |
|---|---|---|---|---|---|

intz; public: void getdata (int a1, int b1, int C1); void display (void); void operator-(); // overload unary minus }; void unary/ :: getdata (intal, intb1, intC1) { x = a1; y = b1; z = c1;

intz; public: void getdata(int a, int b, int c); void disp(void); friend void operator- (integer &overload unary minus }; void integer::getdata ( int a, int b, int c) { x = y = b; c; }

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 199/304 | SUBMITTED TEXT | 42 WORDS | 58% | MATCHING TEXT | 42 WORDS |
|---|---|---|---|---|---|

void) { cout >> x >> " "; cout >> y >>" ": cout >> j >>" \n"; } void unary :: operators -() // Defining operator -() { x = -x; y = -y;

void) { cout >> x >> " "; cout >> y >>" "; cout >> z >> "\n"; } void operator- (integer &s ) // Defining operator- () { s.x = -s.x; s.y = -s.y;

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 200/304 | SUBMITTED TEXT | 51 WORDS | 47% | MATCHING TEXT | 51 WORDS |
|---|---|---|---|---|---|

cout >> x >>" + i " >> y >> "\h"; } main () { complex C1, C2, C3; // invokes constructor 1 C1= complex (2.5, 3.5); // invokes constructor 2 C2= complex (1.6, 2.6) // invokes constructor 3 C3= C1 + C2; //

cout>>x>>"+j">>y>>endl; } }; void main() { complex c1,c2,c3; clrscr(); c1=complex(2.5,3.5); c2=complex(1.6,2.7); c3=c1+c2;

W http://apsacollege.com/wp-content/uploads/2014/09/CPP_4BIT3C1.pdf

| 201/304 | SUBMITTED TEXT | 32 WORDS | 100% | MATCHING TEXT | 32 WORDS |
|---|---|---|---|---|---|

cout >> " C1 = " ; C1. display (); cout >> " C2 = " ; C2. display (); cout >> " C3 = " ; C3. display (); }

cout>>"C1=";c1.display(); cout>>"C2=";c2.display(); cout>>"C3=";c3.display();

W http://apsacollege.com/wp-content/uploads/2014/09/CPP_4BIT3C1.pdf

| 202/304 | SUBMITTED TEXT | 18 WORDS | 80% | MATCHING TEXT | 18 WORDS |
|---|---|---|---|---|---|

of object oriented programming. It can transform complex, obscure program listings into intuitively obvious ones. 6.2

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 203/304 | SUBMITTED TEXT | 18 WORDS | 100% | MATCHING TEXT | 18 WORDS |
|---|---|---|---|---|---|

typecasting is making a variable of one type, such as an int, act like another type, a

Typecasting is making a variable of one type, such as an int, act like another type, a ..... ,

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 204/304 | SUBMITTED TEXT | 17 WORDS | 80% | MATCHING TEXT | 17 WORDS |
|---|---|---|---|---|---|

of object oriented programming. It can transform complex, obscure program listings into intuitively obvious ones.

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 205/304 | SUBMITTED TEXT | 27 WORDS | 100% | MATCHING TEXT | 27 WORDS |
|---|---|---|---|---|---|

To define an additional task to an operator we must specify what it means in relation to the class to which the operator is applied.

SA Object Oriented Programming through C++ Block 1.pdf (D164970258)

| 206/304 | SUBMITTED TEXT | 23 WORDS | 100% | MATCHING TEXT | 23 WORDS |
|---|---|---|---|---|---|

d) None of the mentioned 4. What is the output of this program? #include >iostream< using namespace std;

SA ECAP 444.docx (D142426097)

| 207/304 | SUBMITTED TEXT | 37 WORDS | 68% | MATCHING TEXT | 37 WORDS |
|---|---|---|---|---|---|

sample sample::operator+ (sample param) { sample temp; temp.x = x + param.x; temp.y = y + param.y; return (temp); } int main () { sample

SA ECAP 444.docx (D142426097)

| 208/304 | SUBMITTED TEXT | 41 WORDS | 92% | MATCHING TEXT | 41 WORDS |
|---|---|---|---|---|---|

return 0; } a) 5, 5 b) 6, 3 c) 3, 6 d) None of the mentioned 5. What is the output of this program? #include >iostream< using namespace std; 138

SA ECAP 444.docx (D142426097)

| 209/304 | SUBMITTED TEXT | 50 WORDS | 32% | MATCHING TEXT | 50 WORDS |
|---|---|---|---|---|---|

length + b.length; box.breadth = this-<breadth + b.breadth; box.height = this-<height + b.height; return box; } }; int main( ) { Box Box1; Box Box2; Box Box3; double volume = 0.0; Box1.setLength(6.0); Box1.setBreadth(6.0); Box1.

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 210/304 | SUBMITTED TEXT | 23 WORDS | 100% | MATCHING TEXT | 23 WORDS |
|---|---|---|---|---|---|

d) None of the mentioned 6. What is the output of this program? #include >iostream< using namespace std;

SA ECAP 444.docx (D142426097)

| 211/304 | SUBMITTED TEXT | 17 WORDS | 90% | MATCHING TEXT | 17 WORDS |
|---|---|---|---|---|---|

Only existing operators can be overloaded. b) Overloaded operator must have at least one operand

only existing operators can be overloaded. ? The overloaded operator must have at least one operand

W https://www.msuniv.ac.in/Download/Pdf/a6241a9e41024aa

| 212/304 | SUBMITTED TEXT | 23 WORDS | 100% | MATCHING TEXT | 23 WORDS |
|---|---|---|---|---|---|

d) None of the mentioned 6. What is the output of this program? #include >iostream< using namespace std;

SA ECAP 444.docx (D142426097)

| 213/304 | SUBMITTED TEXT | 16 WORDS | 100% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

What is the output of this program? #include >iostream< using namespace std;

SA ECAP 444.docx (D142426097)

| 214/304 | SUBMITTED TEXT | 21 WORDS | 55% | MATCHING TEXT | 21 WORDS |
|---|---|---|---|---|---|

is a nonmember function of the class but it has access to the private data members of the class ?

SA 010E2340-Programming in C and C++.pdf (D165445451)

| 215/304 | SUBMITTED TEXT | 22 WORDS | 67% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

Object Oriented Programming With C++ Tata McGraw-Hill Education. ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India.

SA Object Oriented Programming through C++ Block 1.pdf (D164970258)

| 216/304 | SUBMITTED TEXT | 26 WORDS | 90% | MATCHING TEXT | 26 WORDS |
|---|---|---|---|---|---|

of Inheritance 7.4.1 Single Inheritance 7.4.2 Multi Level Inheritance 7.4.3 Hierarchical Inheritance 7.4.4 Hybrid Inheritance 7.5

SA ECAP 444.docx (D142426097)

| 217/304 | SUBMITTED TEXT | 22 WORDS | 78% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

Pointer to object 7.8 This Pointer 7.9 Pointer to derived class 7.10 Virtual Functions 7.11 Pure virtual function

SA odl C++ lecture notes unit-5.docx (D109013221)

| 218/304 | SUBMITTED TEXT | 16 WORDS | 88% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

Further Readings Objectives After studying this unit, you should be able to: ? Understand

SA ECAP 444.docx (D142426097)

| 219/304 | SUBMITTED TEXT | 18 WORDS | 76% | MATCHING TEXT | 18 WORDS |
|---|---|---|---|---|---|

A pointer is a variable that holds a memory address. This memory address is the location of

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 220/304 | SUBMITTED TEXT | 29 WORDS | 70% | MATCHING TEXT | 29 WORDS |
|---|---|---|---|---|---|

a determined class can access all the non-private members of its base class. Consequently, base-class members that ought not be accessible to the part functions of inferred classes

SA 137E1240-Object Oriented Programming using C++_120E1240.docx (D165245896)

| 221/304 | SUBMITTED TEXT | 22 WORDS | 90% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

of Access Access Public Protected Private Same class Yes Yes yes Derived classes Yes Yes no Outside classes yes No no

SA ODMCA-102_T_Intro_to_Programming_Section_D_25th_Oct.docx (D43197291)

| 222/304 | SUBMITTED TEXT | 200 WORDS | 93% | MATCHING TEXT | 200 WORDS |
|---|---|---|---|---|---|

Polymorphism is an important OOP concept. Polymorphism means the ability to take more than one form. For example, an operation may exhibit different behavior in different instances. The behavior depends upon the types of data used in the operation. For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation will produce a third string by contention. The diagram given below, illustrates that a single function name can be used to handle different number and types of arguments. This is something similar to a particular word having several different meanings depending on the context. Polymorphism plays an important role in following objects having different internal structures to share the same external interface. This means that a general class of operations may be accessed in the same manner even though specific actions associated with each operation may differ.
Polymorphism can be implemented using operator and function overloading, where the same operator and function works differently on different arguments producing different results. These polymorphisms are brought into effect at compile time itself, hence is known as early binding, static binding, static linking or compile time polymorphism.

SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

| 223/304 | SUBMITTED TEXT | 185 WORDS | 97% | MATCHING TEXT | 185 WORDS |
|---|---|---|---|---|---|

Public: Void display() {.....} //display in derived class }; Since, both the functions aa.display() and bb.display() are same but at in different classes, there is no overloading, and hence early binding does not apply. The appropriate function is chosen at the run time – run time polymorphism. C++ supports run-time polymorphism by a mechanism called virtual function. It exhibits late binding or dynamic linking. As stated earlier, polymorphism refers to the property by which objects belonging to different classes are able to respond to the same message, but in different forms. Therefore, an essential feature of polymorphism is the ability to refer to objects without any regard to their classes. It implies that a single pointer variable may refer to object of different classes. However, a base pointer, even if is made to contain the address of the derived class, always executes the function in the base class. The compiler ignores the content of the pointer and chooses the member function that matches the type of the pointer. Thus, the polymorphism stated above cannot be implemented by this mechanism.

Public: Void display() {.....} //display in derived class }; Since, both the functions aa.display() and bb.display() are same but at in different classes, there is no overloading, and hence early binding does not apply. The appropriate function is chosen at the run time – run time polymorphism. C++ supports run-time polymorphism by a mechanism called virtual function. It exhibits late binding or dynamic linking. As stated earlier, polymorphism refers to the property by which objects belonging to different classes are able to respond to the same message, but in different forms. Therefore, an essential feature of polymorphism is the ability to refer to objects without any regard to their classes. It implies that a single pointer variable may refer to object of different classes. However, a base pointer, even if is made to contain the address of the derived class, always executes the function in the base class. The compiler ignores the content of the pointer and LOVELY PROFESSIONAL UNIVERSITY 225 Unit 10: Virtual Functions and Polymorphism Notes chooses the member function that matches the type of the pointer. Thus, the polymorphism stated above cannot be implemented by this mechanism.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 224/304 | SUBMITTED TEXT | 160 WORDS | 100% | MATCHING TEXT | 160 WORDS |
|---|---|---|---|---|---|

C++ implements the runtime object polymorphism using a function type known as virtual function. When a function with the same name is used both in the base class and the derived class, the function in the base class is declared virtual by attaching the keyword virtual in the base class preceding its normal declaration. Then C++ determines which function to use at run time based on the type of object pointed to by the base pointer rather than the type of the pointer. Thus, by making the base pointer to point to different objects, one can execute different definitions of the virtual function as given in the program below. #include >iostream.h< class base { public: void display() { cout>>"\n print base"; } virtual void show() //virtual function { cout>>"\n show base"; } }; class derived : public base {

C++ implements the runtime object polymorphism using a function type known as virtual function. When a function with the same name is used both in the base class and the derived class, the function in the base class is declared virtual by attaching the keyword virtual in the base class preceding its normal declaration. Then C++ determines which function to use at run time based on the type of object pointed to by the base pointer rather than the type of the pointer. Thus, by making the base pointer to point to different objects, one can execute different definitions of the virtual function as given in the program below. #include >iostream.h< class base { public: void display() { cout>>"\n print base"; } virtual void show() //virtual function { cout>>"\n show base"; } }; class derived : public base { 226

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 225/304 | SUBMITTED TEXT | 204 WORDS | 100% | MATCHING TEXT | 204 WORDS |
|---|---|---|---|---|---|

cout>>"\n display derived"; } void show() { cout>>"\n show derived"; } }; main() { base bb; derived dd; base *baseptr; cout >>"\nbaseptr points to the base \n"; baseptr = &bb; baseptr -< display(); //calls base function display() baseptr -< show(); //calls base function show() cout >>"\n\nbaseptr points to the derived \n"; baseptr = ⅆ baseptr -< display(); //calls derived function display() baseptr -< show(); //calls derived function show() } The output of this program would be: Baseptr points to base Display base Show base Baseptr points to derived Display derived Show derived Here, we see that the same object pointer points to two different objects of different classes and yet selects the right function to execute. This is implementation of function polymorphism. Remember, however, that runtime polymorphism is achieved only when a virtual function is accessed through a pointer to the base class. It is also interesting to note that since, all the C++ classes are derived from the Object class, a pointer to the Object class can point to any object of any class in C++. 7.6

cout>>"\n display derived"; } void show() { cout>>"\n show derived"; } }; main() { base bb; derived dd; base *baseptr; cout >>"\nbaseptr points to the base \n"; baseptr = &bb; baseptr -< display(); //calls base function display() baseptr -< show(); //calls base function show() cout >>"\n\nbaseptr points to the derived \n"; baseptr = ⅆ baseptr -< display(); //calls derived function display() baseptr -< show(); //calls derived function show() } The output of this program would be: Baseptr points to base Display base Show base Baseptr points to derived Display derived Show derived Here, we see that the same object pointer points to two different objects of different classes and yet selects the right function to execute. This is implementation of function polymorphism. Remember, however, that runtime polymorphism is achieved only when a virtual function is accessed through a pointer to the base class. It is also interesting to note that since, all the C++ classes are derived from the Object class, a pointer to the Object class can point to any object of any class in C++.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 226/304 | SUBMITTED TEXT | 44 WORDS | 46% | MATCHING TEXT | 44 WORDS |
|---|---|---|---|---|---|

a function with same name. For instance, let us consider the following code snippet. Class aa { Int x; Public: Void display() {.....} //display in base class }; Class bb : public aa { Int

SA ECAP 444.docx (D142426097)

| 227/304 | SUBMITTED TEXT | 92 WORDS | 50% | MATCHING TEXT | 92 WORDS |
|---|---|---|---|---|---|

and &. The & is a unary operator that returns the memory address of its operand. (Remember, a unary operator only requires one operand.) For example, m = &count; place into m the memory address of the variable count. This address is the computer's internal location of the variable. It has nothing to do with the value of count. You can think of & as returning "the address of." Therefore, the preceding assignment statement means "m receives the address of count". To understand the above assignment better, assume that the variable

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 228/304 | SUBMITTED TEXT | 50 WORDS | 52% | MATCHING TEXT | 50 WORDS |
|---|---|---|---|---|---|

the value 2000. The second pointer operator, *, is the complement of &. It is a unary operator that returns the value located at the address that follows. For example, if m contains the memory address of the variable count, q = *m; places the value of count into

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 229/304 | SUBMITTED TEXT | 23 WORDS | 82% | MATCHING TEXT | 23 WORDS |
|---|---|---|---|---|---|

Both & and * have a higher precedence than all other arithmetic operators except the unary minus, with which they are equal.

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 230/304 | SUBMITTED TEXT | 28 WORDS | 98% | MATCHING TEXT | 28 WORDS |
|---|---|---|---|---|---|

your pointer variables always point to the correct type of data. For example, when you declare a pointer to be of type int, the compiler assumes that

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 231/304 | SUBMITTED TEXT | 27 WORDS | 71% | MATCHING TEXT | 27 WORDS |
|---|---|---|---|---|---|

object: #include >iostream< #include >string< using namespace std; class student { private: int rollno; string name; public:

SA ECAP 444.docx (D142426097)

| 232/304 | SUBMITTED TEXT | 22 WORDS | 50% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

a pointer to the object that invoked it. Remember that the this pointer is automatically passed to all member functions. Therefore,

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 233/304 | SUBMITTED TEXT | 124 WORDS | 45% | MATCHING TEXT | 124 WORDS |
|---|---|---|---|---|---|

include >iostream.h< class Base { public: void display() { cout >>"\n Display base "; } virtual void show() { cout >>"\n show base"; } }; Class derived : Public Base { public: void display() { cout >>"\n Display derived"; } void show() { cout >>"\n show derived"; } }; main() 166 Object Oriented Programmimg with C++ Notes { Base B; Derived D; // Declarations Base *bptr; cout >>"\n bptr points to Base\n"; bptr = &B; bptr -<display(); // calls Base version bptr -<show(); // calls Base version cout >>"\n\n bptr points to Derived \n"; bptr = &D; bptr -<display(); // calls

include >iostream.h< class base { public: void display() { cout>>"\n print base"; } virtual void show() //virtual function { cout>>"\n show base"; } }; class derived : public base { 226 LOVELY PROFESSIONAL UNIVERSITY Object-oriented Notes public: display() { cout>>"\n derived"; } void show() { cout>>"\n show derived"; } }; base bb; derived dd; base * cout >>"\ nbaseptr points to the base \n"; baseptr = &bb; baseptr -< display(); //calls base function display() baseptr -< show(); //calls base function show() cout >>"\n\ nbaseptr points to the derived \n"; baseptr = ⅆ baseptr -< display(); //calls

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 234/304 | SUBMITTED TEXT | 33 WORDS | 60% | MATCHING TEXT | 33 WORDS |
|---|---|---|---|---|---|

include >iostream.h< class base // base class { public: virtual void show() // virtual function { cout >>"\n Base "; } }; class Derv1 : Public Base //

include >iostream.h< class base { public: void display() { cout>>"\print base"; } virtual void show() //virtual function { cout>>"\n show base"; } }; class derived : public base { 226

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 235/304 | SUBMITTED TEXT | 32 WORDS | 88% | MATCHING TEXT | 32 WORDS |
|---|---|---|---|---|---|

class BaseClass { int x; public: void setx(int i) { x = i; } int getx() { return x; } };

SA 137E1240-Object Oriented Programming using C++_120E1240.docx (D165245896)

| 236/304 | SUBMITTED TEXT | 27 WORDS | 82% | MATCHING TEXT | 27 WORDS |
|---|---|---|---|---|---|

Virtual Base Class Assume you have two derived classes B and C that have a common base class A, and you additionally have another class D

Virtual Base Classes Suppose you have two derived classes B and C that have a common base class A , and you also have another class D

W https://www.vidyarthiplus.com/vp/attachment.php?aid=46806

| 237/304 | SUBMITTED TEXT | 163 WORDS | 88% | MATCHING TEXT | 163 WORDS |
|---|---|---|---|---|---|

from B and C. You can declare the base class A as virtual to guarantee that B and C share the same subobject of A. In the accompanying illustration, an object of class D has two particular subobjects of class L, one through class B1 and another through class B2. You can use the keyword virtual before the base class specifiers in the base lists of classes B1 and B2 to show that one and only subobject of type L, shared by class B1 and class B2, exists. For example: virt1 class L { /* ... */ }; // indirect base class class B1 : virtual public L { /* ... */ }; class B2 : virtual public L { /* ... */ }; class D : public B1, public B2 { /* ... */ }; // valid Using the keyword virtual in this example ensures that an object of class D inherits only one subobject of class L. 7.13

from B and C . You can declare the base class A as virtual to ensure that B and C share the same subobject of A . In the following example, an object of class D has two distinct subobjects of class L , one through class B1 and another through class B2 . You can use the keyword virtual in front of the base class specifiers in the base lists of classes B1 and B2 to indicate that only one subobject of type L , shared by class B1 and class B2 , exists. For example: class L { /* ... */ }; // indirect base class class B1 : virtual public L { /* ... */ }; class B2 : virtual public L { /* ... */ }; class D : public B1, public B2 { /* ... */ }; // valid Using the keyword virtual in this example ensures that an object of class D inherits only one subobject of class L .

W https://www.vidyarthiplus.com/vp/attachment.php?aid=46806

| 238/304 | SUBMITTED TEXT | 56 WORDS | 100% | MATCHING TEXT | 56 WORDS |
|---|---|---|---|---|---|

private: double length; // Length of a box double breadth; // Breadth of a box double height; // Height of a box };

private: double length; // Length of a box double breadth; // Breadth of a box double height; // Height of a box }; //

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 239/304 | SUBMITTED TEXT | 39 WORDS | 52% | MATCHING TEXT | 39 WORDS |
|---|---|---|---|---|---|

public: void show() { cout >>"\n Derv1"; } }; class Derv2 : Public Base // derived class2 { public: void show() { cout >>"\n Derv2";} }; void main() { Derv1 dv1; //

SA 010E2340-Programming in C and C++.pdf (D165445451)

| 240/304 | SUBMITTED TEXT | 104 WORDS | 51% | MATCHING TEXT | 104 WORDS |
|---|---|---|---|---|---|

int h) { height = h; } protected: int width; int height; }; // Derived classes class Rectangle: public Shape { public: int getArea() { return (width * height); } }; class Triangle: public Shape { public: int getArea() { return (width * height)/2; } }; int main(

SA OOP through C++ (Block 2).pdf (D148964031)

| 241/304 | SUBMITTED TEXT | 20 WORDS | 76% | MATCHING TEXT | 20 WORDS |
|---|---|---|---|---|---|

base class constructor does not take any arguments, the derived class need not have a constructor function. However, if

base class constructor any arguments, the derived class need not have a constructor function. However, if

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 242/304 | SUBMITTED TEXT | 24 WORDS | 100% | MATCHING TEXT | 24 WORDS |
|---|---|---|---|---|---|

endl; return 0; } When the above code is compiled and executed, it produces the following result: Total

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 243/304 | SUBMITTED TEXT | 35 WORDS | 64% | MATCHING TEXT | 35 WORDS |
|---|---|---|---|---|---|

in c++? a) __cdecl b) __stdcall c) __pascal d) __fastcall 4. What is the output of this program? #include >iostream< using namespace std; int add(int first, int

SA ECAP 444.docx (D142426097)

| 244/304 | SUBMITTED TEXT | 26 WORDS | 100% | MATCHING TEXT | 26 WORDS |
|---|---|---|---|---|---|

a) 25 b) 35 c) 40 d) 45 5. What is the output of this program? #include >iostream<

SA ECAP 444.docx (D142426097)

| 245/304 | SUBMITTED TEXT | 28 WORDS | 100% | MATCHING TEXT | 28 WORDS |
|---|---|---|---|---|---|

a) 2 b) 20 c) 21 d) 22 6. What is the output of this program? #include >iostream< using namespace std;

SA ECAP 444.docx (D142426097)

| 246/304 | SUBMITTED TEXT | 19 WORDS | 90% | MATCHING TEXT | 19 WORDS |
|---|---|---|---|---|---|

What is the output of this program? #include >iostream< using namespace std; int func (int

SA ECAP 444.docx (D142426097)

| 247/304 | SUBMITTED TEXT | 28 WORDS | 67% | MATCHING TEXT | 28 WORDS |
|---|---|---|---|---|---|

Object Oriented Programming With C++ Tata McGraw-Hill Education. ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India. 178

SA Object Oriented Programming through C++ Block 1.pdf (D164970258)

| 248/304 | SUBMITTED TEXT | 24 WORDS | 100% | MATCHING TEXT | 24 WORDS |
|---|---|---|---|---|---|

The different aspects of C++’s I/O system, such as console I/O and disk I/O, are actually just different perspectives on the same mechanism.

The different aspects of C++'s I/O system, such as console I/O and disk I/O, are actually just different perspectives on the same mechanism.

W https://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf

| 249/304 | SUBMITTED TEXT | 16 WORDS | 88% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

Further Readings Objectives After studying this unit, you should be able to: ? Understand

SA ECAP 444.docx (D142426097)

| 250/304 | SUBMITTED TEXT | 155 WORDS | 95% | MATCHING TEXT | 155 WORDS |
|---|---|---|---|---|---|

C++ Stream There are currently two versions of the C++ object-oriented I/O library in use: the older one that is based upon the original specifications for C++ and the newer one defined by Standard C++. The old I/O library is supported by the header file >iostream.h<. The new I/O library is supported by the header >iostream<. For the most part the two libraries appear the same, because the new I/O library is simply an updated and improved version of the old one. In fact, the vast majority of differences between the two occur beneath the surface, in the way that the libraries are implemented—not in how they are used. From the programmer’s perspective, there are two main differences between the old and new C++/O libraries. First, the new I/O library contains a few additional features and defines some new data types. Thus, the new I/O library is essentially a superset of

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 251/304 | SUBMITTED TEXT | 232 WORDS | 97% | MATCHING TEXT | 232 WORDS |
|---|---|---|---|---|---|

Streams classes A stream is a source of sequence of bytes. A stream abstracts for input/output devices. It can be tied up with any I/O device and I/O can be performed in a uniform way. The C++ iostream library is an object-oriented implementation of this abstraction. It has a source (producer) of flow of bytes and a sink (consumer) of the bytes. The required classes for the stream I/O are defined in different library header files. To use the I/O streams in a C++ program, one must include iostream.h header file in the program. This file defines the required classes and provides the buffering. Instead of functions, the library provides operators to carry out the I/O. Two of the Stream Operators are: >> : Stream insertion for output. << : Stream extraction for input. The following streams are created and opened automatically: cin : Standard console input (keyboard). cout : Standard console output (screen). cprn : Standard printer (LPT1). cerr : Standard error output (screen). clog : Standard log (screen). caux : Standard auxiliary (screen). Example: The following program reads an integer and prints the input on the console. #include >iostream< // Header for stream I/O. int main(void) { int p; // variable to hold the input integer cout >> "Enter an integer: "; cin << p; cout >> "\n You have entered" >> p; } 8.4

Streams A stream is a source of sequence of bytes. A stream abstracts for input/output devices. It can be tied up with any I/O device and I/O can be performed in a uniform way. The C++ iostream library is an object-oriented implementation of this abstraction. It has a source (producer) of flow of bytes and a sink (consumer) of the bytes. The required classes for the stream I/O are defined in different library header files. To use the I/O streams in a C++ program, one must include iostream.h header file in the program. This file defines the required classes and provides the buffering. Instead of functions, the library provides operators to carry out the I/O. Two of the Stream Operators are: >> : Stream insertion for output. << : Stream extraction for input. The following streams are created and opened automatically: cin : Standard console input (keyboard). cout : Standard console output (screen). cprn : Standard printer (LPT1). cerr : Standard error output (screen). clog : Standard log (screen). caux : Standard auxiliary (screen). The following program reads an integer and prints the input on the console. #include >iostream< // Header for stream I/O. int main(void) { int p; // variable to hold the input integer cout >> “Enter an integer: “; cin << p; cout >> “\n You have entered “ >> p; } 12.2

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 252/304 | SUBMITTED TEXT | 13 WORDS | 100% | MATCHING TEXT | 13 WORDS |
|---|---|---|---|---|---|

Unformatted I/O operations Unformatted Input/Output is the most basic form of input/output.

Unformatted I/O Operations Unformatted Input/Output is the most basic form of input/output.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 253/304 | SUBMITTED TEXT | 18 WORDS | 100% | MATCHING TEXT | 18 WORDS |
|---|---|---|---|---|---|

Unformatted input/output transfers the internal binary representation of the data directly between memory and the file.

Unformatted input/output transfers the internal binary representation of the data directly between memory and the file.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 254/304 | SUBMITTED TEXT | 51 WORDS | 92% | MATCHING TEXT | 51 WORDS |
|---|---|---|---|---|---|

Formatted output converts the internal binary representation of the data to ASCII characters which are written to the output file. Formatted input reads characters from the input file and converts them to internal form. Formatted I/O can be either "Free" format or "Explicit" format. 180

Formatted output converts the internal binary representation of the data to ASCII characters which are written to the output file. Formatted input reads characters from the input file and LOVELY PROFESSIONAL UNIVERSITY 263 Unit 12: Console I/O Notes converts them to internal form. Formatted I/O can be either “Free” format or “Explicit” format,

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 255/304 | SUBMITTED TEXT | 44 WORDS | 100% | MATCHING TEXT | 44 WORDS |
|---|---|---|---|---|---|

the old one. Nearly all programs originally written for the old library will compile without substantive changes when the new library is used. Second, the old-style I/O library was in the global namespace. The new-style library is in the std namespace. 8.3

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 256/304 | SUBMITTED TEXT | 65 WORDS | 100% | MATCHING TEXT | 65 WORDS |
|---|---|---|---|---|---|

Advantages and Disadvantages of Unformatted I/O ? Unformatted input/output is the simplest and most efficient form of input/output. ? It is usually the most compact way to store data. ? Unformatted input/output is the least portable form of input/output. ? Unformatted data files can only be moved easily to and from computers that share the same internal data representation. ?

Advantages and Disadvantages of Unformatted I/O Unformatted input/output is the simplest and most efficient form of input/output. It is usually the most compact way to store data. Unformatted input/output is the least portable form of input/output. Unformatted data files can only be moved easily to and from computers that share the same internal data representation.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 257/304 | SUBMITTED TEXT | 53 WORDS | 100% | MATCHING TEXT | 53 WORDS |
|---|---|---|---|---|---|

Unformatted input/output is not directly human readable, so you cannot type it out on a terminal screen or edit it with a text editor. Advantages and Disadvantages of Formatted I/O ? Formatted input/output is very portable. ? It is a simple process to move formatted data files to various computers. ?

Unformatted input/output is not directly human readable, so you cannot type it out on a terminal screen or edit it with a text editor. Advantages and Disadvantages of Formatted I/O Formatted input/output is very portable. It is a simple process to move formatted data files to various computers,

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 258/304 | SUBMITTED TEXT | 20 WORDS | 100% | MATCHING TEXT | 20 WORDS |
|---|---|---|---|---|---|

Formatted files are human readable and can be typed to the terminal screen or edited with a text editor.

Formatted files are human readable and can be typed to the terminal screen or edited with a text editor.

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 259/304 | SUBMITTED TEXT | 27 WORDS | 95% | MATCHING TEXT | 27 WORDS |
|---|---|---|---|---|---|

void open (const char* filename, ios::openmode mode = ios::in | ios::out); //

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 260/304 | SUBMITTED TEXT | 21 WORDS | 76% | MATCHING TEXT | 21 WORDS |
|---|---|---|---|---|---|

include >iostream< 3. #include >fstream< 4. #include >cstdlib< 5. #include >string< 6. using namespace std; 7. int main() { 8.

include >iostream< #include >istream< #include >sstream< #include >string< using namespace std; int main() {

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 261/304 | SUBMITTED TEXT | 21 WORDS | 76% | MATCHING TEXT | 21 WORDS |
|---|---|---|---|---|---|

include >iostream< 3. #include >fstream< 4. #include >cstdlib< 5. #include >string< 6. using namespace std; 7. int main() { 8.

include >iostream< #include >istream< #include >sstream< #include >string< using namespace std; int main() {

W https://mu.ac.in/wp-content/uploads/2020/12/Object-Oriented-Programming-F.Y.-MCA-Semester-I.pdf

| 262/304 | SUBMITTED TEXT | 14 WORDS | 100% | MATCHING TEXT | 14 WORDS |
|---|---|---|---|---|---|

the number of digits printed to the right of the decimal point. This

the number of digits printed to the right of the decimal point. This

W https://www.vidyarthiplus.com/vp/attachment.php?aid=46806

| 263/304 | SUBMITTED TEXT | 25 WORDS | 45% | MATCHING TEXT | 25 WORDS |
|---|---|---|---|---|---|

ios::in - open file for input operation ? ios::out - open file for output operation ? ios::app - output appends at the end of

SA 010E2340-Programming in C and C++.pdf (D165445451)

| 264/304 | SUBMITTED TEXT | 24 WORDS | 100% | MATCHING TEXT | 24 WORDS |
|---|---|---|---|---|---|

d) None of the mentioned 4. What is the output of this program? #include >iostream< using namespace std;

SA ECAP 444.docx (D142426097)

| 265/304 | SUBMITTED TEXT | 83 WORDS | 44% | MATCHING TEXT | 83 WORDS |
|---|---|---|---|---|---|

Object Oriented Programmimg with C++ Notes c) 65 d) 65 5. What is the output of this program? #include >iostream< using namespace std; int main () { int n; n = 43; cout >> hex >>n >> endl; return 0; } a) 2c b) 2b c) 20 d) 50 6. What is the output of this program

SA DECAP444_OBJECT_ORIENTED_PROGRAMMING_USING_C++.pdf (D142327140)

| 266/304 | SUBMITTED TEXT | 62 WORDS | 86% | MATCHING TEXT | 62 WORDS |
|---|---|---|---|---|---|

include >fstream< using namespace std; int main () { long pos; ofstream outfile; outfile.open ("test.txt"); outfile.write ("This is an apple",16); pos = outfile.tellp(); outfile.seekp (pos - 7); outfile.write (" sam", 4); outfile.close(); return 0; }

SA OOP through C++ (Block 2).pdf (D148964031)

| 267/304 | SUBMITTED TEXT | 103 WORDS | 52% | MATCHING TEXT | 103 WORDS |
|---|---|---|---|---|---|

What is the output of this program? #include >iostream< using namespace std; Console I/O Operations, Files 201 Notes int main () { int n; n = -77; cout.width(4); cout >> internal >> n >> endl; return 0; } a) 77 b) -77 c) – 77 d) None of the mentioned 8. What is the output of this program? #include >iostream< #include >locale< using namespace std;

SA ECAP 444.docx (D142426097)

| 268/304 | SUBMITTED TEXT | 29 WORDS | 100% | MATCHING TEXT | 29 WORDS |
|---|---|---|---|---|---|

cin: Standard console input (keyboard). ? cout: Standard console output (screen). ? cprn: Standard printer (LPT1). ? cerr : Standard error output (screen). ? clog: Standard log (screen).

cin : Standard console input (keyboard). cout : Standard console output (screen). cprn : Standard printer (LPT1). cerr : Standard error output (screen). clog : Standard log (screen).

W http://ebooks.lpude.in/computer_application/bca/term_2/DCAP107_DCAP404_OBJECT_ORIENTED_PROGRAMMIN ...

| 269/304 | SUBMITTED TEXT | 25 WORDS | 90% | MATCHING TEXT | 25 WORDS |
|---|---|---|---|---|---|

endl; return 0; } a) 3.14 b) 3.14159 c) Error d) None of the mentioned 9.

SA ECAP 444.docx (D142426097)

| 270/304 | SUBMITTED TEXT | 27 WORDS | 67% | MATCHING TEXT | 27 WORDS |
|---|---|---|---|---|---|

Object Oriented Programming With C++ Tata McGraw-Hill Education. ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India.

SA Object Oriented Programming through C++ Block 1.pdf (D164970258)

| 271/304 | SUBMITTED TEXT | 16 WORDS | 88% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

Further Readings Objectives After studying this unit, you should be able to: ? Understand

SA ECAP 444.docx (D142426097)

| 272/304 | SUBMITTED TEXT | 53 WORDS | 97% | MATCHING TEXT | 53 WORDS |
|---|---|---|---|---|---|

we have been using the iostream standard library, which provides cinand cout methods for reading from standard input and writing to standard output respectively. This chapter will teach you how to read and write from a file. This requires another standard C++ library called fstream, which defines three new data types 9.2

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 273/304 | SUBMITTED TEXT | 35 WORDS | 90% | MATCHING TEXT | 35 WORDS |
|---|---|---|---|---|---|

represents the output file stream and is used to create files, and write information to files. ? Ifstream: It represents the input file stream and is used to read information from files. ? Fstream:

SA OOP through C++ (Block 2).pdf (D148964031)

| 274/304 | SUBMITTED TEXT | 39 WORDS | 89% | MATCHING TEXT | 39 WORDS |
|---|---|---|---|---|---|

represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files. 204 Object Oriented Programmimg with C++

SA OOP through C++ (Block 2).pdf (D148964031)

| 275/304 | SUBMITTED TEXT | 20 WORDS | 100% | MATCHING TEXT | 20 WORDS |
|---|---|---|---|---|---|

Opening a File A file must be opened before you can read from it or write to it. Either

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 276/304 | SUBMITTED TEXT | 31 WORDS | 93% | MATCHING TEXT | 31 WORDS |
|---|---|---|---|---|---|

ofstream or fstream object may be used to open a file for writing and ifstream object is used to open a file for reading purpose only. Syntax for open() function:

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 277/304 | SUBMITTED TEXT | 33 WORDS | 96% | MATCHING TEXT | 33 WORDS |
|---|---|---|---|---|---|

First argument specifies the name and location of the file to be opened. ? Second argument of the open() member function defines the mode in which the file should be opened. 9.4

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 278/304 | SUBMITTED TEXT | 30 WORDS | 91% | MATCHING TEXT | 30 WORDS |
|---|---|---|---|---|---|

Closing a File When a C++ program terminates it automatically: ? Closes all the streams ? Release all the allocated memory ? Close all the opened files.

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 279/304 | SUBMITTED TEXT | 23 WORDS | 59% | MATCHING TEXT | 23 WORDS |
|---|---|---|---|---|---|

and zero (which means FALSE) generally. Rules for using end-of-document (eof( )): ? Continuously test for the end-of-document condition before

SA INF_1016.pdf (D164968061)

| 280/304 | SUBMITTED TEXT | 13 WORDS | 96% | MATCHING TEXT | 13 WORDS |
|---|---|---|---|---|---|

ios::app All output to that file to be appended to the end.

SA ECAP 444.docx (D142426097)

| 281/304 | SUBMITTED TEXT | 13 WORDS | 100% | MATCHING TEXT | 13 WORDS |
|---|---|---|---|---|---|

ios::in Open a file for reading. ios::out Open a file for writing.
ios::

SA 120E1240_ Object Oriented Programming Using C++.doc (D165245825)

| 282/304 | SUBMITTED TEXT | 11 WORDS | 100% | MATCHING TEXT | 11 WORDS |
|---|---|---|---|---|---|

If you want to open a file in write mode

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 283/304 | SUBMITTED TEXT | 8 WORDS | 100% | MATCHING TEXT | 8 WORDS |
|---|---|---|---|---|---|

ofstream outfile; outfile.open("file.dat", ios::out | ios::trunc );

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 284/304 | SUBMITTED TEXT | 19 WORDS | 100% | MATCHING TEXT | 19 WORDS |
|---|---|---|---|---|---|

open a file for reading and writing purpose as follows: fstream
afile; afile.open("file.dat", ios::out | ios::in ); 9.7

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 285/304 | SUBMITTED TEXT | 119 WORDS | 97% | MATCHING TEXT | 119 WORDS |
|---|---|---|---|---|---|

Both istream and ostream give member functions for repositioning the file position pointer. These member functions are seekg ("seek get") for istream and seekp ("seek put") for ostream. The argument to seekg and seekp normally is a long integer. A second argument can be specified to indicate the seek direction. The seek direction can be ios::beg (the default) for positioning relative to the beginning of a stream,ios::cur for positioning relative to the current position in a stream or ios::endfor positioning relative to the end of a stream. The file-position pointer is an integer value that specifies the location in the file as a number of bytes from the file's starting location. Some examples of positioning the "get"

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 286/304 | SUBMITTED TEXT | 45 WORDS | 92% | MATCHING TEXT | 45 WORDS |
|---|---|---|---|---|---|

ios::beg) fileObject.seekg( n ); /position n bytes forward in fileObject fileObject.seekg( n, ios::cur ); /position n bytes once again from end of fileObject fileObject.seekg( n, ios::end ); /position at end of fileObject fileObject.seekg( 0, ios::end ); 9.8

SA 248E1110-Object Oriented Programing using C++(Id 2732).doc (D165248029)

| 287/304 | SUBMITTED TEXT | 21 WORDS | 95% | MATCHING TEXT | 21 WORDS |
|---|---|---|---|---|---|

The file stream classes support a number of member functions for performing the input and output operations on files. Functions

SA OOP through C++ (Block 2).pdf (D148964031)

| 288/304 | SUBMITTED TEXT | 45 WORDS | 100% | MATCHING TEXT | 45 WORDS |
|---|---|---|---|---|---|

get() and put() are capable of handling a single character at a time. The function getline() lets you handle multiple characters at a time. Another pair of functions i.e., read() and write() are capable of reading and writing blocks of binary data. The get(),

SA ODL Learning Materials (ALL 5 UNITS).pdf (D109014230)

| 289/304 | SUBMITTED TEXT | 54 WORDS | 93% | MATCHING TEXT | 54 WORDS |
|---|---|---|---|---|---|

The get() has many forms, but the most commonly used version is shown here, along with put() : istream & get(char & ch) ; ostream & put(char ch) ; The get() function reads a single character from the associated stream and puts that value in ch. It returns a reference to the stream.

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 290/304 | SUBMITTED TEXT | 12 WORDS | 100% | MATCHING TEXT | 12 WORDS |
|---|---|---|---|---|---|

ch to the stream and returns a reference to the stream.

SA C++ From The Ground Up_ 3rd Edition (2003).pdf (D1118784)

| 291/304 | SUBMITTED TEXT | 79 WORDS | 80% | MATCHING TEXT | 79 WORDS |
|---|---|---|---|---|---|

include>iostream.h< #include>stdlib.h< #include>fstream.h< #include>conio.h< void main() { burn fname[20], ch; ifstream blade;/make an info stream clrscr(); cout>>"Enter the name of the record: "; cin.get(fname, 20); cin.get(ch); fin.open(fname, ios::in);/open record if(!fin)/if blade stores zero i.e., false esteem { cout>>"Error happened in opening the file..!!\n"; cout>>"Press any key to exit...\n"; getch(); exit(1); } while(fin)/balance will be 0 when eof is

SA odl C++ lecture notes unit-5.docx (D109013221)

| 292/304 | SUBMITTED TEXT | 22 WORDS | 92% | MATCHING TEXT | 22 WORDS |
|---|---|---|---|---|---|

fin.get(ch);/read a character cout>>ch;/show the character } cout>>"\nPress any key to exit...\n"; fin.close(); getch(); }

SA odl C++ lecture notes unit-5.docx (D109013221)

| 293/304 | SUBMITTED TEXT | 93 WORDS | 100% | MATCHING TEXT | 93 WORDS |
|---|---|---|---|---|---|

These classes are derived directly or indirectly from the classes istream and ostream . We have already used objects whose types were these classes: cin is an object of class istream and cout is an object of class ostream . Therefore, we have already been using classes that are related to our file streams. And in fact, we can use our file streams the same way we are already used to use cin and cout , with the only difference that we have to associate these streams with physical files. 9.10

SA 137E1240-Object Oriented Programming using C++_120E1240.docx (D165245896)

| 294/304 | SUBMITTED TEXT | 20 WORDS | 73% | MATCHING TEXT | 20 WORDS |
|---|---|---|---|---|---|

Which of the following is not a file opening mode ____ . (a) ios::ate (b) ios::nocreate (c) ios::noreplace (d) ios::

SA INF_1016.pdf (D164968061)

| 295/304 | SUBMITTED TEXT | 17 WORDS | 100% | MATCHING TEXT | 17 WORDS |
|---|---|---|---|---|---|

that will be performing both input and output operations must be declared as class _____ . (

SA OOP through C++ (Block 2).pdf (D148964031)

| 296/304 | SUBMITTED TEXT | 17 WORDS | 78% | MATCHING TEXT | 17 WORDS |
|---|---|---|---|---|---|

To create an output stream, we must declare the stream to be of class _____ . (

SA OOP through C++ (Block 2).pdf (D148964031)

| 297/304 | SUBMITTED TEXT | 16 WORDS | 71% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

By default, all the files are opened in _____mode . (a) Binary (b) Text (c)

SA ECAP 444.docx (D142426097)

| 298/304 | SUBMITTED TEXT | 14 WORDS | 60% | MATCHING TEXT | 14 WORDS |
|---|---|---|---|---|---|

is the default mode of opening the file? (a) ios::in|ios::out (b) ios::in|ios::out|ios::trunc (c) ios::in|ios::

SA ECAP 444.docx (D142426097)

| 299/304 | SUBMITTED TEXT | 17 WORDS | 89% | MATCHING TEXT | 17 WORDS |
|---|---|---|---|---|---|

What is a open()? 8. Describe briefly the features of I/O system supported by C++. 9.

What is a stream? 5.2.Describe briefly the features of I/O system supported by C++. 5.3.

W https://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf

| 300/304 | SUBMITTED TEXT | 29 WORDS | 58% | MATCHING TEXT | 29 WORDS |
|---|---|---|---|---|---|

a file pointer? (a) ios::cur (b) ios::set (c) ios::end (d) ios::beg 10. It is not possible to combine two or more file opening mode in open () method. (

SA INF_1016.pdf (D164968061)

| 301/304 | SUBMITTED TEXT | 36 WORDS | 92% | MATCHING TEXT | 36 WORDS |
|---|---|---|---|---|---|

represents the output file stream and is used to create files and to write information to files. ? Ifstream: It represents the input file stream and is used to read information from files. ? Fstream:

SA OOP through C++ (Block 2).pdf (D148964031)

| 302/304 | SUBMITTED TEXT | 31 WORDS | 98% | MATCHING TEXT | 31 WORDS |
|---|---|---|---|---|---|

represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files. ?

SA OOP through C++ (Block 2).pdf (D148964031)

| 303/304 | SUBMITTED TEXT | 16 WORDS | 96% | MATCHING TEXT | 16 WORDS |
|---|---|---|---|---|---|

ios::app: All output to that file to be appended to the end.

SA ECAP 444.docx (D142426097)

| 304/304 | SUBMITTED TEXT | 21 WORDS | 67% | MATCHING TEXT | 21 WORDS |
|---|---|---|---|---|---|

Object Oriented Programming With C++ Tata McGraw-Hill Education. ? Subhash, K. U. (2010) Object Oriented Programming With C++ Pearson Education India.

SA Object Oriented Programming through C++ Block 1.pdf (D164970258)