

SIX WEEKS SUMMER TRAINING REPORT

On

Ebox C++ Object oriented programming

**From 24/04/21 to 14/07/21**

**SUBMITTED BY**

**Name of student:**

**Bhavesh Singh**

**Registration Number:**

**11902409** **Program Name:**

B.Tech School of Computer Science and Engineering, Lovely Professional University, Phagwara.

# DECLARATION

I hereby declare that I have completed my six weeks summer training at Ebox from 24th May 2021 to 14th

July 2021 under the guidance of Induja. I have declared that I have worked with full dedication during these six weeks of training and my learning outcomes fulfill the requirements of training for the award of degree of

Bachelor of Technology (Computer Science and Engineering), Lovely Professional University, Phagwara.

Name of Student: Bhavesh Singh

Registration No:

11902409

Date:

15th July 2021

# Acknowledgements

The successful completion of this summer training was not mere sincerity, hard work and continuous strive for innovation and discoveries. I owe this summer training to many helpful, supportive and kind people who had been so kind in supporting and guiding in every milestone of this training. It gives me immense pleasure to express my sincere gratitude and hearty acknowledgements to all those who unconditionally helped me throughout my journey of project. I am very thankful to the Lovely

Professional University for providing this opportunity. It is my utmost joy that from the beginning to end and in every course of the training, my supervisor Miss. Induja had been so supportive and

helpful. I am very honored and privileged to have mam beside me all time for continuous guidance,

inspiration and encouragement. I would take this opportunity to thank sir for his unwavering support

and everything. I am also very grateful to every helpful and supportive people including my friends who helped me a lot when I got stuck while solving difficult object oriented problems

# Summer Training Certificate



**![Text

Description automatically generated with medium confidence]()**



**CONTENTS**

Chapter 01: Object Oriented Programming,Basic Concept

Chapter 02: BENEFITS OF OOP, Basics of C++

Chapter 03: Tokens, Basic Data types in C++

Chapter 04: Symbolic Constant, Operators

Chapter 05: Control Structures, Functions in C++

Chapter 06: Function Overloading, Class

Chapter 07: Member Function,Nesting of Member function

**Object Oriented Programming**

## Module I

Introduction to object oriented programming, user defined types, structures, unions, polymorphism, encapsulation. Getting started with C++ syntax, data-type, variables, strings, functions, default values in functions, recursion, namespaces, operators, flow control, arrays and pointers.

## Module II

Abstraction mechanism: Classes, private, public, constructors, destructors, member data, member functions, inline function, friend functions, static members, and references. Inheritance: Class hierarchy, derived classes, single inheritance, multiple, multilevel, hybrid inheritance, role of virtual base class, constructor and destructor execution, base initialization using derived class constructors.

Polymorphism: Binding, Static binding, Dynamic binding, Static polymorphism: Function Overloading, Ambiguity in function overloading, Dynamic polymorphism: Base class pointer, object slicing, late binding, method overriding with virtual functions, pure virtual functions, abstract classes.

Operator Overloading: This pointer, applications of this pointer, Operator function, member and non member operator function, operator overloading, I/O operators. Exception handling: Try, throw, and catch, exceptions and derived classes, function exception declaration, unexpected exceptions, exception when handling exceptions, resource capture and release.

**Module-1:**

**Chapter -1**

**Introduction:**

Programmers write instructions in various programming languages to perform their computation tasks such as:

1. Machine level Language
2. Assembly level Language
3. High level Language

**Machine level Language :**

Machine code or machine language is a set of instructions executed directly by a computer's central processing unit (CPU). Each instruction performs a very specific task, such as a load, a jump, or an ALU operation on a unit of data in a CPU register or memory. Every program directly executed by a CPU is made up of a series of such instructions.

**Assembly level Language :**

An assembly language (or assembler language) is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions. Assembly language is converted into executable machine code by a utility program referred to as an assembler; the conversion process is referred to as assembly, or assembling the code.

**High level Language :**

High-level language is any programming language that enables development of a program in much simpler programming context and is generally independent of the computer's hardware architecture. High-level language has a higher level of abstraction from the computer, and focuses more on the programming logic rather than the underlying hardware components such as memory addressing and register utilization.

The first high-level programming languages were designed in the 1950s. Now there are dozens of different languages, including Ada , Algol, BASIC, COBOL, C, C++, JAVA, FORTRAN, LISP, Pascal, and Prolog. Such languages are considered high-level because they are closer to human languages and farther from machine languages.

## Procedure Oriented Programming Language

In the procedure oriented approach, the problem is viewed as sequence of things to be done such as reading , calculation and printing.

Procedure oriented programming basically consist of writing a list of instruction or actions for the computer to follow and organizing these instruction into groups known as functions.

Function

-

1

Function

-

3

Function

-

2

Main

program



The disadvantage of the procedure oriented programming languages is:

1. Global data access
2. It does not model real word problem very well

Function

-

1

Global

data

Function

-

3

Global

data

3.

No

data

hiding

Function

-

2

Local

data

Local

data

Local

data

Characteristics of procedure oriented programming:

1. Emphasis is on doing things(algorithm)
2. Large programs are divided into smaller programs known as functions.
3. Most of the functions share global data
4. Data move openly around the system from function to function

**Object Oriented Programing**

“Object oriented programming as an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand”.

Communication

**Object**

**C**

Data

Functions

Functions

Data

Functions

Data

**Object**

**A**

**Object**

**B**

Features of the Object Oriented programming

* 1. Emphasis is on doing rather than procedure.
  2. programs are divided into what are known as objects.
  3. Data structures are designed such that they characterize the objects.
  4. Functions that operate on the data of an object are tied together in the data structure.
  5. Data is hidden and can’t be accessed by external functions.
  6. Objects may communicate with each other through functions.
  7. New data and functions can be easily added.
  8. Follows bottom-up approach in program design.

**BASIC CONCEPTS OF OBJECTS ORIENTED PROGRAMMING**

1. Objects
2. Classes
3. Data abstraction and encapsulation
4. Inheritance
5. Polymorphism
6. Dynamic binding
7. Message passing

**OBJECTS**

Objects are the basic run-time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program must handle.

The fundamental idea behind object oriented approach is to combine both data and function into a single unit and these units are called objects.

The term objects means a combination of data and program that represent some real word entity. For example: consider an example named Amit; Amit is 25 years old and his salary is 2500. The Amit may be represented in a computer program as an object. The data part of the object would be (name: Amit, age: 25, salary: 2500)

The program part of the object may be collection of programs (retrive of data, change age, change of salary). In general even any user –defined type-such as employee may be used. In the Amit object the name, age and salary are called attributes of the object.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Object: Student |  | | STUDENT | |
| DATA Name  Date-of-birth  Marks |  | T | otal |  |
|  |  |
| A | erage |
| FUNCTIONS  Total  Average  Display |  |  |
| D | isplay10 |
|  |  |

**DATA ABSTRACTION :**

Abstraction refers to the act of representing essential features without including the back ground details or explanations. Classes use the concept of abstraction and are defined as size, width and cost and functions to operate on the attributes.

**DATA ENCAPSALATION :**

The wrapping up of data and function into a single unit (called class) is known as encapsulation. The data is not accessible to the outside world and only those functions which are wrapped in the class can access it. These functions provide the interface between the objects data and the program.

**INHERITENCE :**

Inheritance is the process by which objects of one class acquire the properties of another class. In the concept of inheritance provides the idea of reusablity. This mean that we can add additional features to an existing class with out modifying it. This is possible by desining a new class will have the combined features of both the classes.

**POLYMORPHISIM:**

Polymorphism means the ability to take more than one form. An operation may exhibit different instance. The behaviour depends upon the type of data used in the operation.

A language feature that allows a function or operator to be given more than one definition. The types of the arguments with which the function or operator is called determines which definition will be used.

Overloading may be operator overloading or function overloading.

**MESSAGE PASSING :**

An object oriented program consists of a set of objects that communicate with each

other.

A message for an object is a request for execution of a procedure and therefore will

invoke a function (procedure) in the receiving object that generates the desired result. Message passing involves specifying the name of the object, the name of the function (message) and information to be sent.

**Employee . Salary (name)**

**Object Information**

**Message**

### Chapter - 2

**BENEFITS OF OOP:**

Oop offers several benefits to both the program designer and the user. Object-oriented contributes to the solution of many problems associated with the development and quality of software products.

The principal advantages are :

1. Through inheritance we can eliminate redundant code and extend the use of existing classes.
2. We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
3. This principle of data hiding helps the programmer to build secure programs that can’t be invaded by code in other parts of the program.
4. It is possible to have multiple instances of an object to co-exist with out any interference.
5. It is easy to partition the work in a project based on objects.
6. Object-oriented systems can be easily upgraded from small to large systems.
7. Message passing techniques for communication between objects makes the interface description with external systems much simpler.
8. Software complexity can be easily managed.

**APPLICATION OF OOP:**

The most popular application of oops up to now, has been in the area of user interface design such as windows. There are hundreds of windowing systems developed using oop techniques.

Real business systems are often much more complex and contain many more objects with complicated attributes and methods. Oop is useful in this type of applications because it can simplify a complex problem. The promising areas for application of oop includes.

**Basics of C++**

C ++ is an object oriented programming language, C ++ was developed by Jarney Stroustrup at AT & T Bell lab, USA in early eighties. C ++ was developed from c and simula 67 language. C ++ was early called ‘C with classes’.

**C++ Comments:**

C++ introduces a new comment symbol //(double slash). Comments start with a

double slash symbol and terminate at the end of line. A comment may start any where in the line and what ever follows till the end of line is ignored. Note that there is no closing symbol.

The double slash comment is basically a single line comment. Multi line comments can be written as follows:

// this is an example of

// c++ program // thank you

The c comment symbols /\* ….\*/ are still valid and more suitable for multi line comments.

/\* this is an example of c++ program \*/

**Output Operator:**

The statement cout <<”Hello, world” displayed the string with in quotes on the screen. The identifier cout can be used to display individual characters, strings and even numbers. It is a predefined object that corresponds to the standard output stream. Stream just refers to a flow of data and the standard Output stream normally flows to the screen display. The cout object, whose properties are defined in iostream.h represents that stream. The insertion operator << also called the ‘put to’ operator directs the information on its right to the object on its left.

**Cascading Of I/O Operator:**

cout<<”sum=”<<sum<<”\n”;

cout<<”sum=”<<sum<<”\n”<<”average=”<<average<<”\n”; cin>>number1>>number2;

**Structure Of A Program :**

Probably the best way to start learning a programming language is by writing a program. Therefore, here is our first program: // my first program in C++

#include <iostream> using namespace std;

int main ()

{ cout << "Hello World!"; return 0;

}

Output:-Hello World!

The first panel shows the source code for our first program. The second one shows the result of the program once compiled and executed. The way to edit and compile a program depends on the compiler you are using. Depending on whether it has a Development Interface or not and on its version. Consult the compilers section and the manual or help included with your compiler if you have doubts on how to compile a C++ console program.

The previous program is the typical program that programmer apprentices write for the first time, and its result is the printing on screen of the "Hello World!" sentence. It is one of the simplest programs that can be written in C++, but it already contains the fundamental components that every C++ program has. We are going to look line by line at the code we have just written:

**// my first program in C++**

This is a comment line. All lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program. The programmer can use them to include short explanations or observations within the source code itself. In this case, the line is a brief description of what our program is.

**#include <iostream>**

Lines beginning with a hash sign (#) are directives for the preprocessor. They are not regular code lines with expressions but indications for the compiler's preprocessor. In this case the directive #include<iostream> tells the preprocessor to include the iostream standard file. This specific file (iostream) includes the declarations of the basic standard input-output library in C++, and it is included because its functionality is going to be used later in the program.

**using namespace std;**

All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name *std*. So in order to access its functionality we declare with this expression that we will be using these entities. This line is very frequent in C++ programs that use the standard library, and in fact it will be included in most of the source codes included in these tutorials.

**int main ()**

This line corresponds to the beginning of the definition of the main function. The main function is the point by where all C++ programs start their execution, independently of its location within the source code. It does not matter whether there are other functions with other names defined before or after it – the instructions contained within this function's definition will always be the first ones to be

executed in any C++ program. For that same reason, it is essential that all C++ programs have a main function.

The word main is followed in the code by a pair of parentheses (()). That is because it is a function declaration: In C++, what differentiates a function declaration from other types of expressions are these parentheses that follow its name. Optionally, these parentheses may enclose a list of parameters within them.

Right after these parentheses we can find the body of the main function enclosed in braces ({}).

What is contained within these braces is what the function does when it is executed.

**cout << "Hello World!";**

This line is a C++ statement. A statement is a simple or compound expression that can actually produce some effect. In fact, this statement performs the only action that generates a visible effect in our first program.

cout represents the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters (in this case the Hello World sequence of characters) into the standard output stream (which usually is the screen).

cout is declared in the iostream standard file within the std namespace, so that's why we needed to include that specific file and to declare that we were going to use this specific namespace earlier in our code.

Notice that the statement ends with a semicolon character (;). This character is used to mark the end of the statement and in fact it must be included at the end of all expression statements in all C++ programs (one of the most common syntax errors is indeed to forget to include some semicolon after a statement). **return 0;**

The return statement causes the main function to finish. return may be followed by a return code (in our example is followed by the return code 0). A return code of 0 for the main function is generally interpreted as the program worked as expected without any errors during its execution. This is the most usual way to end a C++ console program.

You may have noticed that not all the lines of this program perform actions when the code is executed. There were lines containing only comments (those beginning by //). There were lines with directives for the compiler's preprocessor (those beginning by #). Then there were lines that began the declaration of a function (in this case, the main function) and, finally lines with statements (like the insertion into cout), which were all included within the block delimited by the braces ({}) of the main function.

The program has been structured in different lines in order to be more readable, but in C++, we do not have strict rules on how to separate instructions in different lines. For example, instead of int main ()

{ cout << " Hello World!"; return 0;

}

We could have written:

int main ()

{ cout << "Hello World!"; return 0; }

All in just one line and this would have had exactly the same meaning as the previous code.

In C++, the separation between statements is specified with an ending semicolon (;) at the end of each one, so the separation in different code lines does not matter at all for this purpose. We can write many statements per line or write a single statement that takes many code lines. The division of

code in different lines serves only to make it more legible and schematic for the humans that may read it.

Let us add an additional instruction to our first program:

// my second program in C++ #include <iostream> using namespace std;

int main ()

{ cout << "Hello World! "; cout << "I'm a C++ program"; return 0;

}

Output:-Hello World! I'm a C++ program

In this case, we performed two insertions into cout in two different statements. Once again, the separation in different lines of code has been done just to give greater readability to the program, since main could have been perfectly valid defined this way:

int main ()

{ cout << " Hello World! ";

cout << " I'm a C++ program ";

return 0;

}

We were also free to divide the code into more lines if we considered it more convenient: int main ()

{ cout << "Hello World!"; cout << "I'm a C++ program"; return 0;

}

**Chapter -3**

**TOKENS:**

The smallest individual units in program are known as **tokens.** C++ has the following

tokens.

i. Keywords ii. Identifiers

iii. Constants iv. Strings

v. Operators

**KEYWORDS:**

The keywords implement specific C++ language feature. They are explicitly reserved

identifiers and can’t be used as names for the program variables or other user defined program elements. The keywords not found in ANSI C are shown in red letter.

**C++ KEYWORDS:**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Asm | double | new | switch |
| Auto | else | operator | template |
| Break | enum | private | this |
| Case | extern | protected | throw |
| Catch | float | public | try |
| Char | for | register | typedef |
| Class | friend | return | union |
| Const | goto | short | unsigned |
| Continue | if | signed | virtual |
| Default | inline | sizeof | void |

Delete long struet while

**IDENTIFIERS:**

Identifiers refers to the name of variable , functions, array, class etc. created by programmer. Each language has its own rule for naming the identifiers.

The following rules are common for both C and C++.

1. Only alphabetic chars, digits and under score are permitted.
2. The name can’t start with a digit.
3. Upper case and lower case letters are distinct.
4. A declared keyword can’t be used as a variable name.

In ANSI C the maximum length of a variable is 32 chars but in c++ there is no b

**BASIC DATA TYPES IN C++**

User

defined

type

Built

in

types

C

++

Data

Types

Array

Function

pointer

Derived

type

Structure

Union

Class

enumeration



|  |  |  |  |
| --- | --- | --- | --- |
| char    int    Integral    type | |  | | --- | | void | | double    Floating    point    float |

Both C and C++ compilers support all the built in types. With the exception of void the basic datatypes may have several modifiers preceding them to serve the needs of various situations. The modifiers signed, unsigned, long and short may applied to character and integer basic data types.

However the modifier long may also be applied to double.

Data types in C++ can be classified under various categories.

|  |  |  |  |
| --- | --- | --- | --- |
| **TYPE** | **BYTES** | **RANGE** | |
| char | 1 | -128 to – 127 | |
| usigned | 1 | 0 to 265 | |
| sgned char | 1 | | -128 to 127 | |
| int | 2 | | -32768 to 32768 | |
| unsigned int | 2 | | 0 to 65535 | |
| singed int | 2 | | -32768 to 32768 | |
| short int | 2 | | -32768 to 32768 | |
| long int | 4 | | -2147483648 to 2147483648 | |
| signed long int | 4 | | -2147483648 to 2147483648 | |
| unsigned long int | 4 | | 0 to 4294967295 | |
| float | 4 | | 3.4E-38 to 3.4E+38 | |
| double | 8 | | 1.7E -308 to 1.7E +308 | |
| long double | 10 | | 3.4E-4932 to 1.1E+ 4932 | |

The type void normally used for:

* 1. To specify the return type of function when it is not returning any value.
  2. To indicate an empty argument list to a function.

Example:

**Void function(void);**

Another interesting use of void is in the declaration of genetic pointer Example:

**Void \*gp;**

Assigning any pointer type to a void pointer without using a cast is allowed in both C and ANSI C. In ANSI C we can also assign a void pointer to a non-void pointer without using a cast to non void pointer type. This is not allowed in C ++.

**USER DEFINED DATA TYPES:**

**STRUCTERS AND CLASSES**

We have used user defined data types such as struct,and union in C. While these 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 a variable. The class variables are known as objects, which are the central focus of oops

**SYMBOLIC CONSTANT:**

There are two ways of creating symbolic constants in c++.

1. using the qualifier const.
2. defining a set of integer constants using enum keywords.

In both C and C++, any value declared as const can’t be modified by the program in any way. In C++, we can use const in a constant expression. Such as

**const int size = 10 ; char name (size) ;**

This would be illegal in C. const allows us to create typed constants instead of having to use #defme to create constants that have no type information.

**const size=10;**

**Means**

**const int size =10;**

C++ requires a const to be initialized. ANSI C does not require an initializer, if none is given, it initializes the const to 0.

In C++ const values are local and in ANSI C const values are global .However they can be made local made local by declaring them as static .In C++ if we want to make const value as global then declared **DECLARATION OF VARIABLES:**

In ANSIC C all the variable which is to be used in programs must be declared at the beginning of the program .But in C++ we can declare the variables any whose in the program where it requires .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.

**REFERENCE VARIABLES:**

C++interfaces a new kind of variable known as the reference variable. A references 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 the variuble.

A reference variable is created as follows:

**Synatx: Datatype & reference –name=variable name;**

Example:

**float total=1500; float &sum=total;**

Here sum is the alternative name for variables total, both the variables refer to the same data object in the memory .

A reference variable must be initialized at the time of declaration .

Note that C++ assigns additional meaning to the symbol *&* here *&* is not an address operator .The notation float *&* means reference to float.

Example:

**int n[10];**

**int &x=n[10]; char &a=’\n’;**

**L 9**

**E**

**C**

**T OPERATORSU IN C++ :**

**R**

C++ has a rich set of operators. All C operators are valid in C++ also. In

**E**

addition. C++ introduces some new operators.

**-**

|  |  |  |
| --- | --- | --- |
| < <    i  n s    e r    t  i    o  n      o p | on operator  : :  : :\*  \*  .\*  Delete  Endl  New  Setw | scope resolution operator pointer to member declarator pointer to member operator pointer to member operator memory release operator line feed operator memory allocation operator field width operator |

e r **SCOPE RESOLUTION OPERATOR:**

a

t Like C,C++ is also a block-structured language. Block -structured o language. Blocks and scopes can be used in constructing programs. We know same r variables can be declared in different blocks because the variables declared in blocks

are local to that function.

>

>**Memory Management Operator**

e C uses malloc and calloc functions to allocate memory dynamically at run time . Similarly it x uses the functions Free( ) to free dynamically allocated memory. We use dynamic t allocation techniques when it is not known in advance how much of memory space as r needed . a c C++ also support those functions it also defines two unary operators new t and delete that perform the task of allocating and freeing the memory in a better and easier way. i

### Chapter -4

**CONTROL STRUCTURES:**

Like c,c++, supports all the basic control structures and implements them various control statements.

**The if statement:**

The if statement is impklemented in two forms:

1. simple if statement

1. if… else statement

**Simple if statement:**

if (condition)

{

Action;

}

**If.. else statement**

If (condition)

Statment1

Else

Statement2

**The switch statement**

This is a multiple-branching statement where, based on a condition, the control is transferred to one of the many possible points;

Switch(expr)

{

case 1: action1; break; case 2: action2; break;

..

..

default: message

}

**The while statement:**

Syn:

While(condition)

{

Stements

**The do-while statement:**

Syn:

do

{

Stements

} while(condition);

**The for loop:**

for(expression1;expression2;expression3)

{

Statements;

Statements;

}

**FUNCTION IN C++ :**

**The main( ) Functon ;**

ANSI does not specify any return type for the main ( ) function which is the starting point for the execution of a program . The definition of main( ) is :-

main()

{

//main program statements

}

This is property valid because the main () in ANSI C does not return any value. 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 terminating.

The main () function in C++ is therefore defined as follows.

int main( )

{

return(0) }

Since the return type of functions is int by default, the key word int in the main( ) header is optional.

**INLINE FUNCTION:**

To eliminate the cost of calls to small functions C++ proposes a new feature called inline function. An inline function is a function that is expanded inline when it is invoked .That is the compiler replaces the function call with the corresponding function code.

The inline functions are defined as follows:-

inline function-header

{

function body;

}

Example: inline double cube (double a)

{ return(a\*a\*a);

}

The above inline function can be invoked by statements like c=cube(3.0); d=cube(2.5+1.5);

remember that the inline keyword merely sends a request, not a command to the compliler. The compiler may ignore this request if the function definition is too long or too complicated and compile the function as a normal function.

C++ allows us to call a function with out specifying all its arguments.In such cases, the function assigns a default value to the parameter which does not have a matching aguments 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.

Example: float amount (float principle, int period ,float rate=0.15);

The default value is specified in a manner syntactically similar to a variable

initialization .The above prototype declares a default value of 0.15 to the argument rate. A subsequent function call like value=amount(5000,7); //one argument missing

passes the value of 5000 to principle and 7 to period and then lets the function, use default value of 0.15 for rate.

The call:- value=amount(5000,5,0.12);

//no missing argument passes an explicite value of 0.12 rate.

One important point to note is that only the trailing arguments can have default values. That is, we must add default from right to left .We cannot provide a default to a particular argument in the middle of an argument list.

Example:- int mul(int i, int j=5,int k=10);//illegal

int mul(int i=0,int j,int k=10);//illegal int mul(int i=5,int j);//illegal int mul(int i=2,int j=5,int k=10);//illegal

Default arguments are useful in situation whose some arguments always have the some value.

For example,bank interest may retain the same for all customers for a particular period of deposit.

Example:

**#include<iostream.h> #include<stdio.h> mainQ**

**{**

**float amount; float value(float p,int n,float r=0.15);**

**void printline(char ch=’\*’,int len=40); printline( ); amount=value(5000.00,5); cout<<”\n final value=”<<amount<<endl; printline(‘=’); //function definitions float value (float p,int n, float r)**

**{ float si; si=(p\*n\*r)/100; return(si); }**

**void printline (char ch,int len)**

**{ for(inti=l;i<=len;i++) cout<<ch<<endl;**

**}**

**output:-**

**\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* final value=10056.71613**

**= = = = = = = = = = = = = = =**

Advantage of providing the default arguments are:

1. We can use default arguments to add new parameters to the existing functions.
2. Default argument s can be used to combine similar functions into one.

**CONST ARGUMENT:-**

In C++, an argument to a function can be declared as unit as const as shown below. int strlen(const char \*p); int length(const string &s);

The qualifier const tells the compiler that the function should not modify the argument .the compiler will generate an error when this condition is violated .This type of declaration is significant only when we pass arguments by reference or pointers.

### Chapter -5

**FUNCTION OVERLOADING:**

Overloading refers to the use of the same thing for different purposes . C++ also

permits overloading functions .This means that we can use the same function name to creates functions that perform a variety of different tasks. This is known as function polymorphism in oops.

Using the concepts of function overloading , a family of functions with one function

name but with different argument lists in the functions call .The correct function to be invoked is determined by checking the number and type of the arguments but not on the function type.

For example an overloaded add() function handles different types of data as shown

below.

**//Declaration int add(int a, int b); //prototype 1 int add (int a, int b, int c); //prototype 2 double add(double x, double y); //prototype 3 double add(double p , double q); //prototype 4**

**//function call cout<<add(5,10); //uses prototype 1 cout<<add(15,10.0); //uses prototype 4 cout<<add(12.5,7.5); //uses prototype 3 cout<<add(5,10,15); //uses prototype 2 cout<<add(0.75,5); //uses prototype 5**

A function call first matches the prototype having the same no and type of arguments and then calls the appropriate function for execution.

The function selection invokes the following steps:-

1. The compiler first tries to find an exact match in which the types of actual arguments are the same and use that function .
2. If an exact match is not found the compiler uses the integral promotions to the actual arguments such as :

char to int

float to double to find a match

1. When either of them tails ,the compiler tries to use the built in conversions to the actual arguments and them uses the function whose match is unique . If the conversion is possible to have multiple matches, then the compiler will give error message.

Example: long square (long n); double square(double x); A function call such as :- square(lO)

Will cause an error because int argument can be converted to either long or

double .There by creating an ambiguous situation as to which version of square( )should be used.

**PROGRAM**

#include<iostream.h> int volume(double,int); double volume( double , int ); double volume(longint ,int ,int);

main( )

{ cout<<volume(10)<<endl; cout<<volume(10)<<endl; cout<<volume(10)<<endl;

} int volume( ini s)

{ return (s\*s\*s); //cube

} double volume( double r, int h)

{ return(3.1416\*r\*r\*h); //cylinder

} long volume (longint 1, int b, int h)

{ return(1\*b\*h); //cylinder

**Module-2:**

### Chapter -6

**CLASS:-**

Class is a group of objects that share common properties and relationships .In C++, a class is a new data type that contains member variables and member functions that operates on the variables. A class is defined with the keyword class. It allows the data to be hidden, if necessary from external use. When we defining a class, we are creating a new abstract data type that can be treated like any other built in data type.

Generally a class specification has two parts:-

1. Class declaration
2. Class function definition

the class declaration describes the type and scope of its members. The class function definition describes how the class functions are implemented.

Syntax:- class class-name

{ private: variable declarations; function declaration ; public: variable declarations; function declaration;

};

The members that have been declared as private can be accessed only from with in the class. On the other hand , public members can be accessed from outside the class also. The data hiding is the key feature of oops. The use of keywords private is optional by default, the members of a class are private.

The variables declared inside the class are known as data members and the functions

are known as members mid the functions. Only the member functions can have access to the private data members and private functions. However, the public members can be accessed from the outside the class. The binding of data and functions together into a single class type variable is referred to as encapsulation.

Syntax:- class item { int member; float cost;

public: void getldata (int a ,float b); void putdata (void);

The class item contains two data members and two function members, the data

members are private by default while both the functions are public by declaration. The function getdata() can be used to assign values to the member variables member and cost, and putdata() for displaying their values . These functions provide the only access to the data members from outside the class.

**CREATING OBJECTS:**

Once a class has been declared we can create variables of that type

by using the class name.

Example: item x;

creates a variables x of type item. In C++, the class variables are known as objects. Therefore x is called an object of type item.

item x, y ,z also possible. class item

{

}x ,y ,z; would create the objects x ,y ,z of type item.

**ACCESSING CLASS MEMBER:**

The private data of a class can be accessed only through the member functions of that

class. The main() cannot contains statements that the access number and cost directly.

Syntax: object name.function-name(actual arguments);

Example:- x. getdata(100,75.5);

It assigns value 100 to number, and 75.5 to cost of the object x by

implementing the getdata() function . similarly the statement

x. putdata ( ); //would display the values of data members.

x. number = 100 is illegal .Although x is an object of the type item to which number belongs , the number can be accessed only through a member function and not by the object directly.

Example: class xyz

{

Int x;

Int y; public:

int z;

};

xyz p;

|  |  |
| --- | --- |
| p. x =0; | error . x is private |
| p, z=10; | ok ,z is public |

**DEFINING MEMBER FUNCTION:**

Member can be defined in two places

* Outside the class definition
* Inside the class function

**OUTSIDE THE CLASS DEFlNAT1ON;**

Member function that are declared inside a class have to be defined separately

outside the class.Their definition are very much like the normal functions.

An important difference between a member function and a normal

function is that a member function incorporates a membership.Identify label in the header. The ‘label’ tells the compiler which class the function belongs to.

Syntax:

return type class-name::function-name(argument declaration )

{ function-body

}

The member ship label class-name :: tells the compiler that the function function -

name belongs to the class class-name . That is the scope of the function is restricted to the class- name specified in the header line. The :: symbol is called scope resolution operator.

Example: void item :: getdata (int a , float b )

{ number=a; cost=b;

} void item :: putdata ( void)

{

cout<<”number=:”<<number<<endl; cout<<”cost=”<<cost<<endl;

}

The member function have some special characteristics that are often used in the program development.

* Several different classes can use the same function name. The "membership label" will resolve their scope, member functions can access the private data of the class .A non member function can't do so.
* A member function can call another member function directly, without using the dot operator.

**INSIDE THE CLASS DEF1NATION:**

Another method of defining a member function is to replace the function declaration by the actual function definition inside the class .

Example: class item

{

Intnumber; float cost;

public:

void getdata (int a ,float b); void putdata(void)

{

}

};

**A C++ PROGRAM WITH CLASS:** cout<<number<<endl; cout<<cost<<endl;

# include< iostream. h> class item

{ int number; float cost;

public: void getdata ( int a , float b); void putdala ( void)

{

cout<<“number:”<<number<<endl; cout<<”cost :”<<cost<<endl;

}

};

**Q.**Write a simple program using class in C++ to input subject mark and prints it. ans:

class marks

{ private :

int ml,m2; public:

void getdata(); void displaydata();

}; void marks: :getdata()

{ cout<<”enter 1st subject mark:”; cin>>ml;

cout<<”enter 2nd subject mark:”; cin>>m2;

} void marks: :displaydata()

{ cout<<”Ist subject mark:”<<ml<<endl ; cout<<”2nd subject mark:”<<m2;

} void main() { clrscr(); marks x;

x.getdata();

**Chapter -7**

**NESTING OF MEMBER FUNCTION;**

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); void 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:”; cin>>m>>n;

} void set::display(void)

{

cout<<”largestvalue=”<<largest()<<”\n”;

} void main() {

set A;

A.input( );

A.display( );

}

output:

Input values of m and n:

3017 largest value= 30

**Private member functions:**

Although it is a normal practice to place all the data items in a private section and all the functions in public, some situations may require contain functions to be hidden from the outside calls. Tasks such as deleting an account in a customer file or providing increment to and 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 can not invoke a private function using the dot operator.

Class sample

{ int m; void read (void); void write (void);

}; if si is an object of sample, then s.read(); is illegal. How ever the function read() can be called by the function update ( ) to update the value of m. void sample :: update(void)

{ read( );

}

#include<iostream.h> class part

{ private:

int modelnum,partnum; float cost;

public:

void setpart ( int mn, int pn ,float c)

{

modelmim=mn; partnum=pn; cost=e;

} void showpart ( )

{

Cout<<endl<<”model:”<<modelnum<<end1;

Cout<<”num:”<< partnum <<endl

Cout<<”cost:”<<”$”<cost;

} }; void main() { part pl,p2; p1.setpart(644,73,217.55); p2.setpart(567,89,789.55); pl.showpart(); pl.showpart();

}

output:- model:644 num:73 cost: $217550003 model: 567 num:89 cost: $759.549988

#indude<iostream.h> class distance

{ private:

int feet; float inches; public:

void setdist ( int ft, float in)

{ feet=ft; inches=in;

} void getdist()

{

cout<<”enter feet:”; cin>>feet; cout<<”enter inches:”; cin>>inches;

} void showdist()

{

cout<< feet<<”\_”inches«endl;

} }; void main( )

{ distance dl,d2; d1.setdist(1 1,6.25); d2.getdata();

cout<<endl<<”dist:”<<d 1 .showdist(); cout<<”\n”<<”dist2:”; d2.showdist();

}

**output:-** enter feet: 12 enter inches: 6.25 dist 1:”11’- 6.1.5” dist 2: 12’- 6.25

*CONCLUSION AND FUTURE PRESPECTIVE*

Oop offers several benefits to both the program designer and the user. Object-oriented contributes to the solution of many problems associated with the development and quality of software products. The

principal advantages are : 1. Through inheritance we can eliminate redundant code and extend the use of

existing classes. 2. We can build programs from the standard working modules that communicate with one

another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity. 3. This principle of data hiding helps the programmer to build secure programs that can’t be invaded by code in other parts of the program. 4. It is possible to have multiple instances of an

object to co-exist with out any interference. 5. It is easy to partition the work in a project based on objects.

6. Object-oriented systems can be easily upgraded from small to large systems. 7. Message passing techniques for communication between objects makes the interface description with external systems much

simpler. 8. Software complexity can be easily managed

P.T.O

Bibliography

***Text Books:***

1. Object Oriented Programming with C++ by E. Balagurusamy, McGraw-Hill Education (India) 2.ANSI and Turbo C++ by Ashoke N. Kamthane, Pearson Education

***Reference Books:***

1. Big C++ - Wiley India
2. C++: The Complete Reference- Schildt, McGraw-Hill Education (India)
3. C++ and Object Oriented Programming – Jana, PHI Learning
4. Object Oriented Programming with C++ - Rajiv Sahay, Oxford
5. Mastering C++ - Venugopal, McGraw-Hill Education (India)

P.T.O