

Notes

**Subject: Object Oriented Programming
through C++ (IT-03)**

**Class: Information Technology BSc. 2nd
year (Semester –III)**

Syllabus for B. Sc. 2nd year (Semester-III)

Subject: Information Technology

OBJECT ORIENTED PROGRAMMING THROUGH C++ (IT-03)

UNIT-I: Introduction to Object Oriented Programming

Introduction: Object Oriented Programming (OOP), need for OOP, characteristics of OOP, OOP Concepts—Classes, Objects, Abstraction, Encapsulation, Inheritance and polymorphism.

C++ Basics: Overview, Program, Structure, identifiers, variables, constants, enums, operators, typecasting.

UNIT-II: Control Structures and Functions

Control Structures: if, if-else, switch-case, while, do-while, for etc.

C++ Functions: friend functions, in-line functions, macro vs in-line functions, reference variables and call by reference, function overloading and default arguments.

UNIT-III: Objects, Classes and Inheritance

Objects and classes: Basics of object and class in C++, Public and private members, Scope resolution operator, Access specifiers and accessing class members, static data and function members, Constructors and their types, Destructors, operator overloading (unary, binary), type conversion, virtual class, friend class.

Inheritance: Concept of inheritance, types of inheritance, single, multiple, multilevel, hierarchical, hybrid, protected members, overriding, virtual base class.

UNIT-IV: Polymorphism, I/O and File Management

Polymorphism: Pointers in C++, pointers and objects, virtual and pure virtual functions, abstract class, implementing polymorphism.

I/O and File Management: Concept of streams, cin and cout objects, formatted and unformatted I/O, manipulators, file stream, C++ file stream classes, file management functions, Binary and random files.

Recommended Books:

1. "Object Oriented Programming with C++" E Balagurusamy, TMH
2. "Object Oriented Programming in Turbo C++", Robert Lafzre
3. "Object Oriented Programming with ANSI and Turbo C++", Ashok Kamthane, Pearson
4. "The Complete Reference C++", Herbert Schildt, TMH

Unit-I

Introduction

Object oriented Programming

Object oriented Programming is defined 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. Writing object-oriented programs involves creating classes, creating objects from those classes, and creating applications, which are stand-alone executable programs that use those objects. After being created, classes can be reused over and over again to develop new programs. Thinking in an object-oriented manner involves envisioning program components as objects that belong to classes and are similar to concrete objects in the real world; then, you can manipulate the objects and have them interrelate with each other to achieve a desired result.

Basic Concepts of Object oriented Programming

1. Class

A class is a user defined data type. A class is a logical abstraction. It is a template that defines the form of an object. A class specifies both code and data. It is not until an object of that class has been created that a physical representation of that class exists in memory. When you define a class, you declare the data that it contains and the code that operates on that data. Data is contained in instance variables defined by the class known as data members, and code is contained in functions known as member functions. The code and data that constitute a class are called members of the class.

2. Object

An object is an identifiable entity with specific characteristics and behavior. An object is said to be an instance of a class. Defining an object is similar to defining a variable of any data type. Space is set aside for it in memory.

3. Encapsulation

Encapsulation is a programming mechanism that binds together code and the data it manipulates, and that keeps both safe from outside interference and misuse. C++'s basic unit of encapsulation is the class. Within a class, code or data or both may be private to that object or public. Private code or data is known to and accessible by only another part of the object. That is, private code or data cannot be accessed by a piece of the program that exists outside the object. When code or data is public, other parts of your program can access it even though it is defined within an object. Typically, the public parts of an object are used to provide a controlled interface to the private elements of the object. This insulation of the data from direct access by the program is called data hiding.

4. Data abstraction

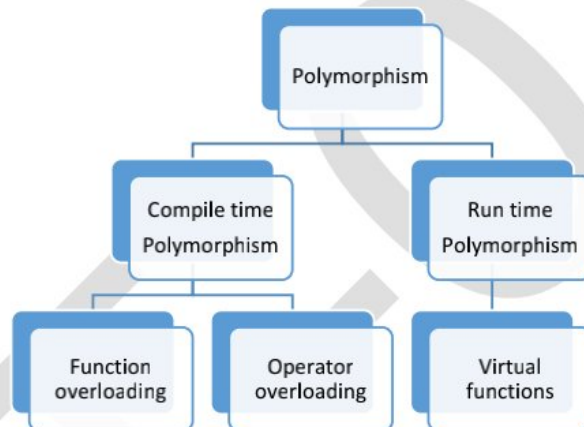
In object oriented programming, each object will have external interfaces through which it can be made use of. There is no need to look into its inner details. The object itself may be made of many smaller objects again with proper interfaces. The user needs to know the external interfaces only to make use of an object. The internal details of the objects are hidden which makes them abstract. The technique of hiding internal details in an object is called data abstraction.

5. Inheritance

Inheritance is the mechanism by which one class can inherit the properties of another. It allows a hierarchy of classes to be build, moving from the most general to the most specific. When one class is inherited by another, the class that is inherited is called the base class. The inheriting class is called the derived class. In general, the process of inheritance begins with the definition of a base class. The base class defines all qualities that will be common to any derived class. . In OOPs, the concept of inheritance provides the idea of reusability. In essence, the base class represent the most general description of a set of traits. The derived class inherits those general traits and adds properties that are specific to that class.

6. Polymorphism

Polymorphism (from the Greek, meaning “many forms”) is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation. The concept of polymorphism is often expressed by the phrase “one interface, multiple methods.” This means that it is possible to design a generic interface to a group of related activities. This helps reduce complexity by allowing the same interface to be used to specify a general class of action. It is the compiler’s job to select the specific action as it applies to each situation.



In compile time polymorphism, the compiler is able to select the appropriate function for a particular call at compile time. In C++, it is possible to use one function name for many different purposes. This type of polymorphism is called function overloading. Polymorphism can also be applied to operators. In that case, it is called operator overloading.

In run time polymorphism, the compiler selects the appropriate function for a particular call while the program is running. C++ supports a mechanism known as virtual functions to achieve run time polymorphism.

Need for Object oriented Programming

Object-oriented programming scales very well, from the most trivial of problems to the most complex tasks. It provides a form of abstraction that resonates with techniques people use to solve problems in their everyday life.

Object-oriented programming was developed because limitations were discovered in earlier approaches to programming. There were two related problems. First, functions have unrestricted access to global data. Second, unrelated functions and data, the basis of the procedural paradigm, provide a poor model of the real world.

Benefits of Object oriented Programming

1. **Simplicity:** Software objects model real world objects, so the complexity is reduced and the program structure is very clear.
2. **Modularity:** Each object forms a separate entity whose internal workings are decoupled from other parts of the system.
3. **Modifiability:** It is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods.
4. **Extensibility:** adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones.
5. **Maintainability:** objects can be maintained separately, making locating and fixing problems easier.
6. **Re-usability:** objects can be reused in different programs.

C++

C++ is an object oriented programming language. It was developed by Bjarne Stroustrup in 1979 at Bell Laboratories in Murray Hill, New Jersey. He initially called the new language "C with Classes." However, in 1983 the name was changed to C++.

C++ is a superset of C. Stroustrup built C++ on the foundation of C, including all of C's features, attributes, and benefits. Most of the features that Stroustrup added to C were designed to support object-oriented programming. These features comprise of classes, inheritance, function overloading and operator overloading. C++ has many other new features as well, including an improved approach to input/output (I/O) and a new way to write comments.

C++ is used for developing applications such as editors, databases, personal file systems, networking utilities, and communication programs. Because C++ shares C's efficiency, much high-performance systems software is constructed using C++.

A Simple C++ Program

```
#include<iostream.h>
#include<conio.h>
int main()
{
    cout<< "Simple C++ program without using class";
    return 0;
}
```

Lines beginning with a hash sign (#) are directives read and interpreted by what is known as the preprocessor. They are special lines interpreted before the compilation of the program itself begins. In this case, the directive `#include <iostream.h>`, instructs the preprocessor to include a section of standard C++ code, known as header `iostream` that allows to perform standard input and output operations, such as writing the output of this program to the screen.

The function named `main` is a special function in all C++ programs; it is the function called when the program is run. The execution of all C++ programs begins with the `main` function, regardless of where the function is actually located within the code.

The open brace (`{`) indicates the beginning of `main`'s function definition, and the closing brace (`}`) indicates its end.

The statement :

```
cout<< "Simple C++ program without using class";
```

causes the string in quotation marks to be displayed on the screen. The identifier `cout` (pronounced as `c out`) denotes an object. It points to the standard output device namely the console monitor. The operator `<<` is called insertion operator. It directs the string on its right to the object on its left.

The program ends with this statement:

```
return 0;
```

This causes zero to be returned to the calling process (which is usually the operating system). Returning zero indicates that the program terminated normally. Abnormal program termination should be signaled by returning a nonzero value.

The general structure of C++ program with classes is shown as:

1. Documentation Section
2. Preprocessor Directives or Compiler Directives Section
 - (i) Link Section
 - (ii) Definition Section
3. Global Declaration Section
4. Class declaration or definition
5. Main C++ program function called `main ()`

C++ keywords

When a language is defined, one has to design a set of instructions to be used for communicating with the computer to carry out specific operations. The set of instructions which are used in programming, are called keywords. These are also known as reserved words of the language. They have a specific meaning for the C++ compiler and should be used for giving specific instructions to the computer. These words cannot be used for any other purpose, such as naming a variable. C++ is a case-sensitive language, and it requires that all keywords be in lowercase. C++ keywords are:

asm	auto	bool	break
case	catch	char	class
const	const_cast	continue	default
delete	do	double	dynamic_cast
else	enum	explicit	export
extern	false	float	for
friend	goto	if	inline
int	long	mutable	namespace
new	operator	private	protected
public	register	reinterpret_cast	return
short	signed	sizeof	static
static_cast	struct	switch	template
this	throw	true	try
typedef	typeid	typename	union
unsigned	using	virtual	void
volatile	wchar_t	while	

Identifiers

An identifier is a name assigned to a function, variable, or any other user-defined item. Identifiers can be from one to several characters long.

Rules for naming identifiers:

- Variable names can start with any letter of the alphabet or an underscore. Next comes a letter, a digit, or an underscore.
- Uppercase and lowercase are distinct.
- C++ keywords cannot be used as identifier.

Data types

Data type defines size and type of values that a variable can store along with the set of operations that can be performed on that variable. C++ provides built-in data types that correspond to integers, characters, floating-point values, and Boolean values. There are the seven basic data types in C++ as shown below:

Type	Meaning
char(character)	holds 8-bit ASCII characters
wchar_t(Wide character)	holds characters that are part of large character sets
int(Integer)	represent integer numbers having no fractional part
float(floating point)	stores real numbers in the range of about 3.4×10^{-38} to 3.4×10^{38} , with a precision of seven digits.

double(Double floating point)	Stores real numbers in the range from 1.7×10^{-308} to 1.7×10^{308} with a precision of 15 digits.
bool(Boolean)	can have only two possible values: true and false.
Void	Valueless

C++ allows certain of the basic types to have modifiers preceding them. A modifier alters the meaning of the base type so that it more precisely fits the needs of various situations. The data type modifiers are: signed, unsigned, long and short

Type	Minimal Range
char	-127 to 127
unsigned char	0 to 255
signed char	-127 to 127
int	-32,767 to 32,767
unsigned int	0 to 65,535
signed int	Same as int
short int	-32,767 to 32,767
unsigned short int	0 to 65,535
signed short int	Same as short int
long int	-2,147,483,647 to 2,147,483,647
signed long int	Same as long int
unsigned long int	0 to 4,294,967,295
float	$1E-37$ to $1E+37$, with six digits of precision
double	$1E-37$ to $1E+37$, with ten digits of precision
long double	$1E-37$ to $1E+37$, with ten digits of precision

This figure shows all combinations of the basic data types and modifiers along with their size and range for a 16-bit word machine

Variable

A variable is a named area in memory used to store values during program execution. Variables are run time entities. A variable has a symbolic name and can be given a variety of values. When a variable is given a value, that value is actually placed in the memory space assigned to the variable. All variables must be declared before they can be used. The general form of a declaration is:

```
type variable_list;
```

Here, type must be a valid data type plus any modifiers, and variable_list may consist of one or more identifier names separated by commas. Here are some declarations:

```
int i,j,l;
```

```
short int si;
```

```
unsigned int ui;
```

```
double balance, profit, loss;
```


Constants

Constants refer to fixed values that the program cannot alter. Constants can be of any of the basic data types. The way each constant is represented depends upon its type. Constants are also called literals. We can use keyword const prefix to declare constants with a specific type as follows:

```
const type variableName = value;
```

e.g,

```
const int LENGTH = 10;
```

Enumerated Types

An enumerated type declares an optional type name and a set of zero or more identifiers that can be used as values of the type. Each enumerator is a constant whose type is the enumeration. Creating an enumeration requires the use of the keyword enum. The general form of an enumeration type is:

```
enum enum-name { list of names } var-list;
```

Here, the enum-name is the enumeration's type name. The list of names is comma separated.

For example, the following code defines an enumeration of colors called color and the variable c of type color. Finally, c is assigned the value "blue".

```
enum color { red, green, blue } =c;
```

By default, the value of the first name is 0, the second name has the value 1 and the third has the value 2, and so on. But you can give a name, a specific value by adding an initializer. For example, in the following enumeration, green will have the value 5.

```
enum color { red, green=5, blue };
```

Here, blue will have a value of 6 because each name will be one greater than the one that precedes it.

Operator

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ is rich in built-in operators. Generally, there are six type of operators: Arithmetical operators, Relational operators, Logical operators, Assignment operators, Conditional operators, Comma operator.

Arithmetical operators

Arithmetical operators +, -, *, /, and % are used to performs an arithmetic (numeric) operation.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

You can use the operators +, -, *, and / with both integral and floating-point data types. Modulus or remainder % operator is used only with the integral data type.

Relational operators

The relational operators are used to test the relation between two values. All relational operators are binary operators and therefore require two operands. A relational expression returns zero when the relation is false and a non-zero when it is true. The following table shows the relational operators.

Relational Operators	Meaning
<	Less than
<=	Less than or equal to
==	Equal to
>	Greater than
>=	Greater than or equal to
!=	Not equal to

Logical operators

The logical operators are used to combine one or more relational expression. The logical operators are

Operators	Meaning
	OR
&&	AND
!	NOT

Assignment operator

The assignment operator '=' is used for assigning a variable to a value. This operator takes the expression on its right-hand-side and places it into the variable on its left-hand-side. For example:

```
m = 5;
```

The operator takes the expression on the right, 5, and stores it in the variable on the left, m.

```
x = y = z = 32;
```

This code stores the value 32 in each of the three variables x, y, and z. In addition to standard assignment operator shown above, C++ also support compound assignment operators.

Compound Assignment Operators

Operator	Example	Equivalent to
+=	A += 2	A = A + 2
-=	A -= 2	A = A - 2
%=	A %= 2	A = A % 2
/=	A /= 2	A = A / 2
*=	A *= 2	A = A * 2

Increment and Decrement Operators

C++ provides two special operators viz '++' and '--' for incrementing and decrementing the value of a variable by 1. The increment/decrement operator can be used with any type of variable but it cannot be used with any constant. Increment and decrement operators each have two forms, pre and post.

The syntax of the increment operator is:

Pre-increment: ++variable

Post-increment: variable++

The syntax of the decrement operator is:

Pre-decrement: —variable

Post-decrement: variable—

In Prefix form first variable is first incremented/decremented, then evaluated

In Postfix form first variable is first evaluated, then incremented / decremented.

Conditional operator

The conditional operator ?: is called ternary operator as it requires three operands. The format of the conditional operator is :

Conditional_ expression ? expression1 : expression2;

If the value of conditional expression is true then the expression1 is evaluated, otherwise expression2 is evaluated.

```
int a = 5, b = 6;
```

```
big = (a > b) ? a : b;
```

The condition evaluates to false, therefore big gets the value from b and it becomes 6.

The comma operator

The comma operator gives left to right evaluation of expressions. When the set of expressions has to be evaluated for a value, only the rightmost expression is considered.

```
int a = 1, b = 2, c = 3, i; // comma acts as separator, not as an operator
```

```
i = (a, b); // stores b into i would first assign the value of a to i, and then assign value of b to variable i.
```

So, at the end, variable i would contain the value 2.

The sizeof operator

The sizeof operator can be used to find how many bytes are required for an object to store in memory. For example

```
sizeof (char) returns 1
```

```
sizeof (float) returns 4
```

Typecasting

Typecasting is the concept of converting the value of one type into another type. For example, you might have a float that you need to use in a function that requires an integer.

Implicit conversion

Almost every compiler makes use of what is called automatic typecasting. It automatically converts one type into another type. If the compiler converts a type it will normally give a warning. For example this warning: conversion from 'double' to 'int', possible loss of data. The problem with this is, that you get a warning (normally you want to compile without warnings and errors) and you are not in control. With control we mean, you did not decide to convert to another type, the compiler did. Also the possible loss of data could be unwanted.

Explicit conversion

The C++ language have ways to give you back control. This can be done with what is called an explicit conversion.

Four typecast operators

The C++ language has four typecast operators:

- `static_cast`
- `reinterpret_cast`
- `const_cast`
- `dynamic_cast`

Type Conversion

The Type Conversion is that which automatically converts the one data type into another but remember we can store a large data type into the other. For example we can't store a float into int because a float is greater than int.

When a user can convert the one data type into then it is called as the **type casting**. The type Conversion is performed by the **compiler** but a casting is done by the **user** for example converting a float into int. When we use the Type Conversion then it is called the promotion. In the type casting when we convert a large data type into another then it is called as the demotion. When we use the type casting then we can loss some data.

Unit –II

Control Structures

Control structures allows to control the flow of program's execution based on certain conditions C++ supports following basic control structures:

- 1) Selection Control structure
- 2) Loop Control structure

1) Selection Control structure:

Selection Control structures allows to control the flow of program's execution depending upon the state of a particular condition being true or false .C++ supports two types of selection statements :if and switch. Condition operator (?:) can also be used as an alternative to the if statement.

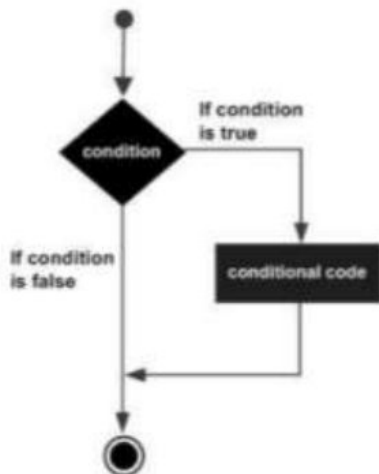
A.1) If Statement:

The syntax of an if statement in C++ is:

```
if(condition)
{
// statement(s) will execute if the condition is true
}
```

If the condition evaluates to true, then the block of code inside the if statement will be executed. If it evaluates to false, then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Flowchart showing working of if statement



```
// A Program to find whether a given number is even or odd using if ... else statement
```

```
#include<iostream.h>
```

```
#include<conio.h>
```

```
int main()
```

```
{ int n;
```

```
cout<<"enter number";
```

```
cin>>n;
```

```
if(n%2==0)
```

```
cout<<"Even number";
```

```
else
```

```
cout<<"Odd number";
```

```
return 0;
```

```
}
```

A.2) The if...else Statement

The syntax is shown as:

```
if(condition){
```

```
// statement(s) will execute if the condition is true
```

```
}
```

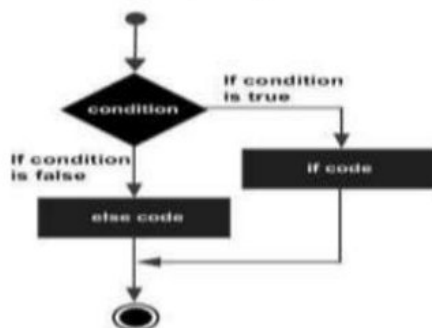
```
else{
```

```
// statement(s) will execute if condition is false
```

```
}
```

If the condition evaluates to true, then the if block of code will be executed, otherwise else block of code will be executed.

Flowchart



A.3) if...else if...else Statement

An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.

The Syntax is shown as:

```
if(condition 1){  
    // Executes when the condition 1 is true  
}  
else if(condition 2){  
    // Executes when the condition 2 is true  
}  
else if(condition 3){  
    // Executes when the condition 3 is true  
}  
else {  
    // executes when the none of the above condition is true.  
}
```

A.4) Nested if Statement

It is always legal to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).

The syntax for a nested if statement is as follows:

```
if( condition 1){  
    // Executes when the condition 1 is true  
    if(condition 2){  
        // Executes when the condition 2 is true  
    }  
}
```

B) Switch

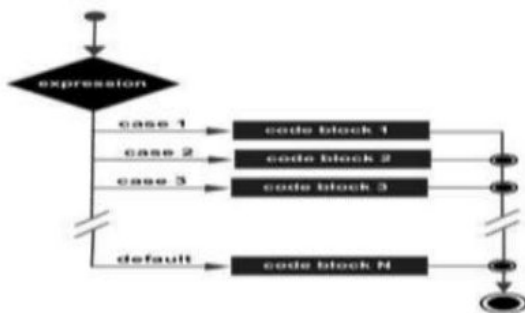
C++ has a built-in multiple-branch selection statement, called switch, which successively tests the value of an expression against a list of integer or character constants. When a match is found, the statements associated with that constant are executed. The general form of the switch statement is:

```
switch (expression) {  
  case constant1:  
    statement sequence  
    break;  
  case constant2:  
    statement sequence  
    break;  
  case constant3:  
    statement sequence  
    break;  
  .  
  .  
  default  
    statement sequence  
}
```

The expression must evaluate to a character or integer value. Floating-point expressions, for example, are not allowed. The value of expression is tested, in order, against the values of the constants specified in the case statements. When a match is found, the statement sequence associated with that case is executed until the break statement or the end of the switch statement is reached. The default statement is executed if no matches are found. The default is optional and, if it is not present, no action takes place if all matches fail.

The break statement is one of C++'s jump statements. You can use it in loops as well as in the switch statement. When break is encountered in a switch, program execution "jumps" to the line of code following the switch statement.

Flowchart



2) Loop control structures

A loop statement allows us to execute a statement or group of statements multiple times. Loops or iterative statements tell the program to repeat a fragment of code several times or as long as a certain condition holds. C++ provides three convenient iterative statements: while, for, and do-while.

while loop

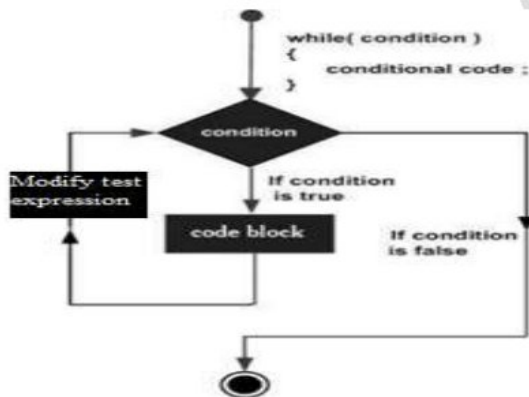
A while loop statement repeatedly executes a target statement as long as a given condition is true. It is an entry-controlled loop.

The syntax of a while loop in C++ is:

```
while(condition){  
statement(s);  
}
```

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true. After each execution of the loop, the value of test expression is changed. When the condition becomes false, program control passes to the line immediately following the loop.

Flowchart



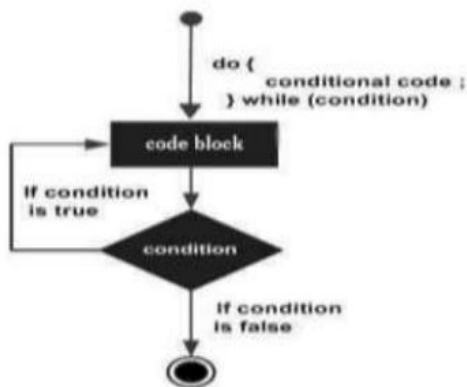
```
// A program to display numbers from 1 to 100  
#include<iostream.h>  
#include<conio.h>  
int main(){  
int i=1;  
while(i<=100){  
cout<<i ;  
i++;  
}  
return 0;  
}
```

The do-while Loop

The do-while loop differs from the while loop in that the condition is tested after the body of the loop. This assures that the program goes through the iteration at least once. It is an exit-controlled loop.

```
do{  
statement(s);  
}while( condition );
```

The conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested. If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.



```
// A program to display numbers from 1 to 100  
#include<iostream.h>  
#include<conio.h>  
int main(){  
int i=1;  
do  
{  
cout<<i ;  
i++;  
} while(i<=100);  
return 0;  
}
```

for Loop

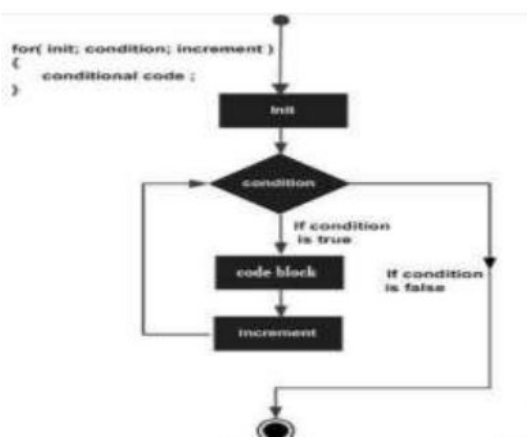
A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. The syntax of a for loop in C++ is:

```
for ( init; condition; increment ){  
statement(s);  
}
```

Here is the flow of control in a for loop:

1. The init step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
2. Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
3. After the body of the for loop executes, the flow of control jumps back up to the increment statement. This statement allows you to update any C++ loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
4. The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.

Flow Diagram



// A program to display numbers from 1 to 100

```
#include<iostream.h>
```

```
#include<conio.h>
```

```
int main() {
```

```
    int i ;
```

```
    for (i=1;i<=100;i++)
```

```
    {
```

```
        cout<<i ;
```

```
    return 0;
```

```
    }
```

C++ Functions

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. The function's code is stored in only one place in memory, even though the function is executed many times in the course of the program's execution. Functions help to reduce the program size when same set of instructions are to be executed again and again. A general function consists of three parts, namely, function declaration (or prototype), function definition and function call.

Function declaration — prototype:

A function has to be declared before using it, in a manner similar to variables and constants. A function declaration tells the compiler about a function's name, return type, and parameters and how to call the function. The general form of a C++ function declaration is as follows:

```
return_type function_name( parameter list );
```

Function definition

The function definition is the actual body of the function. The function definition consists of two parts namely, function header and function body.

The general form of a C++ function definition is as follows:

```
return_type function_name( parameter list )
{
    body of the function
}
```

Here, **Return Type**: A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.

Function Name: This is the actual name of the function.

Parameters: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

Function Body: The function body contains a collection of statements that define what the function does.

Calling a Function

To use a function, you will have to call or invoke that function. To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value.

A c++ program calculating factorial of a number using functions

```
#include<iostream.h>
#include<conio.h>
int factorial(int n);           //function declaration
int main(){
```



```

int no, f;
cout<<"enter the positive number:-";
cin>>no;
f=factorial(no);           //function call
cout<<"\nThe factorial of a number"<<no<<"is"<<f;
return 0;
}
int factorial(int n)      //function definition
{   int i , fact=1;
for(i=1;i<=n;i++){
    fact=fact*i;
}
return fact;
}

```

Inline Functions

An inline function is a function that is expanded inline at the point at which it is invoked, instead of actually being called. The reason that inline functions are an important addition to C++ is that they allow you to create very efficient code. Each time a normal function is called, a significant amount of overhead is generated by the calling and return mechanism. Typically, arguments are pushed onto the stack and various registers are saved when a function is called, and then restored when the function returns. The trouble is that these instructions take time. However, when a function is expanded inline, none of those operations occur. Although expanding function calls in line can produce faster run times, it can also result in larger code size because of duplicated code. For this reason, it is best to inline only very small functions. `inline` is actually just a request, not a command, to the compiler. The compiler can choose to ignore it. Also, some compilers may not inline all types of functions. If a function cannot be inlined, it will simply be called as a normal function.

A function can be defined as an inline function by prefixing the keyword `inline` to the function header as given below:

```

inline function header {
function body
}

// A program illustrating inline function
#include<iostream.h>
#include<conio.h>
inline int max(int x, int y){
if(x>y)
return x;

```

```

else
return y;
}
int main() {
int a,b;
cout<<"enter two numbers";
cin>>a>>b;
cout << "The max is: " <<max(a,,b) << endl;
return 0;
}

```

Macros Vs inline functions

Preprocessor macros are just substitution patterns applied to your code. They can be used almost anywhere in your code because they are replaced with their expansions before any compilation starts. Inline functions are actual functions whose body is directly injected into their call site. They can only be used where a function call is appropriate.

inline functions are similar to macros (because the function code is expanded at the point of the call at compile time), inline functions are parsed by the compiler, whereas macros are expanded by the preprocessor. As a result, there are several important differences:

- Inline functions follow all the protocols of type safety enforced on normal functions.
- Inline functions are specified using the same syntax as any other function except that they include the `inline` keyword in the function declaration.
- Expressions passed as arguments to inline functions are evaluated once.
- In some cases, expressions passed as arguments to macros can be evaluated more than once.
- macros are expanded at pre-compile time, you cannot use them for debugging, but you can use inline functions.

Reference variable

A reference variable is an alias, that is, another name for an already existing variable. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable. To declare a reference variable or parameter, precede the variable's name with the `&`. The syntax for declaring a reference variable is:

```
datatype &Ref = variable name;
```

Example:

```

int main(){
int var1=10;           //declaring simple variable
int & var2=var1; //declaring reference variable
cout<<"\n value of var2 =" << var2;

```

```
return 0;
}
```

var2 is a reference variable to var1. Hence, var2 is an alternate name to var1. This code prints the value of var2 exactly as that of var1.

Call by reference

Arguments can be passed to functions in one of two ways: using call-by-value or call-by-reference. When using call-by-value, a copy of the argument is passed to the function. Call-by-reference passes the address of the argument to the function. By default, C++ uses call-by-value.

Provision of the reference variables in c++ permits us to pass parameter to the functions by reference. When we pass arguments by reference, the formal arguments in the called function become aliases to the actual arguments in the calling function. This means that when the function is working with its own arguments, it is actually working on the original data.

Example

```
#include <iostream.h>
#include<conio.h>
void swap(int &x, int &y); // function declaration
int main (){
int a = 10, b=20;
cout << "Before swapping"<<endl;
cout<< "value of a : " << a <<" value of b : " << b << endl;
swap(a, b); //calling a function to swap the values.
cout << "After swapping"<<endl;
cout<<" value of a : " << a<< "value of b : " << b << endl;
return 0;
}
void swap(int &x, int &y) { //function definition to swap the values.
int temp;
temp = x;
x = y;
y = temp;
}
```

Output:

```
Before swapping      value of a:10 value of b:20
After swapping      value of a:20 value of b:10
```

Function Overloading

Function overloading is the process of using the same name for two or more functions. Each redefinition of the function must use either different types of parameters or a different number of parameters. It is only through these differences that the compiler knows which function to call in any given situation. Overloaded functions can help reduce the complexity of a program by allowing related operations to be referred to by the same name. To overload a function, simply declare and define all required versions. The compiler will automatically select the correct version based upon the number and/or type of the arguments used to call the function. Two functions differing only in their return types cannot be overloaded.

Example

```
#include<iostream.h>
#include<conio.h>
int sum(int p,int q,int r);
double sum(int l,double m);
float sum(float p,float q)
int main(){
cout<<"sum="<< sum(11,22,33);           //calls func1
cout<<"sum="<< sum(10,15.5);           //calls func2
cout<<"sum="<< sum(13.5,12.5);       //calls func3
return 0;
}
int sum(int p,int q,int r){           //func1
return(a+b+c);
}
double sum(int l,double m){         //func2
return(l+m);
}
float sum(float p,float q){         //func3
return(p+q);
}
```

Default arguments

C++ allows a function to assign a parameter a default value when no argument corresponding to that parameter is specified in a call to that function. The default value is specified in a manner syntactically similar to a variable initialization. All default parameters must be to the right of any parameters that don't have defaults. We cannot provide a default value to a particular argument in the middle of an argument list. When you create a function that has one or more default arguments, those arguments must be specified only once: either in the function's prototype or in the function's definition if the definition precedes the function's first use.

Default arguments are useful if you don't want to go to the trouble of writing arguments that, for example, almost always have the same value. They are also useful in cases where, after a program is written, the programmer decides to increase the capability of a function by adding another argument. Using default arguments means that the existing function calls can continue to use the old number of arguments, while new function calls can use more.

Example

```
#include <iostream.h>
#include<conio.h>
int sum(int a, int b=20){
return( a + b);
}
int main (){
int a = 100, b=200, result;
result = sum(a, b);      //here a=100 , b=200
cout << "Total value is :" << result << endl;
result = sum(a);        //here a=100 , b=20(using default value)
cout << "Total value is :" << result << endl;
return 0;
}
```

Unit-III

Class

A class is a user defined data type. A class is a logical abstraction. It is a template that defines the form of an object. A class specifies both code and data. It is not until an object of that class has been created that a physical representation of that class exists in memory. When you define a class, you declare the data that it contains and the code that operates on that data. Data is contained in instance variables defined by the class known as data members, and code is contained in functions known as member functions. The code and data that constitute a class are called members of the class. The general form of class declaration is:

```
class class-name {  
    access-specifier:  
    data and functions  
    access-specifier:  
    data and functions  
    // ...  
    access-specifier:  
    data and functions  
} object-list;
```

The object-list is optional. If present, it declares objects of the class. Here, access-specifier is one of these three C++ keywords:

public private protected

By default, functions and data declared within a class are private to that class and may be accessed only by other members of the class. The public access_specifier allows functions or data to be accessible to other parts of your program. The protected access_specifier is needed only when inheritance is involved.

Example:

```
#include<iostream.h>  
#include<conio.h>  
Class myclass {           // class declaration  
// private members to myclass  
int a;  
public:  
// public members to myclass  
void set_a(intnum);  
int get_a( );  
};
```

Object

An object is an identifiable entity with specific characteristics and behavior. An object is said to be an instance of a class. Defining an object is similar to defining a variable of any data type: Space is set aside for it in memory. Defining objects in this way means creating them. This is also called instantiating them. Once a Class has been declared, we can create objects of that Class by using the class Name like any other built-in type variable as shown:

```
className objectName
```

Example

```
void main() {  
myclass ob1, ob2;           //these are object of type myclass  
// ... program code  
}
```

Accessing Class Members

The main() cannot contain statements that access class members directly. Class members can be accessed only by an object of that class. To access class members, use the dot (.) operator. The dot operator links the name of an object with the name of a member. The general form of the dot operator is shown here:

```
object.member
```

Example

```
ob1.set_a(10);
```

The private members of a class cannot be accessed directly using the dot operator, but through the member functions of that class providing data hiding. A member function can call another member function directly, without using the dot operator.

C++ program to find sum of two numbers using classes

```
#include<iostream.h>  
#include<conio.h>  
class A{  
int a,b,c;  
public:  
void sum(){  
cout<<"enter two numbers";  
cin>>a>>b;  
c=a+b;  
cout<<"sum="<<c;  
}  
};
```

```

int main(){
    A u;
    u.sum();
    getch();
    return(0);
}

```

Scope Resolution operator

Member functions can be defined within the class definition or separately using scope resolution operator (::). Defining a member function within the class definition declares the function inline, even if you do not use the inline specifier. Defining a member function using scope resolution operator uses following declaration

```

return-type class-name::func-name(parameter- list) {
// body of function
}

```

Here the class-name is the name of the class to which the function belongs. The scope resolution operator (::) tells the compiler that the function func-name belongs to the class class-name. That is, the scope of the function is restricted to the class-name specified.

```

Class myclass {
int a;
public:
void set_a(intnum); //member function declaration
int get_a(); //member function declaration
};
//member function definition outside class using scope resolution operator
void myclass :: set_a(intnum)
{
a=num;
}
int myclass::get_a() {
return a;
}

```

Another use of scope resolution operator is to allow access to the global version of a variable. In many situation, it happens that the name of global variable and the name of the local variable are same .In

this while accessing the variable, the priority is given to the local variable by the compiler. If we want to access or use the global variable, then the scope resolution operator (::) is used. The syntax for accessing a global variable using scope resolution operator is as follows:-

```
:: Global-variable-name
```

Static Data Members

When you precede a member variable's declaration with static, you are telling the compiler that only one copy of that variable will exist and that all objects of the class will share that variable. Unlike regular data members, individual copies of a static member variable are not made for each object. No matter how many objects of a class are created, only one copy of a static data member exists. Thus, all objects of that class use that same variable. All static variables are initialized to zero before the first object is created. When you declare a static data member within a class, you are not defining it. (That is, you are not allocating storage for it.) Instead, you must provide a global definition for it elsewhere, outside the class. This is done by redeclaring the static variable using the scope resolution operator to identify the class to which it belongs. This causes storage for the variable to be allocated.

One use of a static member variable is to provide access control to some shared resource used by all objects of a class. Another interesting use of a static member variable is to keep track of the number of objects of a particular class type that are in existence.

Static Member Functions

Member functions may also be declared as static. They may only directly refer to other static members of the class. Actually, static member functions have limited applications, but one good use for them is to "preinitialize" private static data before any object is actually created. A static member function can be called using the class name instead of its objects as follows:

```
class name :: function name

//Program showing working of static class members
#include <iostream.h>
#include<conio.h>
class static_type {
static int i;                //static data member
public:
static void init(int x) {i = x;} //static member function
void show() {cout << i;};
int static_type :: i;        // static data member definition
int main(){
static_type::init(100);     //Accessing static function
static_type x;
x.show();
return 0;
}
```


Constructor:

A constructor is a special member function whose task is to initialize the objects of its class. It is special because its name is same as the class name. The constructor is invoked whenever an object of its associated class is created. It is called constructor because it constructs the value data members of the class. The constructor functions have some special characteristics.

- They should be declared in the public section.
- They are invoked automatically when the objects are created.
- They do not have return types, not even void and therefore, they cannot return values.
- They cannot be inherited, though a derived class can call the base class constructor.

Example:

```
#include< iostream.h>
#include<conio.h>
class myclass {          // class declaration
int a;
public:
myclass ();             //default constructor
void show ();
};
myclass :: myclass() {
cout <<"In constructor\n";
a=10;
}
myclass :: show() {
cout<< a;
}
int main() {
int ob; // automatic call to constructor
ob.show ();
return 0;
}
```

In this simple example the constructor is called when the object is created, and the constructor initializes the private variable a to 10.

Default constructor

The default constructor for any class is the constructor with no arguments. When no arguments are passed, the constructor will assign the values specifically assigned in the body of the constructor. It can be zero or any other value. The default constructor can be identified by the name of the class followed by empty parentheses. Above program uses default constructor. If it is not defined explicitly, then it is automatically defined implicitly by the system.

Parameterized Constructor

It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.

```
#include <iostream.h>
#include<conio.h>
class myclass {
int a, b;
public:
myclass(int i, int j) //parameterized constructor
{a=i; b=j;}
void show() { cout << a << " " << b;}
};
int main() {
myclass ob(3, 5); //call to constructor
ob.show();
return 0;
}
```

C++ supports constructor overloading. A constructor is said to be overloaded when the same constructor with different number of argument and types of arguments initializes an object.

Copy Constructors

The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. If class definition does not explicitly include copy constructor, then the system automatically creates one by default. The copy constructor is used to:

- Initialize one object from another of the same type.
- Copy an object to pass it as an argument to a function.
- Copy an object to return it from a function.

The most common form of copy constructor is shown here:

```
classname (const classname &obj) {
// body of constructor
```

```
}
```

Here, obj is a reference to an object that is being used to initialize another object. The keyword const is used because obj should not be changed.

Destructor

A destructor destroys an object after it is no longer in use. The destructor, like constructor, is a member function with the same name as the class name. But it will be preceded by the character Tilde (~). A destructor takes no arguments and has no return value. Each class has exactly one destructor. If class definition does not explicitly include destructor, then the system automatically creates one by default. It will be invoked implicitly by the compiler upon exit from the program to clean up storage that is no longer accessible.

```
// A Program showing working of constructor and destructor
```

```
#include<iostream.h>
```

```
#include<conio.h>
```

```
class Myclass{
```

```
public:
```

```
int x;
```

```
Myclass(){           //Constructor
```

```
x=10; }
```

```
~Myclass(){         //Destructor
```

```
cout<<"Destructing...";
```

```
}
```

```
int main(){
```

```
Myclass ob1, ob2;
```

```
cout<<ob1.x<<" "<<ob2.x;
```

```
return 0; }
```

Output:

```
10 10
```

```
Destructing.....
```

```
Destructing.....
```

Friend function

In general, only other members of a class have access to the private members of the class. However, it is possible to allow a nonmember function access to the private members of a class by declaring it as a friend of the class. To make a function a friend of a class, you include its prototype in the class declaration and precede it with the friend keyword. The function is declared with friend keyword. But while defining friend function, it does not use either keyword friend or :: operator. A function can be a friend of more than one class. Member function of one class can be friend functions of another class. In such cases they are defined using the scope resolution operator.

A friend, function has following characteristics.

- It is not in the scope of the class to which it has been declared as friend.
- A friend function cannot be called using the object of that class. It can be invoked like a normal function without help of any object.
- It cannot access the member variables directly & has to use an object name dot membership operator with member name.
- It can be declared either in the public or the private part of a class without affecting its meaning.
- Usually, it has the object as arguments.

Program to illustrate use of friend function

```
#include<iostream.h>
#include<conio.h>
class A{
    int x, y;
public:
    friend void display(A &obj);
    void getdata() {
        cin>>x>>y;
    }
};
void display(A &obj){
    cout<<obj.x<<obj.y;
}
int main(){
    A a;
    a.getdata();
    display(a);
    getch();
    return 0;
}
```

Operator overloading

There is another useful methodology in C++ called operator overloading. The language allows not only functions to be overloaded, but also most of the operators, such as +, -, *, /, etc. As the name suggests, here the conventional operators can be programmed to carry out more complex operations. This overloading concept is fundamentally the same i.e. the same operators can be made to perform

different operations depending on the context. Such operators have to be specifically defined and appropriate function programmed. When an operator is overloaded, none of its original meaning is lost. It is simply that a new operation, relative to a specific class, is defined. For example, a class that defines a linked list might use the + operator to add an object to the list. A class that implements a stack might use the + to push an object onto the stack.

An operator function defines the operations that the overloaded operator will perform relative to the class upon which it will work. An operator function is created using the keyword operator. The general form of an operator function is

```
type classname::operator#(arg-list) { // operations
}
```

Here, the operator that you are overloading is substituted for the #, and type is the type of value returned by the specified operation. Operator functions can be either members or nonmembers of a class. Nonmember operator functions are often friend functions of the class.

These operators cannot be overloaded:- ., ::, .*, ?

The process of overloading involves the following steps:

- 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.
- Define the operator function to implement the required operations.

Overloading a unary operator using member function

Overloading a unary operator using a member function, the function takes no parameters. Since, there is only one operand, it is this operand that generates the call to the operator function. There is no need for another parameter.

Overloading unary minus operator

```
#include<iostream.h>
#include<conio.h>
class A {
int x,y,z;
public:
void getdata(int a,int b,int c) {
x=a;
y=b;
z=c;
}
void display() {
cout<<"\nx="<<x<<"\ny="<<y<<"\nz="<<z;
```



```

}
void operator -() //unary minus overload function
{
    x=-x;
    y=-y;
    z=-z;
}
};
int main() {
    A a;
    a.getdata(2,3,4);
    a.display();
    -a;          //activates operator -() function
    a.display();
    getch();
    return 0;
}

```

Overloading binary operator

When a member operator function overloads a binary operator, the function will have only one parameter. This parameter will receive the object that is on the right side of the operator. The object on the left side is the object that generates the call to the operator function and is passed implicitly by this pointer. 'this' can be used in overloading + operator .

```

#include<iostream.h>
#include<conio.h>
class A{
    int x,y;
public:
    void input() {
        cin>>x>>y;
    }
    void display() {
        cout<<"\nx="<<x<<"\ny="<<y<<"\nx+y="<<x+y;
    }
    A operator+(A p);    //overload binary + operator
};

```

```

A A :: operator+(A p) {
    A t;
    t.x=x + p.x;
    t.y=y + p.y;
    return t;
}
int main(){
    A a1, a2, a3;
    a1.input();
    a2.input();
    a3=a2+a1;           //activates operator+() function
    a3.display();
    getch();
    return 0;
}

```

this Pointer

It is facilitated by another interesting concept of C++ called this pointer. 'this' is a C++ keyword. 'this' always refers to an object that has called the member function currently. We can say that 'this' is a pointer. It points to the object that has called this function this time. While overloading binary operators, we use two objects, one that called the operator function and the other, which is passed to the function. We referred to the data member of the calling object, without any prefix. However, the data member of the other object had a prefix. Always 'this' refers to the calling object place of the object name.

Inheritance

Inheritance is the mechanism by which one class can inherit the properties of another. It allows a hierarchy of classes to be build, moving from the most general to the most specific. When one class is inherited by another, the class that is inherited is called the base class. The inheriting class is called the derived class. In general, the process of inheritance begins with the definition of a base class. The base class defines all qualities that will be common to any derived class. In essence, the base class represent the most general description of a set of traits. The derived class inherits those general traits and adds properties that are specific to that class. When one class inherits another, it uses this general form:

```

class derived-class-name : access base-class-name{
// ...
}

```

Here access is one of the three keywords: public, private, or protected. The access specifier determines how elements of the base class are inherited by the derived class.

When the access specifier for the inherited base class is **public**, all public members of the base class become public members of the derived class. If the access specifier is **private**, all public members of the base class become private members of the derived class. In either case, any private members of the base class remain private to it and are inaccessible by the derived class.

It is important to understand that if the access specifier is **private**, public members of the base become private members of the derived class. If the access specifier is not present, it is private by default.

The **protected** access specifier is equivalent to the private specifier with the sole exception that protected members of a base class are accessible to members of any class derived from that base. Outside the base or derived classes, protected members are not accessible. When a protected member of a base class is inherited as public by the derived class, it becomes a protected member of the derived class. If the base class is inherited as private, a protected member of the base becomes a private member of the derived class. A base class can also be inherited as protected by a derived class. When this is the case, public and protected members of the base class become protected members of the derived class (of course, private members of the base remain private to it and are not accessible by the derived class).

Program to illustrate concept of inheritance

```
#include<iostream.h>
#include<conio.h>
class base //base class
{
int x,y;
public:
void show() {
cout<<"In base class";
}
};
class derived : public base //derived class
{
int a,b;
public:
void show2() {
cout<<"\nIn derived class";
}
};
int main() {
derived d;
d.show(); //uses base class's show() function
```

```

d.show2();    //uses derived class's show2() function
getch();
return 0;
}

```

Types of Inheritances

Single Inheritance

The process in which a derived class inherits traits from only one base class, is called single inheritance. In single inheritance, there is only one base class and one derived class. The derived class inherits the behavior and attributes of the base class. However the vice versa is not true. The derived class can add its own properties i.e. data members (variables) and functions. It can extend or use properties of the base class without any modification to the base class. We declare the base class and derived class as given below:

```

class base_class {
};
class derived_class : visibility-mode base_class {
};

```

Program to illustrate concept of single inheritance

```

#include<iostream.h>
#include<conio.h>
class base //base class
{
int x,y;
public:
void show() {
cout<<"In base class";
}
};
class derived : public base //derived class
{
int a,b;
public:
void show2() {
cout<<"\nIn derived class";
}
};

```

```

    }
};
int main() {
    derived d;
    d.show();    //uses base class's show() function
    d.show2();  //uses derived class's show2() function
    getch();
    return 0;
}

```

Ambiguity in single Inheritance

Whenever a data member and member functions are defined with the same name in both the base and derived class, ambiguity occurs. The scope resolution operator must be used to refer to particular class as: object name.class name :: class member

Multiple Inheritance

The process in which a derived class inherits traits from several base classes, is called multiple inheritance. In Multiple inheritance, there is only one derived class and several base classes. We declare the base classes and derived class as given below:

```

class base_class1 {
};
class base_class2 {
};
class derived_class : visibility-mode base_class1 , visibility-mode base_class2 {
};

```

Multilevel Inheritance

The process in which a derived class inherits traits from another derived class, is called Multilevel Inheritance. A derived class with multilevel inheritance is declared as :

```

class base_class {
};
class derived_class1 : visibility-mode base_class {
};
class derived_class 2: visibility-mode derived_class1 {
};

```

Here, derived_class 2 inherits traits from derived_class 1 which itself inherits from base_class.

Hierarchical Inheritance

The process in which traits of one class can be inherited by more than one class is known as Hierarchical inheritance. The base class will include all the features that are common to the derived classes. A derived class can serve as a base class for lower level classes and so on.

Hybrid Inheritance

The inheritance hierarchy that reflects any legal combination of other types of inheritance is known as hybrid Inheritance.

Overriding

Overriding is defined as the ability to change the definition of an inherited method or attribute in a derived class. When multiple functions of the same name exist with different signatures it is called function overloading. When the signatures are the same, they are called function overriding. Function overriding allows a derived class to provide specific implementation of a function that is already provided by a base class. The implementation in the derived class overrides or replaces the implementation in the corresponding base class.

Virtual base classes

A potential problem exists when multiple base classes are directly inherited by a derived class. To understand what this problem is, consider the following class hierarchy:

Here the base class Base is inherited by both Derived1 and Derived2. Derived3 directly inherits both Derived1 and Derived2. However, this implies that Base is actually inherited twice by Derived3. First it is inherited through Derived1, and then again through Derived2. This causes ambiguity when a member of Base is used by Derived3. Since two copies of Base are included in Derived3, is a reference to a member of Base referring to the Base inherited indirectly through Derived1 or to the Base inherited indirectly through Derived2? To resolve this ambiguity, C++ includes a mechanism by which only one copy of Base will be included in Derived3. This feature is called a virtual base class.

In situations like this, in which a derived class indirectly inherits the same base class more than once, it is possible to prevent multiple copies of the base from being present in the derived class by having that base class inherited as virtual by any derived classes. Doing this prevents two or more copies of the base from being present in any subsequent derived class that inherits the base class indirectly. The virtual keyword precedes the base class access specifier when it is inherited by a derived class.

```
// This program uses a virtual base class.
```

```
#include <iostream>
using namespace std;
class Base {
public:
int i;
};
```

```

// Inherit Base as virtual
classDerived1 : virtual publicBase {
public:
int j;
};
// Inherit Base as virtual here, too
classDerived2 : virtual publicBase {
public:
int k;
};
// Here Derived3 inherits both Derived1 and Derived2.
// However, only one copy of base is inherited.
class Derived3 : publicDerived1, publicDerived2 {
public:
int product() { return i*j*k; }
};
int main() {
Derived3 ob;
ob.i = 10; // unambiguous because virtual Base
ob.j = 3;
ob.k = 5;
cout << "Product is: " << ob.product() << "\n";
return 0;
}

```

If Derived1 and Derived2 had not inherited Base as virtual, the statement ob.i=10 would have been ambiguous and a compile-time error would have resulted. It is important to understand that when a base class is inherited as virtual by a derived class, that base class still exists within that derived class.

For example, assuming the preceding program, this fragment is perfectly valid:

```

Derived1 ob;
ob.i = 100;

```

Friend Classes

It is possible for one class to be a friend of another class. When this is the case, the friend class and all of its member functions have access to the private members defined within the other class. For example,

```
// Using a friend class.
#include<iostream.h>
#include<conio.h>
class A{
    int x, y;
public:
    friend void display(A &obj);
    friend class B;
    void getdata() {
        cin>>x>>y;
    }
};
class B{
    int p,q;
public:
    void get(A &obj) {
        p=obj.x;
        q=obj.y;
    }
};
void display(A &obj){
    cout<<obj.x<<obj.y;
}
int main(){
    A a;
    B b;
    b.get(a);
    a.getdata();
    display(a);
    getch();
}
```

```
return 0;  
}
```

It is critical to understand that when one class is a friend of another, it only has access to names defined within the other class. It does not inherit the other class. Specifically, the members of the first class do not become members of the friend class.

IT-03

Unit-IV

Pointer

A pointer is a variable that contains a memory address. Very often this address is the location of another object, such as a variable. For example, if x contains the address of y, then x is said to “point to” y. Pointer variables must be declared as such. The general form of a pointer variable declaration is

```
type *var-name;
```

Here, type is the pointer’s base type. The base type determines what type of data the pointer will be pointing to. var-name is the name of the pointer variable.

To use pointer:

- We define a pointer variable.
- Assign the address of a variable to a pointer.
- Finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand.

Example:

```
int a=10;           //normal variable
int*p;             //declare pointer
p = &a;            // Assign the address of a variable “a” to a pointer “p”
cout<<"a"<<*p;    //prints a=10
```

OBJECT POINTERS

We have been accessing members of an object by using the dot operator. However, it is also possible to access a member of an object via a pointer to that object. When a pointer is used, the arrow operator (->) rather than the dot operator is employed. We can declare an object pointer just as a pointer to any other type of variable is declared. Specify its class name, and then precede the variable name with an asterisk. To obtain the address of an object, precede the object with the & operator, just as you do when taking the address of any other type of variable.

Here is a simple example,

```
#include< iostream>
#include<conio.h>
class myclass {
int a;
public:
myclass(int x); //constructor
int get( );
};
myclass :: myclass(int x) {
```



```

a=x;
}
int myclass :: get() {
return a;
}
int main() {
myclass ob(120); //create object
myclass *p; //create pointer to object
p=&ob; //put address of ob into p
cout <<"value using object: " <<ob.get();
cout <<"\n";
cout <<"value using pointer: " <<p->get();
return 0;
}

```

Notice how the declaration : `myclass *p;` creates a pointer to an object of myclass. It is important to understand that creation of an object pointer does not create an object. It creates just a pointer to one. The address of ob is put into p by using the statement:

```
p=&ob;
```

Finally, the program shows how the members of an object can be accessed through a pointer.

Pointers to Derived Types

Pointers to base classes and derived classes are related in ways that other types of pointers are not. In general, a pointer of one type cannot point to an object of another type. However, base class pointers and derived objects are the exceptions to this rule. In C++, a base class pointer can also be used to point to an object of any class derived from that base. For example, assume that you have a base class called B and a class called D, which is derived from B. Any pointer declared as a pointer to B can also be used to point to an object of type D. Therefore, given

```
B *p; //pointer p to object of type B
```

```
B B_ob; //object of type B
```

```
D D_ob; //object of type D
```

both of the following statements are perfectly valid:

```
p = &B_ob; //p points to object B
```

```
p = &D_ob; //p points to object D, which is an object derived from B
```

A base pointer can be used to access only those parts of a derived object that were inherited from the base class. Thus, in this example, p can be used to access all elements of D_ob inherited from B_ob. However, elements specific to D_ob cannot be accessed through p.

Another point to understand is that although a base pointer can be used to point to a derived object, the reverse is not true. That is, you cannot access an object of the base type by using a derived class pointer.

C++ Virtual Function

A virtual function is a member function that is declared within a base class and redefined by a derived class. In order to make a function virtual, you have to add keyword virtual in front of a function definition. When a class containing a virtual function is inherited, the derived class redefines the virtual function relative to the derived class. The virtual function within the base class defines the form of the interface to that function. Each redefinition of the virtual function by a derived class implements its operation as it relates specifically to the derived class. That is, the redefinition creates a specific method. When a virtual function is redefined by a derived class, the keyword virtual is not needed. A virtual function can be called just like any member function. However, what makes a virtual function interesting, and capable of supporting run-time polymorphism, is what happens when a virtual function is called through a pointer. When a base pointer points to a derived object that contains a virtual function and that virtual function is called through that pointer, C++ determines which version of that function will be executed based upon the type of object being pointed to by the pointer. And this determination is made at run time. Therefore, if two or more different classes are derived from a base class that contains a virtual function, then when different objects are pointed to by a base pointer, different versions of the virtual function are executed.

// A simple example using a virtual function.

```
#include<iostream.h>
#include<conio.h>
class base {
public:
virtual void func( ) {
cout<< "Using base version of func(): ";
}
};
class derived1 : public base {
public:
voidfunc( ) {
cout<< "Using derived1's version of func(): ";
}
};
class derived2 : public base {
public:
voidfunc( ) {
```

```

cout<< "Using derived2's version of func(): ";
}
};

int main() {
base *p;
base ob;
derived1 d_ob1;
derived2 d_ob2;

p = &ob;
p->func(); // use base's func()
p = &d_ob1;
p->func(); // use derived1's func()
p = &d_ob2;
p->func(); // use derived2's func()
return 0;
}

```

Pure virtual functions

Sometimes when a virtual function is declared in the base class, there is no meaningful operation for it to perform. This situation is common because often a base class does not define a complete class by itself. Instead, it simply supplies a core set of member functions and variables to which the derived class supplies the remainder. When there is no meaningful action for a base class virtual function to perform, the implication is that any derived class must override this function. To ensure that this will occur, C++ supports pure virtual functions. A pure virtual function has no definition relative to the base class. Only the function prototype is included. To make a pure virtual function, use this general form:

```
virtual type func-name(parameter-list) = 0;
```

The key part of this declaration is the setting of the function equal to 0. This tells the compiler that no body exists for this function relative to the base class. When a virtual function is made pure, it forces any derived class to override it. If a derived class does not, a compile-time error results. Thus, making a virtual function pure is a way to guaranty that a derived class will provide its own redefinition.

Abstract class

When a class contains atleast one pure virtual function, it is referred to as an abstract class. Since, an abstract class contains atleast one function for which no body exists, it is, technically, an incomplete type, and no objects of that class can be created. Thus, abstract classes exist only to be inherited. It is important to understand, however, that you can still create a pointer to an abstract class, since it is through the use of base class pointers that run-time polymorphism is achieved. (It is also possible to have a reference to an abstract class.) .

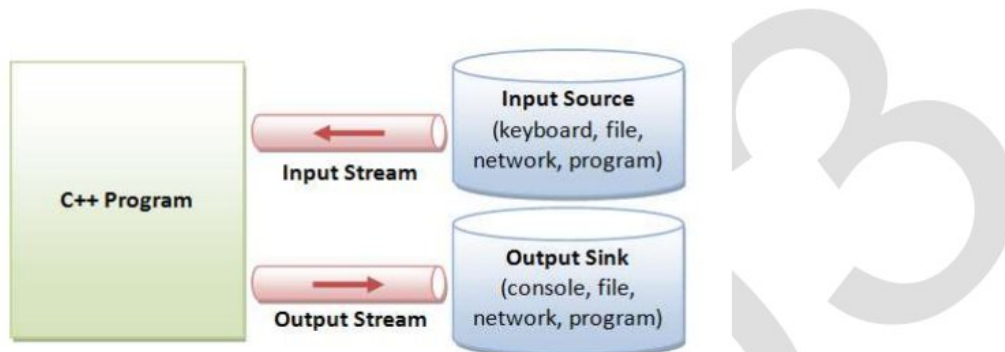
C++ Streams

The C++ I/O system operates through streams. A stream is logical device that either produces or consumes information. A stream is linked to a physical device by the C++ I/O system. All streams behave in the same manner, even if the actual physical devices they are linked to differ. Because all streams act the same, the I/O system presents the programmer with a consistent interface.

Two types of streams:

Output stream: a stream that takes data from the program and sends (writes) it to destination.

Input stream: a stream that extracts (reads) data from the source and sends it to the program.



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.

C++ provides supports for its I/O system in the header file <iostream>. Just as there are different kinds of I/O (for example, input, output, and file access), there are different classes depending on the type of I/O. The following are the most important stream classes:

Class istream :- Defines input streams that can be used to carry out formatted and unformatted input operations. It contains the overloaded extraction (>>) operator functions. Declares input functions such get(), getline() and read().

Class ostream :- Defines output streams that can be used to write data. Declares output functions put and write(). The ostream class contains the overloaded insertion (<<) operator function

When a C++ program begins, these four streams are automatically opened:

Stream	Meaning	Default Device
cin	Standard input	Keyboard
cout	Standard output	Screen
cerr	Standard error	Screen
clog	Buffer version of cerr	Screen

Cin and Cout objects

cout is an object of class ostream. The cout is a predefined object that represents the standard output stream in C++. Here, the standard output stream represents monitor. In the language of C++, the << operator is referred to as the insertion operator because it inserts data into a stream. It inserts or sends the contents of variable on its right to the object on its left.

For example:

```
cout << "Programming in C++";
```

Here << operator is called the stream insertion operator and is used to push data into a stream (in this case the standard output stream)

cin is an object of class istream. cin is a predefined object that corresponds to the standard input stream. The standard input stream represents keyboard. The >> operator is called the extraction operator because it extracts data from a stream. It extracts or takes the value from the keyboard and assigns it to the variable on its right.

For example:

```
int number;  
cin >> number;
```

Here >> operator accepts value from keyboard and stores in variable number.

Unformatted Input/Output Functions

Functions get and put

The get function receives one character at a time. There are two prototypes available in C++ for get as given below:

```
get (char *)  
get ()
```

Their usage will be clear from the example below:

```
char ch ;  
cin.get (ch);
```

In the above, a single character typed on the keyboard will be received and stored in the character variable ch.

Let us now implement the get function using the other prototype:

```
char ch ;  
ch = cin.get();
```

This is the difference in usage of the two prototypes of get functions.

The complement of get function for output is the put function of the ostream class. It also has two forms as given below:


```
cout.put (var);
```

Here the value of the variable var will be displayed in the console monitor. We can also display a specific character directly as given below:

```
cout.put ('a');
```

getline and write functions

C++ supports functions to read and write a line at one go. The getline() function will read one line at a time. The end of the line is recognized by a new line character, which is generated by pressing the Enter key. We can also specify the size of the line.

The prototype of the getline function is given below:

```
cin.getline (var, size);
```

When we invoke the above statement, the system will read a line of characters contained in variable var one at a time. The reading will stop when it encounters a new line character or when the required number (size-1) of characters have been read, whichever occurs earlier. The new line character will be received when we enter a line of size less than specified and press the Enter key. The Enter key or Return key generates a new line character. This character will be read by the function but converted into a NULL character and appended to the line of characters.

Similarly, the write function displays a line of given size. The prototype of the write function is given below:

```
write (var, size) ;
```

where var is the name of the string and size is an integer.

Formatted I/O via manipulators

The C++ I/O system allows you to format I/O operations. For example, you can set a field width, specify a number base, or determine how many digits after the decimal point will be displayed. I/O manipulators are special I/O format functions that can occur within an I/O statement.

Manipulator	Purpose	Input/Output
boolalpha	Turns on boolalphaflag	Input/Output
dec	Turns on decflag	Input/Output
endl	Outputs a newline character and flushes the stream	Output
ends	Outputs a null	Output
fixed	Turns on fixed flag	Output
flush	Flushes a stream	Output
hex	Turns on hexflag	Input/Output
internal	Turns on internalflag	Output
left	Turns on leftflag	Output
noboolalpha	Turns off boolalphaflag	Input/Output

noshowbase	Turns off showbaseflag	Output
noshowpoint	Turns off showpointflag	Output
noshowpos	Turns off showposflag	Output
noskipws	Turns off skipwsflag	Input
nounitbuf	Turns off unitbufflag	Output
nouppercase	Turns off uppercaseflag	Output
oct	Turns on octflag	Input/Output
resetiosflags(fmtflads f)	Turns off the flags specified in f	Input/Output
right	Turns on rightflag	Output
scientific	Turns on scientificflag	Output
setbase(int base)	Sets the number base to base	Input/Output
setfill(int ch)	Sets the fill char ch	Output
setiosflags(fmtflags f)	Turns on the flags specified by f	Input/Output
setprecision(int p)	Sets the number of digits of precision	Output
setw(int w)	Sets the field width to w	Output
showbase	Turns on showbaseflag	Output
showpoint	Turns on showpointflag	Output
showpos	Turns on showposflag	Output
skipws	Turns on skipwsflag	Input
unitbuf	Turns on unitbuf	Output
uppercase	Turns on uppercaseflag	Output
ws	Skips leading white space	Input

The following program demonstrates several of the I/O manipulators:

```
#include<iostream>
#include<iomanip>
using namespacestd;
int main() {
cout<< hex << 100 << endl;
cout<< oct<< 10 << endl;
cout<< setfill('X') << setw(10);
cout<< 100 << " hi " << endl;
return0;
```

```
}
```

This program displays the following:

```
64
```

```
13
```

```
XXXXXXXX144 hi
```

File I/O

A file is a bunch of bytes stored on some storage devices like hard disk, floppy disk etc. File I/O and console I/O are closely related. In fact, the same class hierarchy that supports console I/O also supports the file I/O. To perform file I/O, you must include `<fstream>` in your program. It defines several classes, including `ifstream`, `ofstream` and `fstream`. In C++, a file is opened by linking it to a stream. There are three types of streams: input, output and input/output. Before you can open a file, you must first obtain a stream.

To create an input stream, declare an object of type `ifstream`.

To create an output stream, declare an object of type `ofstream`.

To create an input/output stream, declare an object of type `fstream`.

For example, this fragment creates one input stream, one output stream and one stream capable of both input and output:

```
ifstream in; // input;
```

```
ofstream out; // output;
```

```
fstream io; // input and output
```

Once you have created a stream, one way to associate it with a file is by using the function `open()`. This function is a member function of each of the three stream classes. The prototype for each is shown here:

```
void ifstream::open(const char*filename,openmode mode=ios::in);
```

```
void ofstream::open(const char*filename,openmode mode=ios::out | ios::trunc);
```

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

Here `filename` is the name of the file, which can include a path specifier. The value of the mode determines how the file is opened. It must be a value of type `open mode`, which is an enumeration defined by `ios` that contains the following value:

- `ios::app`: causes all output to that file to be appended to the end. Only with files capable of output.
- `ios::ate`: causes a seek to the end of the file to occur when the file is opened.
- `ios::out`: specify that the file is capable of output.
- `ios::in`: specify that the file is capable of input.
- `ios::binary`: causes the file to be opened in binary mode. By default, all files are opened in text mode. In text mode, various character translations might take place, such as carriage return/linefeed

sequences being converted into newlines. However, when a file is opened in binary mode, no such character translations will occur.

- `ios::trunc`: causes the contents of a pre-existing file by the same name to be destroyed and the file to be truncated to zero length. When you create an output stream using `ofstream`, any pre-existing file is automatically truncated.

All these flags can be combined using the bitwise operator OR (`|`). For example, if we want to open the file `example.bin` in binary mode to add data we could do it by the following call to member function `open`:

```
ofstream myfile;
```

```
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

Each of the `open` member functions of classes `ofstream`, `ifstream` and `fstream` has a default mode that is used if the file is opened without a second argument:

Class	default mode parameter
<code>ofstream</code>	<code>ios::out</code>
<code>ifstream</code>	<code>ios::in</code>
<code>fstream</code>	<code>ios::in ios::out</code>

Closing file

When we are finished with our input and output operations on a file we shall close it so that the operating system is notified and its resources become available again. For that, we call the stream's member function `close`. This member function takes flushes the associated buffers and closes the file:

```
myfile.close();
```

Once this member function is called, the stream object can be re-used to open another file, and the file is available again to be opened by other processes. In case that an object is destroyed while still associated with an open file, the destructor automatically calls the member function `close`.

To write to a file, you construct a `ofstream` 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()`.

There are two ways of storing data in a file as given below:

Binary form and Text form

Suppose, we want to store a five digit number say 19876 in the text form, then it will be stored as five characters. Each character occupies one byte, which means that we will require five bytes to store five-digit integer in the text form. This requires storage of 40 bits. Therefore, if we can store them in binary form, then we will need only two bytes or 16 bits to store the number. The savings will be much more when we deal with floating point numbers.

When we store a number in text form, we convert the number to characters. However, storing a number in binary form requires storing it in bits. However, for a character, the binary representation as well as the text representation are one and the same since, in either case, it occupies eight bits. The text format is easy to read. We can even use a notepad to read and edit a text file. The portability of

text file is also assured. In case of numbers, binary form is more appropriate. It also occupies lesser space when we store it in binary form and hence it will be faster. The default mode is text.

Unformatted, binary I/O

C++ supports a wide range of unformatted file I/O functions. The unformatted functions give you detailed control over how files are written and read. The lowest-level unformatted I/O functions are `get()` and `put()`. You can read a byte by using `get()` and write a byte by using `put()`. These functions are member functions of all input and output stream classes, respectively. The `get()` function has many forms, but the most commonly used version is shown here, along with `put()`:

```
istream &get(char&ch);
```

```
ostream &put(char&ch);
```

To read and write blocks of data, use `read()` and `write()` functions, which are also member functions of the input and output stream classes, respectively. Their prototypes are:

```
istream &read(char*buf, streamsize num);
```

```
ostream &write(const char*buf, streamsize num);
```

The `read()` function reads `num` bytes from the stream and puts them in the buffer pointed to by `buf`. The `write()` function writes `num` bytes to the associated stream from the buffer pointed by `buf`. The `streamsize` type is some form of integer. An object of type `streamsize` is capable of holding the largest number of bytes that will be transferred in any I/O operation. If the end of file is reached before `num` characters have been read, `read()` stops and the buffer contains as many characters as were available.

When you are using the unformatted file functions, most often you will open a file for binary rather than text operations. The reason for this is easy to understand: specifying `ios::binary` prevents any character translations from occurring. This is important when the binary representations of data such as integers, floats and pointers are stored in the file. However, it is perfectly acceptable to use the unformatted functions on a file opened in text mode, as long as that the file actually contains only text. But remember, some character translation may occur.

Random Access

File pointers

C++ also supports file pointers. A file pointer points to a data element such as character in the file. The pointers are helpful in lower level operations in files. There are two types of pointers:

get pointer

put pointer

The get pointer is also called input pointer. When we open a file for reading, we can use the get pointer. The put pointer is also called output pointer. When we open a file for writing, we can use put pointer. These pointers are helpful in navigation through a file. When we open a file for reading, the get pointer will be at location zero and not 1. The bytes in the file are numbered from zero. Therefore, automatically when we assign an object to `ifstream` and then initialize the object with a file name, the get pointer will be ready to read the contents from 0th position. Similarly, when we want to write we will assign to an `ofstream` object a filename. Then, the put pointer will point to the 0th position of the given file name after it is created. When we open a file for appending, the put pointer will point to the

0th position. But, when we say write, then the pointer will advance to one position after the last character in the file.

File pointer functions

There are essentially four functions, which help us to navigate the file as given below Functions

Function	Purpose
----------	---------

tellg()	Returns the current position of the get pointer
seekg()	Moves the get pointer to the specified location
tellp()	Returns the current position of the put pointer
seekp()	Moves the put pointer to the specified location

```
//To demonstrate writing and reading- using open
```

```
#include<fstream.>
```

```
#include<iostream>
```

```
int main(){ //Writing
```

```
ofstream outf;
```

```
outf.open("Temp2.txt");
```

```
outf<<"Working with files is fun\n";
```

```
outf<<"Writing to files is also fun\n";
```

```
outf.close();
```

```
char buff[80];
```

```
ifstream inf;
```

```
inf.open("Temp2.txt"); //Reading
```

```
while(inf){
```

```
inf.getline(buff, 80);
```

```
cout<<buff<<"\n";
```

```
}
```

```
inf.close();
```

```
return 0;
```

```
}
```

(20518)

Roll No. 16.3.2.21.8.37

B. Sc. (Com. Sc.)-IV Sem.

NP-3607

B. Sc. (Com. Science) Examination, May 2018

OOP Through C++

(BCS-403)

Time : Three Hours]

[Maximum Marks : 75

Note : Attempt questions from all Sections as per instructions.

Section-A

(Very Short Answer Questions)

Answer all the *five* questions. Each question carries 3 marks. Very short answer is required. $3 \times 5 = 15$

1. What do you understand by object oriented programming?

(2)

2. What is token ?
3. What do you understand by control structure ?
4. What is user defined data type ? Give one example.
5. What is polymorphism ?

Section-B

(Short Answer Questions)

Answer any *two* questions out of the following three questions. Each question carries $7\frac{1}{2}$ marks. Short answer is required. $7\frac{1}{2} \times 2 = 15$

6. What do you understand by class ? Explain with the suitable program using multiple inheritance.
7. What is abstract classes ? Explain with the suitable example.
8. What do you understand by static member function? Explain with the suitable example.

NP-3607

(3)

Section-C

(Detailed Answer Questions)

Answer any *three* questions out of the following five questions. Each question carries 15 marks. Answer is required in detail. $15 \times 3 = 45$

9. What is constructor ? Explain with the suitable example using parameterized constructor.
10. What do you understand by hybrid inheritance ? Explain with the suitable example.
11. What do you understand by pure virtual function ?
12. Write down the rules for operator overloading.
13. What is destructor ? Explain with the suitable example.

NP-3607-3-

N

(Printed Pages 3)

(20517)

Roll No. 1532807045

B.Sc. (Com.Sc.)-IV Sem.

NP-3607

B.Sc. Examination, May 2017

Computer Science

OOP Through C++

(BCS-403)

Time : Three Hours]

[Maximum Marks : 75

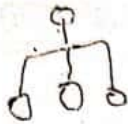
Note: Attempt **all** the sections as per instructions.

Section-A

Note : Attempt **all five** questions. Each question carries **three** marks. Very short answer is required not exceeding 75 words. $3 \times 5 = 15$

1. What is difference b/w Object and Classes?
2. Describe the inheritance as applied to OOP.

P.T.O.

- 
3. Define the term 'Name space'.
 4. What are the advantage of using new Operator and compared to function malloc ()?
 5. When do we make a virtual function 'pure'?

Section-B

Note: Attempt **any two** questions out of the following 3 questions. Each question carries 7.5 marks. short answer is required not exceeding 200 words.

6. What is destructor? Give an example in C++ to illustrate destructor?
7. What is friend function? What are the merits and demerits of using friend functions.
8. (a) What is virtual Base Class
(b) Describe an abstract Classes

Section-C

Note: Attempt **any three (3)** questions out of the following five **(5)** questions. Each

NP-360712

question carries 15 marks. Answer is required in detail.

9. Explain the use of the Multiple inheritance and multilevel inheritance with help of the suitable programming example?
10. What are the constructor and explain all the types of the constructor with help of example.
11. What is Base and Drive Classes? Explain it with the help of suitable example.
12. What is operator overloading Explain in detail about the unary operator overloading with example program.
13. What is visibility mode? What are the difference between inheriting a class with public and private visibility mode?

NP-360713

Section - C

(Detailed Answer Questions)

Note: Attempt any three questions out of the following 5 questions. Each question carries 15 marks. Answer required in detail.

9. (a) What is meant by central tendency? What are the different measures of central tendency? Write down the properties of Ideal average. (7.5)
- (b) Use false position method to find the root of $x.e^x = \cos x$ correct to four decimal places. (7.5)
- (c) Given $\frac{dy}{dx} = (x^2 + 1)e^{-x}$, $y(0) = 0$, $h = 0.1$. Evaluate y in the interval $0.0 \leq x \leq 1.0$ by Runge-Kutta method. (7.5)
- (d) What do you understand by skewness? How is it measured? Distinguish clearly.

NP-360526014

Shivam Chandra

(Printed Pages)

(20612)

Roll No. 6193073

B.Sc. (Com. Sc.) - IV Sem

NP-3605

B.Sc. (Com. Sc.) Examination

June 2012

Computer Oriented Numerical Techniques

(BCS - 401)

Time: Three Hours

(Maximum Marks)

Note: Attempt all questions as per given directions. Students can use calculator.

Section - A

(Very Short Answer Questions)

Note: Attempt all five questions. Each question carries three marks. Very short answer is required not exceeding 75 words.

1. Discuss Chi-square test for the test of significance. (3)
2. Evaluate relative error of the function $f = xy^2z$, if $x = 37.1$, $y = 9.87$, $z = 6.052$ and $\Delta x = 0.3$, $\Delta y = 0.11$, $\Delta z = 0.016$. (3)
3. Discuss the Relation between divided differences and ordinary differences. (3)
4. Evaluate $\int_0^1 \frac{dx}{1+x^2}$ using Boole's rule. (3)
5. Define Correlation and Regression. (3)

Section - B

(Short Answer Questions)

Note: Attempt any two questions out of the following 3 questions. Each question carries 5 marks. Short answer questions required not exceeding 200 words.

1. Use Newton-Raphson method to find a positive root of the equation. (5)

NP-360526012

1. Find the value of $(1.06)^{19}$ if correct to four decimal places. (3)
2. Find the value of $(1.06)^{10}$, $(1.06)^{15}$, $(1.06)^{20}$, $(1.06)^{25}$, $(1.06)^{30}$. (5)
3. Given the table of the values of the function $y = \log x$. (5)

x	y
1000	3.000
1010	3.0043214
1020	3.0086002
1030	3.0128372
1040	3.0170333
1050	3.0211893

Find $\log 1001$.

NP-360526013

P.T.O.

III (b) Evaluate $\int_0^1 \frac{dx}{x^2}$ using Simpson's one-third rule to find the approximate value of $\log 2$.
 12. Find $y(2)$ if $y(x)$ is the solution of $\frac{dy}{dx} = \frac{1}{2}(x+y)$ assuming $y(0)=2, y(0.5)=2.635, y(1.0)=3.595$ and $y(1.5)=4.968$.
 II 13. (a) State and prove Lagrange's formula for interpolation.
 (b) In an examination, the number of candidates who obtained marks between certain limits were as follows:

Marks	No. of candidates
0-19	41
20-39	62
40-59	65
60-79	50
80-99	17

Estimate the number of candidates who obtained fewer than 70 marks.

NP-3605/40014

M (Printed Pages 4)
 (20614) Roll No. 3284031
 B.Sc. (Com.Sc.) IV Sem.
NP-3605
 B.Sc. (Com.Sc.) Examination, June 2014
Computer Oriented Numerical Techniques
(BCS-401)

Time : Three Hours] [Maximum Marks : 75

Note: Attempt all questions as per given directions. Students can use calculator.

Section - A

(Very short answer questions)

Note: Attempt all five questions. Each question carries three marks. Very short answer is required not exceeding 75 words.

1. Find the relative error in computation of $x+y$ for $x=11.75$ and $y=7.23$ having absolute errors $\Delta x = 0.002$ and $\Delta y = 0.005$.

Define operators E and Δ and show that $E = I + \Delta$.

Construct a difference table for the

P.T.O.

divided off

following:

x :	1,	7	12
f(x) :	22	30	82 106 216

4. Show that the co-efficient of correlation between two variates x and y may be expressed in the form $\frac{1}{\sigma_x \sigma_y} \left(\frac{1}{N} \sum xy - \bar{x} \cdot \bar{y} \right)$

Where \bar{x}, \bar{y} are A. Ms and σ_x, σ_y are standard deviations.

5. Differentiate between Trapezoidal rule and Simpson's rule for numerical integration.

Section - B

Note: Attempt any two questions out of the following 3 questions. Each question carries 7.5 marks.

6. From the table given below, compute $\frac{dy}{dx}$

and $\frac{d^2y}{dx^2}$ for $x = 1$

x	1,	2	4	5	6
y	1	8	64	125	216

7. Solve $\frac{dy}{dx}$ by Picard's method

NP-360540012

$x \quad y(0) = 1$

and compare with the exact solution.

8. Find the least square line for the following data points :
(-2, 1), (0, 3), (2, 6), (4, 8) and (6, 13).

Section - C

Note: Attempt any three questions out of the following five questions. Each question carries 15 marks.

9. (a) Describe Muller's method to find roots.
(b) The Newton-Raphson's method to find the root of $x^2 - x - 10 = 0$ which is nearer to 2, correct to three places of decimal.
10. (a) If $\log_{10} x$ is normally distributed with mean 4 and variance 4, find the probability of $1.202 < x < 83180000$. (Given $\log_{10} 1202 = 3.08$ and $\log_{10} 8318 = 3.92$)
(b) Describe chi-square test of significance.

11. (a) Consider the following data:
 $\bar{x} = 36, \bar{y} = 85, \sigma_x = 11, \sigma_y = 9,$
 $r = 0.7$. Find two regression equations and estimate the value of y when $x = 30$.

NP-360540013

P.T.O.

13. (a) The following data are number of seeds germinating out of 10 on damp filter paper for 80 sets of seeds. Fit a binomial distribution to these data:

x	0	1	2	3	4	5	6	7	8	9	10
f	6	20	28	12	8	6	0	0	0	0	0

(b) Explain why there are two lines of regression. Write down two regression equations that may be associated with the following pairs of values:

x	152	114	128	154	144	153	141	117	136
y	193	300	414	594	676	549	320	483	481

NP-360516

15

20

Harsh
Sharma

N

(Printed Pages 6)

(20517)

Roll No. 1532807045

B.Sc. (Com.Sc.) IV Sem.

NP-3605

B.Sc. (Com. Sc.) Examination, May 2017

Computer Oriented Numerical

Techniques

(BCS-401)

Time : Three Hours]

[Maximum Marks : 75

Note : Attempt all questions as per given direc-

tions. Students can use calculator.

Section - A

(Very Short Answer Questions)

Note : Attempt all five questions. Each question car-

ries three marks. Very short answer is re-

quired not exceeding 75 words. $3 \times 5 = 15$

P.T.O.

1. Find the quotient $q = x/y$, where $x = 4.536$ and $y = 1.32$, both x and y being correct to the digits given. Find also the relative error in the result.

$$\frac{\Delta x}{y} =$$

2. Show $\Delta^r y_k = \nabla^r y_{k+r}$

3. Prove that, If $f(x)$ be a polynomial of n^{th} degree in x , then the n^{th} difference of $f(x)$ is constant and $\Delta^{n+1} f(x) = 0$.

4. Calculate the value of the integral

$$\int_4^{5.2} \log x \, dx \text{ by Simpson's } \frac{3}{8} \text{ rule. } m$$

5. The means of two sample of size 50 and 100 respectively are 54.1 and 50.3; the standard deviations are 8,7. Find the mean and standard deviation of the sample of size 150 by combining the two samples.

$$\log x \, dx$$

$$\log 4$$

NP-360512

5:2

Section - B

Note : Attempt any **two** questions out of the following **three** questions. $7.5 \times 2 = 15$

6. Find the value of $y(1.1)$ using Runge - Kutta method of fourth order given that

$$\frac{dy}{dx} = y^2 + xy, \quad y(1) = 1.0, \quad \text{take } h = 0.05.$$

15

7. Find the root of the equation

$$2x = \cos x + 3,$$

1.95
1.1

correct to three decimal places by using iteration method.

8. Using Lagrange's interpolation formula, find the form of the function $y(x)$ from the following table.

x	0	1	3	4
y	-12	0	12	24

$f(x_0)$ $f(x_1)$ $f(x_2)$ $f(x_3)$

NP-360513

P.T.O.

(6)

Father	Son
65	68
63	66
67	68
64	65
68	69
62	66
70	68
66	65
68	71
67	67
69	68
71	70

Calculate coefficient of rank correlation.

NP-3605-6-

(20518)

Roll No. 163291932

B. Sc.(Com. Sc.)-IV Sem.

NP-3605

B. Sc. (Com. Science) Examination, May 2018

Computer Oriented Numerical Techniques

(BCS-401)

Time : Three Hours]

[Maximum Marks : 75

Note : Attempt questions from all Sections as per instructions. Students can use calculator.

Section-A

(Very Short Answer Questions)

Attempt all the *five* questions. Each question carries 3 marks. Very short answer is required not exceeding 75 words. $3 \times 5 = 15$

1. If $u = 3v^7 - 6v$, find the percentage error in u at $v=1$, if the error in v is 0.05.
2. Define the operators Δ , ∇ , δ and E^{-1} and evaluate $\Delta^2 x^3$.

(2)

3. Apply Lagrange's formula to find the form of the function given by :

x	0	1	2
$f(x)$	3	6	10

4. Evaluate $\int_4^{4.6} \log_e(x) dx$ by Simpson's $\frac{1}{3}$ rule, using the data $\log_e 4.0 = 1.38$, $\log_e 4.2 = 1.43$, $\log_e 4.4 = 1.48$, $\log_e 4.6 = 1.52$.

5. A random variable X has the following probability function :

x	-2	-1	0	1	2	3
$p(x)$	0.1	k	0.2	$2k$	0.3	k

find ' k ' .

Section-B

(Short Answer Questions)

Attempt any *two* questions out of the following three questions. Each question carries $7\frac{1}{2}$ marks. Short answer is required. $7\frac{1}{2} \times 2 = 15$

NP-3605

(3)

6. Use the Runge-Kutta fourth order method to solve $10 \frac{dy}{dx} = x^2 + y^2$, $y(0) = 1$ for the interval $0 < x \leq 0.4$ with $h = 0.1$.

7. From the following table find y when $x = 0.15$:

x	0.1	0.2	0.3	0.4	0.5	0.6
y	2	4	5	7	8	10

8. Use the iteration method to find a real root of the following equation correct up to three decimal places:

$$e^x = \cot x.$$

Section-C

(Detailed Answer Questions)

Answer any *three* questions out of the following five questions. Each question carries 15 marks. Answer is required in detail. $15 \times 3 = 45$

9. (a) 200 digits were chosen at random from a set of tables. The frequencies of the digits were :

Digits	0	1	2	3	4	5	6	7	8	9
Frequency	18	19	23	21	16	25	22	20	21	15

NP-3605

(4)

Use χ^2 test to assess the correctness of hypothesis that the digits were distributed in equal numbers in the table, given that the value of χ^2 are respectively 16.9, 18.3 and 19.7 for 9, 10 and 11 degree of freedom at 5% level of significance.

(b) Use Muller's method to find a root of the equation $x^3 - x^2 - x + 1 = 0$.

10. (a) Given the set of tabulated points (1, -3), (3, 9), (4, 30), (6, 40), obtain the value of y when $x=2$ using Newton's divided-difference formula.

(b) Use Picard's method to approximate the value of y when $x=0.1$, given that $y=1$ when $x=0$ and $\frac{dy}{dx} = 3x + y^2$.

11. (a) The curve $y = ce^{bx}$ is fitted to the data :

x	1	2	3	4	5
y	1.5	4.6	13.9	40	90

Find the best value of c and b .

505

(5)

(b) Use Stirling's formula to find u_{32} from the following data :

$$u_{20} = 14, \quad u_{25} = 18, \quad u_{30} = 22, \\ u_{35} = 24, \quad u_{40} = 12, \quad u_{45} = 32.$$

12. (a) Calculate $\int_0^{\pi/2} e^{\sin x} dx$ correct to four decimal places using Simpson's $\frac{3}{8}$ rule.

(b) Obtain $\sqrt{12}$ to five places of decimal by Newton-Raphson method.

13. (a) Find the first derivative of $f(x)$ at $x=0.4$ from the following table :

x	0.1	0.2	0.3	0.4
$f(x)$	1.10517	1.22140	1.34986	1.49182

(b) A sample of 12 fathers and their eldest sons gave the following data about their height in inches :

NP-3605

OOP with C++ by E Balagurusamy > Solution

Chapter 1

Review Questions

1.1: What do you think are the major issues facing the software industry today?

Ans: Many software products are either not finished or not used or not delivered for some major errors. Today some of the quality issues that must be considered for software industry are:

1. Correctness.
2. Maintainability.
3. Reusability.
4. Openness and interoperability.
5. Portability.
6. Security.
7. Integrity.
8. User friendliness.

1.2: Briefly discuss the software evolution during the period 1950— 1990.

Ans: In 1950, the first three modern programming languages whose descendants are still in widespread today.

1. FORTRAN (1955), the “FORMula TRANslator”
2. LISP (1958) the “LIST Procссор”.
3. COBOL, the COMmon Business Oriented Language.

Some important language that were developed in 1950 to 1960 are:

1. Regional Assembly Language–1951
2. Autocode–1952
3. IPL–1954
4. FLOW-MATIC–1955
5. COMTRAN–1957
6. COBOL–1959
7. APL–1962

1.3: What is procedure-oriented programming? What are its main characteristics?

Ans: Conventional programming, using high level language such as COBOL, FORTAN and C is commonly known as procedure oriented programming.

Characteristics :

- a) Emphasis is on doing things (algorithms)
- b) Large programs are divided into small programs known as function.
- c) Most of the function share global data.
- d) Data move openly around the system from function to function.
- e) Function transform data from one form to another.

1.4: Discuss an approach to the development of procedure-oriented programs.

Ans: In the procedure-oriented approach, the problem is viewed as a sequence of things to be done such as

- 1. reading
- 2. Calculating
- 3. Printing.

A number of functions are written to accomplish these tasks.

1.5: Describe how data are shared by functions in a procedure-oriented program.

Ans: In a multi-function program, many important data items are placed as global so that they may be accessed by all the functions. Each function may have its own local data.

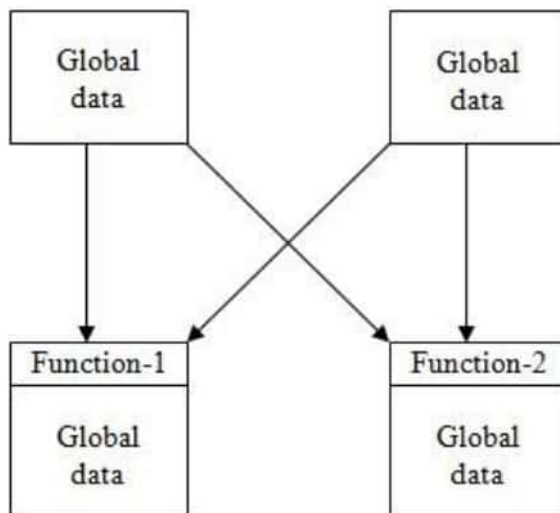


Fig: Data sharing in procedure oriented program

1.6: What is object-oriented programming? How is it different from the procedure-oriented programming?

Ans: Object oriented programming (OOP) is 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 creation copies of such modules on demand.

Different between OOP(Object oriented programming) & POP(Procedure oriented programming):

1. OOP has data hiding feature for which the data of a class cannot be accessed by the member function of other class but POP has no such feature.
2. In OOP we can design our own data-type which is same as built in data type. But in POP we can not do this.

1.7: How are data and functions organized in an object-oriented program?

Ans: Data and functions are belongs to a class. Data is called data member and functions are called member functions. There is a visibility-mode such as public and private. Generally data is private and functions are public.

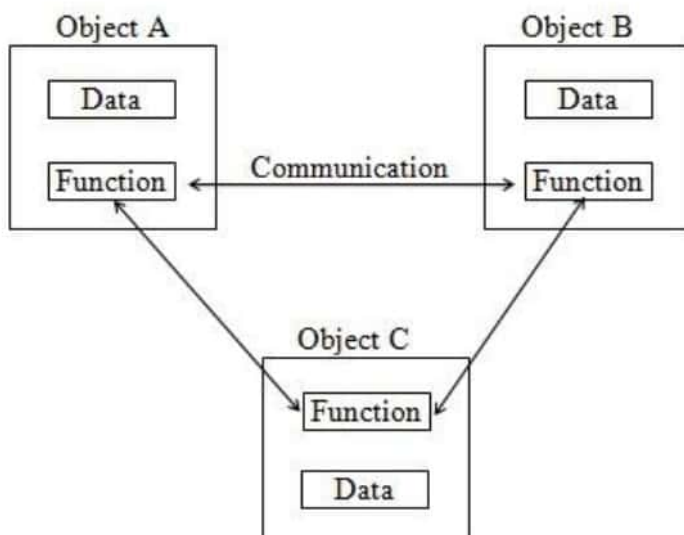


Fig : Organization of data and function in OOP

1.8: What are the unique advantages of an object-oriented programming paradigm?

Ans: The unique advantage of object-oriented program paradigm is to have a working definition of OOP before we proceed further.

1.9: Distinguish between the following terms:

- (a) Objects and classes
- (b) Data abstraction and data encapsulation
- (c) Inheritance and polymorphism
- (d) Dynamic binding and message passing

Ans: (a) Objects are the basic run-time entities which contain data and code to manipulate data where the entire set of data and code of an object can be made as a user-defined data type with the help of a class. In short, objects are members of classes.

(b) Describing the functionality of a class independent of its implementation is called data abstraction. Where data encapsulation means the wrapping up of data and functions into a single unit.

(c) The mechanism of deriving a new class from an old one is called inheritance, where polymorphism means one thing with several distinct terms.

(d) Binding refers to the linking of a procedure call to be executed in response to the call. When binding occurs at run-time, then it is known as dynamic-binding.

Message passing involves specifying the name of the object, the name of the function and the information to be sent

1.10: What kinds of things can become objects in OOP?

Ans: 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 has to handle.

1.11: Describe inheritance as applied to OOP.

Ans: Inheritance is one of the most powerful features of object-oriented programming. Inheritance is the process of creating a new class from an existing class. The new class is called a derived class and the existing class is called a base class.

1.12: What do you mean by dynamic binding? How is it useful in OOP?

Ans: Binding refers to the linking of a procedure call to be executed in response to the call. When binding occurs at run time it is known as dynamic binding. Dynamic binding is useful in OOP such as a function call associated with a polymorphic reference depends on the dynamic type of that reference.

1.13: Now does object-oriented approach differ from object-based approach?

Ans: Object-based programming do not support inheritance and dynamic binding but object-oriented programming do so.

1.14: List a few areas of application of OOP technology.

Ans: Areas of application of OOP technology are :

1. Real-time system.
2. Simulation and modeling.
3. Object oriented database.
4. Hypertext, hypermedia .
5. Decision support and office automation system.

1.15: State whether the following statements are TRUE or FALSE.

- (a) In procedure-oriented programming, all data are shared by all functions.
- (b) The main emphasis of procedure-oriented programming is on algorithms rather than on data.
- (c) One of the striking features of object-oriented programming is the division of programs into objects that represent real-world entities.
- (d) Wrapping up of data of different types into a single unit is known as encapsulation.
- (e) One problem with OOP is that once a class is created it can never be changed.
- (f) Inheritance means the ability to reuse the data values of one object by
- (g) Polymorphism is extensively used in implementing inheritance.
- (h) Object oriented programs are executed much faster than conventional programs.
- (i) Object-oriented systems can scale up better from small to large.
- (j) Object-oriented approach cannot be used to create databases.

Ans:

- a> FALSE
- b> TRUE
- c> TRUE
- d> FALSE
- e> FALSE
- f> TRUE
- g> TRUE
- h> FALSE
- i> TRUE
- j> FALSE

Chapter 2

Review Questions

2.1: State whether the following statements are TRUE or FALSE.

- (a) Since C is a subset of C++, all C programs will run under C++ compilers.
- (b) In C++, a function contained within a class is called a member function.
- (c) Looking at one or two lines of code, we can easily recognize whether a program is written in C or C++.
- (d) In C++, it is very easy to add new features to the existing structure of an object.
- (e) The concept of using one operator for different purposes is known as aerator overloading. 10
The output function printf() cannot be used in C++ programs.

Ans:

- a> FALSE
- b> TRUE
- c> FALSE
- *** most lines of codes are the same in C & C++
- d> TRUE
- e> TRUE
- f> FALSE

2.2: Why do we need the preprocessor directive `#include<iostream>`?

Ans: `#include<iostream>` directive causes the preprocessor to add-the contents of iostream file to the program.

2.3: How does a `main()` function in C++ differ from `main{ }` in C?

Ans: In C `main ()` by default returns the void type but in C++ it returns integer by default.

2.4: What do you think is the main advantage of the comment `//` in C++ as compared to the old C type comment?

Ans: `///` is more easy and time-saving than `/* */`

2.5: Describe the major parts of a C++ program.

Ans: Major parts of a C++ program :

1. Include files
2. Class declaration
3. Member function definitions
4. Main function program

Debugging Exercises

2.1: Identify the error in the following program.

```
#include<iostream.h>
void main()
{
    int i = 0;
    i = i + 1;
    cout << i << " ";
    /*comment \*/i = i + 1;
    cout << i;
}
```

Ans: Syntax error → /* comment */i=i+1;

2.2: Identify the error in the following program.

```
#include<iostream.h>
void main()
{
    short i=2500, j=3000;
    cout >> "i+j=" >> -(i+j);
}
```

Ans: cout >> "i+j=" >> Illegal structure operation. → -(i + j);

2.3: What will happen when you run the following program?

```
#include<iostream.h>
void main()
{
    int i=10, j=5;
    int modResult=0;
    int divResult=0;
    modResult = i%j;
    cout << modResult << " ";
    divResult = i/modResult;
    cout << divResult;
}
```

Ans: floating point Error or divide by zero → divResult = i/modResult;

Note: If this kind of Error exist in a program, the program will successfully compile but it will show Run Error.

2.4: Find errors, if any, in the following C++ statements.

- (a) `cout.<<"x=" x;` (b) `m = 5; // n = 10; // = m + n;` (c) `cin >>x; >>y;`
(d) `cout <<\h "Name:" <<name;`
(e) `cout <<"Enter value:"; cin >> x;`
(f) `/*Addition*/ z = x + y;`

Ans:

	Error	Correction
A	Statement missing	<code>cout<<"x="<<x;</code>
B	No error	
C	Expression-syntax-error Illigal character '\'. Statement missing	<code>cin>>x>>y;</code> <code>cout<<"\n Name"<<name;</code>
E	No error	
F	No error	

Programming Exercises

2.1: Write a program to display the following output using a single cout statement

Maths = 90
Physics = 77
Chemistry = 69

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 int main()
4 {
5
6     char *sub[]={ "Maths","Physics","Chemistry" };
7     int mark[]={90,77,69};
8     for(int i=0;i<3;i++)
9     {
10cout<<setw(10)<<sub[i]<<setw(3)<<"="<<setw(4)<<mark[i]<<endl;
11     }
12 return 0;
13}
```


output

Maths = 90
Physics = 77
Chemistry = 69

2.2: Write a program to read two numbers from the keyboard and display the larger value on the screen.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3
4 int main()
5 {
6     float a,b;
7     cout<<" Enter two values :"<<endl;
8     cin>>a>>b;
9     if(a>b)
10    cout<<" larger value = "<<a<<endl;
11    else
12    cout<<" larger value = "<<b<<endl;
13    return 0;
14}
```

output

Enter two values : 10 20
larger value = 20

2.3: Write a program to input an integer from the keyboard and display on the screen "WELL DONE" that many times.

Solution:

```
1 Solution:
2 #include<iostream.h>
3 #include<iomanip.h>
4 int main()
5 {
6     int n;
7     char *str;
8     str="WELL DONE";
```

```

9  cout<<" Enter an integer value ";
10 cin>>n;
11 for(int i=0;i<n;i++)
12 {
13     cout<<str<<endl;
14 }
15 return 0;
16}

```

output

```

Enter an integer value 5
WELL DONE
WELL DONE
WELL DONE
WELL DONE
WELL DONE

```

2.4: Write a program to read the values a, b and c and display x, where $x = a / b - c$.

Test the program for the following values:

- (a) a = 250, b = 85, c = 25
- (b) a = 300, b = 70, c = 70

Solution:

```

1 #include<iostream.h>
2 #include<iomanip.h>
3 int main()
4 {
5     float a,b,c,x;
6     cout<<" Enter the value of a,b, &c : "<<endl;
7     cin>>a>>b>>c;
8     if((b-c)!=0)
9     {
10         x=a/(b-c);
11         cout<<" x=a/(b-c) = "<<x<<endl;
12     }
13     else
14     {
15         cout<<" x= infinity "<<endl;
16     }
17     return 0;
18}

```

During First Run:**output**

Enter the value of a,b, &c : 250 85 25
 $x = a/(b-c) = 4.166667$

During Second Run:**output**

Enter the value of a,b, &c : 300 70 70
 $x = \text{infinity}$

2.5: Write a C++ program that will ask for a temperature in Fahrenheit and display it in Celsius

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3
4 int main()
5 {
6     float f,theta;
7     cout<<" Enter the temperature in Feranhite scale : ";
8     cin>>f;
9     theta=((f-32)/9)*5;
10    cout<<" Temperature in Celsius = "<<theta<<endl;
11    return 0;
12}
```

output

Enter the temperature in Feranhite scale : 105
Temperature in Celsius = 40.555557

2.6: Redo Exercise 2.5 using a class called temp and member functions.

Solution:

```

1 #include<iostream.h>
2 #include<iomanip.h>
3
4 class temp
5 {
6     float f,theta;
7 public:
8     float conversion(float f);
9 };
10
11 float temp::conversion(float f)
12 {
13     theta=((f-32)/9)*5;
14     return theta;
15 }
16 int main()
17 {
18     temp t;
19     float f;
20     cout<<" Enter temperature in Farenheite scale : "<<endl;
21     cin>>f;
22     cout<<" Temperature in Celsius scale = "<<t.conversion(f)<<endl;
23     return 0;
24 }

```

output

Enter the temperature in Ferenhite scale : 112
Temperature in Celsius = 44.444443

Chapter 3

Review Questions

3.1: Enumerate the rules of naming variables in C++. How do they differ from ANSI C rules?

Ans: Rules of naming variables in C++ are given below :

- Any character from 'a' to 'z' or 'A' to 'Z' can be used.
- Digit can be used but not at the beginning.
- Underscore can be used but space is not permitted.
- A keyword cannot be used as variable name.

In C++, a variable can be declared any where in the program but before the using of the variable.

In C all variables must be declared at the beginning of the program.

3.2: An unsigned int can be twice as large as the signed int. Explain how?

Ans:

In case of unsigned int the range of the input value is : 0 to $2^m - 1$. [where m is no. of bit]

In case of signed int the range of the input value is : -2^{m-1} to $(2^{m-1} - 1)$

$$\begin{aligned} \frac{\text{maximum value for unsigned int}}{\text{value for signed int}} &= \frac{2^m - 1}{(2^{m-1} - 1) + 2^{m-1}} \\ &= \frac{2^m - 1}{2 \cdot 2^{m-1} - 1} \\ &= \frac{2^m - 1}{2^m - 1} \\ &= 2 \end{aligned}$$

So, maximum value for unsigned int can be twice as large as the signed int.

* Here the absolute value of lower value -2^{m-1} for signed int must be considered for finding average value of signed int.

3.3: Why does C++ have type modifiers?

Ans: To serve the needs of various situation.

3.4: What are the applications of void data type in C++?

Ans: Two normal uses of void are

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

Example : void function (void)

Another interesting use of void is in the declaration of generic pointers.

Example :

```
void *gp; //gp is generic pointer.
```

A pointer value of any basic data type can be assigned to a generic pointer

```
int * ip;
```

```
gp = ip; // valid.
```

3.5: Can we assign a void pointer to an int type pointer? If not, why? Now can we achieve this?

Ans: We cannot assign a void pointer to an int type pointer directly. Because to assign a pointer to another pointer data type must be matched. We can achieve this using casting.

Example :
void * gp;
int *ip;
ip = (int *) gp
/br<>

3.6: Describe, with examples, the uses of enumeration data types.

Ans: An enumerated data type is a user-defined type. It provides a way for attaching names to numbers in ANSIC

Example :
enum kuet (EEE, CSE, ECE, CE, ME, IEM);

The enum keyword automatically enumerates
EEE to 0
CSE to 1
ECE to 2
CE to 3
ME to 4
IEM to 5

In C++ each enumerated data type retains its own separate type.

3.7: Describe the differences in the implementation of enum data type in ANSI C and C++.

Ans: Consider the following example :
enum kuet (EEE, CSE, ECE, CE, ME, IEM);

Here, kuet is tag name.

In C++ tag name become new type name. We can declare a new variables Example:

kuet student;
ANSI C defines the types of enum to be int.

In C int value can be automatically converted to on enum value. But in C++ this is not permitted.

Example:
student cgp = 3.01 // Error in C++
// OK in C.

student cgp = (student) 3.01 //OK in C++

3.8: Why is an array called a derived data type?

Ans: Derived data types are the data types which are derived from the fundamental data types. Arrays refer to a list of finite number of same data types. The data can be accessed by an index number from 0 to n. Hence an array is derived from the basic data type, so array is called derived data type.

3.9: The size of a char array that is declared to store a string should be one larger than the number of characters in the string. Why?

Ans: An additional null character must assign at the end of the string that's why the size of char array that is declared to store a string should be one larger than the number of characters in the string.

3.10: The const was taken from C++ and incorporated in ANSI C, although quite differently. Explain.

Ans: In both C and C++, any value declared as const cannot be modified by the program in any way. However there are some differences in implementation. 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. If we use const modifier alone, it defaults to int. For example, `const size = 10;` means `const int size = 10;` C++ requires const to be initialized. ANSI C does not require an initialization if none is given, it initializes the const to 0. In C++ a const is local, it can be made as global defining it as external. In C const is global in nature, it can be made as local declaring it as static.

3.11: How does a constant defined by `const` differ from the constant defined by the preprocessor statement `#define`?

Ans: Consider an example : `#define PI 3.14159`

The preprocessor directive `#define` appearing at the beginning of your program specifies that the identifier PI will be replaced by the text 3.14159 throughout the program.

The keyword `const` (for constant) precedes the data type of a variable specifies that the value of a variable will not be changed throughout the program.

In short, `const` allows us to create typed constants instead of having to use `#define` to create constants that have no type information.

3.12: In C++, a variable can be declared anywhere in the scope. What is the significance of this feature?

Ans: It is very easy to understand the reason of which the variable is declared.

3.13: What do you mean by dynamic initialization of a variable? Give an example.

Ans: When initialization is done at the time of declaration then it is known as dynamic initialization of variable

Example :

```
float area = 3.14159*rad * rad;
```

3.14: What is a reference variable? What is its major use?

Ans: A reference variable provides an alias (alternative name) for a previously defined variable. A major application of reference variables is in passing arguments to functions.

3.15: List at least four new operators added by C++ which aid OOP.

Ans:

New operators added by C++ are :

1. Scope resolution operator ::
2. Memory release operator delete delete
3. Memory allocation operator new
4. Field width operator setw
5. Line feed operator endl

3.16: What is the application of the scope resolution operator :: in C++?

Ans: A major application of the scope resolution operator is in the classes to identify the class to which a member function belongs.

3.17: What are the advantages of using new operator as compared to the junction mallocOr

Ans: Advantages of new operator over malloc ():

1. It automatically computes the size of the data object. We need not use the operator size of.
2. It automatically returns the correct pointer type, so that there is no need to use a type cast.
3. It is possible to initialize the object while creating the memory space.
4. Like any other operator, new and delete can be overloaded.

3.18: Illustrate with an example, how the `setw` manipulator works.

Ans:

`setw` manipulator specifies the number of columns to print. The number of columns is equal the value of argument of `setw ()` function.

For example :

`setw (10)` specifies 10 columns and print the message at right justified.

`cout << set (10) << "1234";` will print

1 2 3 4

If argument is negative message will be printed at left justified.

`cout <<setw(-10)<< "1234";` will print

1 2 3 4

3.19: How do the following statements differ?

- (a) `char *const p;`
- (b) `char const *p;`

Ans:

(a) `Char * const P;` means constant pointer.

(b) `Char const * P;` means pointer to a constant.

In case of (a) we can not modify the address of p.

In case of (b) we can not modify the contents of what it points to.

Debugging Exercises

3.1: What will happen when you execute the following code?

```
1#include <iostream.h>
2void main()
3{
4  int i=0;
5  i=400*400/400;
6  cout<<i;
7}
```

Ans: $i = 400 * 400 / 400$; Here, $400 * 400 = 160000$ which exceeds the maximum value of int variable. So wrong output will be shown when this program will be run.

Correction :

```
1 int I = 0;
```

should be changed as

```
1 long int i = 0;
```

to see the correct output.

3.2: Identify the error in the following program.

```
1 include<iostream.h>
2 void main()
3 {
4   int num[]={ 1,2,3,4,5,6};
5   num[1]==[1]num ? cout<<"Success" : cout<<"Error";
6 }
```

Ans: `num [1] = [1] num?`. You should write index number after *array name* but here index number is mention before array name in **[1] num**

So expression syntax error will be shown.

Correction : `num[1] = num[1]?` is the correct format

3.3: Identify the errors in the following program.

```
1 #include <iostream.h>
2 void main()
3 {
4   int i=5;
5   while(i)
6   {
7     switch(i)
8     {
9       default:
10      case 4:
```

```

11 case 5:
12 break;
13 case 1:
14 continue;
15 case 2:
16 case 3:
17 break;
18 }
19 i--;
20 }
21 }

```

Ans:

```

1 case 1 :
2 continue;

```

The above code will cause the following situation:

Program will be continuing while value of i is 1 and value of i is updating. So infinite loop will be created.

Correction: At last line i- should be changed as i--;

3.4: Identify the errors in the following program.

```

1 #include <iostream.h>
2 #define pi 3.14
3 int squareArea(int &);
4 int circleArea(int &);
5 void main()
6 {
7     int a=10;
8     cout << squareArea(a) << " ";
9     cout << circleArea(a) << " ";
10    cout << a << endl;
11 {
12    int squareArea(int &a)
13 {
14    return a *== a;
15 }
16 int circleArea(int &r)
17 {
18    return r = pi * r * r;
19 }

```

Ans: Assignment operator should be used in the following line:

```
lreturn a *==a;
```

That means the above line should be changed as follows:

```
lreturn a *=a;
```

3.5: Missing

3.6: Find errors, if any, in the following C++ statements.

- (a) long float x;
- (b) char *cp = vp; // vp is a void pointer
- (c) int code = three; // three is an enumerator
- (d) int sp = new; // allocate memory with new
- (e) enum (green, yellow, red);
- (f) int const sp = total;
- (g) const int array_size;
- (h) for (i=1; int i<10; i++) cout << i << "/n"; (i) int & number = 100; (j) float *p = new int 1101;
- (k) int public = 1000; (l) char name[33] = "USA";

Ans:

No.	Error	Correction
(a)	too many types	float x; or double x;
(b)	type must be matched	char *cp = (char*) vp;
(c)	No error	
(d)	syntax error	int*p = new int [10];
(e)	tag name missing	enum colour (green, yellow, red)
(f)	address have to assign instead of content	int const * p = &total;
(g)	C++ requires a const to be initialized	const int array-size = 5;
(h)	Undefined symbol i	for (int I = 1; i <10; i++) cout << i << "/n";
(i)	invalid variable name	int number = 100;
(j)	wrong data type	float *p = new float [10];
(k)	keyword can not be used as a variable name	int public1 = 1000;
(l)	array size of char must be larger than the number of characters in the string	char name [4] = "USA";

Programming Exercises

3.1: Write a function using reference variables as arguments to swap the values of a pair of integers.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3
4 void swap_func(int &a,int &b)
5 {
6
7     cout<<" Before swapping "<<endl
8     <<" a = "<<a<<endl<<" b = "<<b<<endl<<endl;
9     int temp;
10    temp=a;
11    a=b;
12    b=temp;
13    cout<<" After swapping "<<endl
14    <<" a = "<<a<<endl<<" b = "<<b<<endl<<endl;
15
16}
17
18int main()
19{
20    int x,y;
21    cout<<" Enter two integer value : "<<endl;
22    cin>>x>>y;
23    swap_func (x,y);
24    return 0;
25}
```

output

```
Enter two integer value : 56 61
Before swapping
a = 56
b = 61
After swapping
a = 56
b = 61
```


3.2: Write a function that creates a vector of user given size M using new operator.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 int main()
4 {
5     int m;
6     int *v;
7     cout<<" Enter vector size : "<<endl;
8     cin>>m;
9     v=new int [m];
10    cout<<" to check your performance insert "<<m<<" integer value"<<endl;
11    for(int i=0;i<m;i++)
12    {
13        cin>>v[i];
14    }
15    cout<<" Given integer value are :"<<endl;
16    for(i=0;i<m;i++)
17    {
18        if(i==m-1)
19            cout<<v[i];
20        else
21            cout<<v[i]<<",";
22    }
23    cout<<endl;
24    return 0;
25 }
```

output

```
Enter vector size : 5
to check your performance insert 5 integer value
7 5 9 6 1
Given integer value are :
7, 5, 9, 6, 1
```

3.3: Write a program to print the following outputs using for loops

```
1
22
333
4444
```

55555
.....

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 int main()
4 {
5     int n;
6     cout<<" Enter your desired number : "<<endl;
7     cin>>n;
8     cout<<endl<<endl;
9     for(int i=1;i<=n;i++)
10    {
11        for(int j=1;j<=i;j++)
12        {
13            cout<<i;
14        }
15        cout<<endl;
16    }
17    return 0;
18}
```

output

Enter your desired number : 6
1
22
333
4444
55555
666666

3.4: Write a program to evaluate the following investment equation

$$V = P(1+r)^n$$

and print the tables which would give the value of V for various of the following values of P, r and n:

P: 1000, 2000, 3000,.....,10,000

r: 0.10, 0.11, 0.12,.....,0.20

n: 1, 2, 3,.....,10

(Hint: P is the principal amount and V is the value of money at the end of n years. This equation can be recursively written as

$$V = P(1 + r)$$

$$P = V$$

In other words, the value of money at the end of the first year becomes the principal amount for the next year and so on)

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 #include<math.h>
4 #define size 8
5
6 int main()
7 {
8     float v,pf;
9     int n=size;
10    float p[size]={ 1000,2000,3000,4000,5000,6000,7000,8000 };//9000,1000};
11    float r[size]={0.11,0.12,0.13,0.14,0.15,0.16,0.17,0.18};//,0.19,0.20};
12
13    cout<<setw(5)<<"n=1";
14    for(int i =2;i<=size;i++)
15    cout<<setw(9)<<"n="<<i;
16    cout<<"\n";
17
18    for(i=0;i<size;i++)
19    {
20        cout<<setw(-6)<<"p=";
21        for(int j=0;j<size;j++)
22        {
23            if(j==0)
24                pf=p[i];
25
26            v=pf*(1+r[i]);
27
28            cout.precision(2);
29            cout.setf(ios::fixed, ios::floatfield);
30            cout<<v<<setw(10);
31            pf=v;
32        }
33        cout<<"\n";
34
35    }
36    return 0;
37}
```

output

n=1	n=2	n=3	n=4	n=5	n=6	n=7
p=1110	1232.1	1367.63	1518.07	1685.06	1870.41	2076.16
p=2240	2508.8	2809.86	3147.04	3524.68	3947.65	4421.36
p=3390	3830.7	4328.69	4891.42	5527.31	6245.86	7057.82
p=4560	5198.4	5926.18	6755.84	7701.66	8779.89	10009.08
p=5750	6612.5	7604.37	8745.03	10056.79	11565.3	13300.1
p=6960	8073.6	9365.38	10863.84	12602.05	14618.38	16957.32

p=8190 9582.3 11211.29 13117.21 15347.14 17956.15 21008.7
p=9440 11139.2 13144.26 15510.22 18302.06 21596.43 25483.79

3.5: An election is contested by five candidates. The candidates are numbered 1 to 5 and the voting is done by marking the candidate number on the ballot paper. Write a program to read the ballots and count the vote cast for each candidate using an array variable count. In case, a number read is outside the range 1 to 5, the ballot should be considered as a “spoilt ballot” and the program should also count the numbers of “spoilt ballots”.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 int main()
4 {
5     int count[5];
6     int test;
7     for(int i=0;i<5;i++)
8     {
9         count[i]=0;
10    }
11    int spoilt_ballot=0;
12    cout<<" You can vot candidate 1 to 5 "<<endl
13    <<" press 1 or 2 or 3 or 4 or 5 to vote "<<endl
14    <<" candidate 1 or 2 or 3 or 4 or 5 respectively "<<endl
15    <<" press any integer value outside the range 1 to 5 for NO VOTE  "<<endl<<" press any
16negative value to terminate and see result :"<<endl;
17
18    while(1)
19    {
20        cin>>test;
21
22        for(int i=1;i<=5;i++)
23        {
24            if(test==i)
25            {
26                count[i-1]++;
27            }
28        }
29        if(test<0)
30            break;
31        else if(test>5)
32            spoilt_ballot++;
33    }
34    for(int k=1;k<=5;k++)
35        cout<<" candidate "<<k<<setw(12);
36    cout<<endl;
37    cout<<setw(7);
```

```

38     for(k=0;k<5;k++)
39     cout<<count[k]<<setw(13);
40     cout<<endl;
41     cout<<" spoilt_ballot "<<spoilt_ballot<<endl;
42     return 0;
    }

```

output

```

You can vot candidate 1 to 5
press 1 or 2 or 3 or 4 or 5 to vote
candidate 1 or 2 or 3 or 4 or 5 respectively
press any integer value outside the range 1 to S for NO VOTE
press any negative value to terminate and see result :
1
1
1
5
4
3
5
5
2
1
3
6
-1
candidate 1 candidate 2 candidate 3 candidate 4 candidate S
4 1 2 1 3
spoilt_ballot 1

```

3.6: A cricket has the following table of batting figure for a series of test matches:

Player's name	Run	Innings	Time not
outSachin	8430	230	18Saurav
			4200
			130

Write a program to read the figures set out in the above forms, to calculate the batting arranges and to print out the complete table including the averages.

Solution:

```

1 #include<iostream.h>
2 #include<iomanip.h>
3
4 char *serial[3]={" FIRST "," SECOND " ," THIRD "};//global declaration

```



```

5
6 int main()
7 {
8     int n;
9     char name[100][40];
10    int *run;
11    int *innings;
12    int *time_not_out;
13    cout<<" How many players' record would you insert ? :";
14    cin>>n;
15    //name=new char[n];
16    run=new int[n];
17    innings=new int[n];
18    time_not_out=new int[n];
19
20    for(int i=0;i<n;i++)
21    {
22        if(i>2)
23        {
24            cout<<"\n Input details of "<<i+1<<"th"<<" player's"<<endl;
25        }
26        else
27        {
28            cout<<" Input details of "<<serial[i]<<"player's : "<<endl;
29        }
30
31        cout<<" Enter name : ";
32
33        cin>>name[i];
34        cout<<" Enter run : ";
35        cin>>run[i];
36        cout<<" Enter innings : ";
37        cin>>innings[i];
38        cout<<" Enter times not out : ";
39        cin>>time_not_out[i];
40    }
41
42    float *average;
43    average=new float[n];
44    for(i=0;i<n;i++)
45    {
46        float avrg;
47        average[i]=float(run[i])/innings[i];
48
49    }
50    cout<<endl<<endl;
51    cout<<setw(12)<<"player's name "<<setw(11)<<"run"<<setw(12)<<"innings"<<setw(16)<<"Average"<<setw
52out"<<endl;
53    for(i=0;i<n;i++)
54    {
55        cout<<setw(14)<<name[i]<<setw(11)<<run[i]<<setw(9)<<innings[i]<<setw(18)<<average[i]<<setw(15)<<

```

```

56     }
57     cout<<endl;
58
59     return 0;
    }

```

output

```

How many players record would you insert ? :2
Input details of FIRST player's :
Enter name : Sakib-Al-Hassan
Enter run : 1570
Enter innings : 83
Enter times not out : 10
Input details of SECOND player's :
Enter name : Tamim
Enter run : 2000
Enter innings : 84
Enter times not out : 5
player's name run innings Average times not out
Sakib-Al-Hassan 1570 83 18.915663 10
Tamim 2000 84 23.809525 5

```

3.7: Write a program to evaluate the following function to 0.0001% accuracy

- (a) $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$
- (b) $SUM = 1 + \frac{1}{2^2} + \frac{1}{3^3} + \frac{1}{4^4} + \dots$
- (c) $\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$

Solution (a):

```

1 #include<iostream.h>
2 #include<math.h>
3 #include<iomanip.h>
4 #define accuracy 0.0001
5 #define pi 3.1416
6
7 long int fac(int a)
8 {
9     if(a<=1)
10        return 1;
11     else
12        return a*fac(a-1);
13}
14int main()

```

```

15{
16 float y,y1,x,fx;
17 int n=1;
18 int m;
19 //const float pi=3.1416;
20 cout<<" Enter the value of angle in terms of degree: ";
21 cin>>x;
22 float d;
23 d=x;
24 int sign;
25 sign=1;
26if(x<0)
27{
28 x=x*(-1);
29 sign=-1;
30 }
31again:
32 if(x>90 && x<=180)
33 {
34
35 x=180-x;
36
37 }
38 else if(x>180 && x<=270)
39 {
40 x=x-180;
41 sign=-1;
42 }
43 else if(x>270 && x<=360)
44 {
45 x=360-x;
46 sign=-1;
47 }
48
49 else if(x>360)
50 {
51 int m=int(x);
52 float fractional=x-m;
53 x=m%360+fractional;
54 if(x>90)
55 goto again;
56 else
57 sign=1;
58
59 }
60 x=(pi/180)*x;
61 m=n+1;
62 fx=0;
63 for(;;)
64 {
65 long int h=fac(n);

```

```

66     y=pow(x,n);
67     int factor=pow(-1,m);
68     y1=y*factor;
69     fx+=y1/h;
70     n=n+2;
71     m++;
72     if(y/h<=accuracy)
73         break;
74 }
75
76 cout<<"sin("<<d<<")= "<<fx*sign<<endl;
77 return 0;
78}

```

output

Enter the value of angle in terms of degree: 120
sin(120)= 0.866027

Solution (b):

```

1 #include<iostream.h>
2 #include<math.h>
3 #define accuracy 0.0001
4 int main()
5 {
6     int n;
7     float sum,n1,m;
8     n=1;sum=0;
9     for(int i=1;;i++)
10    {
11        n1=float(1)/n;
12        m=pow(n1,i);
13        sum+=m;
14        if(m<=accuracy)
15            break;
16
17        n++;
18    }
19    cout<<sum<<"\n";
20    return 0;
21}
22    Sample Output(b)
23
24    Solution: (c)
25    #include<iostream.h>
26#include<math.h>
27#define accuracy 0.0001

```

```

28
29     long int fac(int n)
30{
31     if(n<=1)
32         return 1;
33     else
34         return n*fac(n-1);
35}
36
37     int main()
38{
39
40     float y,y1,x,fx;
41     int n=1;
42     int m;
43     const float pi=3.1416;
44     cout<<" Enter the value of angle in terms of degree: ";
45     cin>>x;
46     if(x<0)
47         x=x*(-1);
48     x=(pi/180)*x;
49
50     fx=1;
51
52     m=2;
53     float y2;
54     long int h;
55     for(;;)
56     {
57         h=fac(m);
58         int factor=pow(-1,n);
59         y1=pow(x,m);
60         y2=(y1/h)*factor;
61         fx+=y2;
62         if(y1/h<=accuracy)
63             break;
64         m=m+2;
65         n++;
66     }
67     cout<<fx<<"\n";
68}

```

output

Enter the value of angle in terms of degree: 60
0.866025

3.8: Write a program to print a table of values of the function

$$Y = e^{-x}$$

For x varying from 0 to 10 in steps of 0.1. The table should appear as follows

TABLE FOR Y =EXP[-X];

X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.900
1.0									
.									
.									
9.0									

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 #include<math.h>
4     int main()
5 {
6     float x,y;
7     cout<<"          TABLE FOR Y=EXP(-X)          :\n\n";
8     cout<<"x";
9     for(float k=0;k<.7;k=k+0.1)
10    cout<<setw(10)<<k;
11    cout<<"\n";
12    for(k=0;k<10*.7;k=k+0.1)
13    cout<<"-";
14    cout<<"\n";
15        for(float j=0;j<10;j++)
16    {
17        cout<<j<<setw(4);
18        for(float i=0;i<.7;i=i+0.1)
19        {
20            x=i+j;
21            y=exp(-x);
22            cout.precision(6);
23            cout.setf(ios::fixed,ios::floatfield);
24            cout<<setw(10)<<y;
25        }
```

```

26     cout<<"\n";
27     }
28     return 0;
29 }

```

Note: Here we work with 0.4 for a good looking output.

output

TABLE FOR Y=EXP(-X)

x	0	0.1	0.2	0.3	0.4
0	1	0.904837	0.818731	0.740818	0.67032
1	0.367879	0.332871	0.301194	0.272532	0.246597
2	0.135335	0.122456	0.110803	0.100259	0.090718
3	0.049787	0.045049	0.040762	0.036883	0.033373
4	0.018316	0.016573	0.014996	0.013569	0.012277
5	0.006738	0.006097	0.005517	0.004992	0.004517
6	0.002479	0.002243	0.002029	0.001836	0.001662
7	0.000912	0.000825	0.000747	0.000676	0.000611
8	0.000335	0.000304	0.000275	0.000249	0.000225
9	0.000123	0.000112	0.000101	0.000091	0.000083

3.9: Write a program to calculate the variance and standard deviation of

N numbers

$$\text{Variance} = 1/N \sum (x_i - \bar{x})^2$$

$$\text{Standard deviation} = \sqrt{1/N \sum (x_i - \bar{x})^2}$$

$$\text{Where } \bar{x} = 1/N \sum x_i$$

Solution:

```
1 #include<iostream.h>
```

```

2 #include<math.h>
3     int main()
4 {
5     float *x;
6         cout<<" How many number ? .:";
7     int n;
8     cin>>n;
9     x=new float[n];
10    float sum;
11    sum=0;
12    for(int i=0;i<n;i++)
13    {
14        cin>>x[i];
15        sum+=x[i];
16    }
17    float mean;
18    mean=sum/n;
19    float v,v1;
20    v1=0;
21    for(i=0;i<n;i++)
22    {
23        v=x[i]-mean;
24        v1+=pow(v,2);
25    }
26    float variance,std_deviation;
27    variance=v1/n;
28    std_deviation=sqrt(variance);
29    cout<<"\n\n variance = "<<variance<<"\n standard deviation = "<<std_deviation<<"\n";
30
31    return 0;
32}

```

output

```

How many number ? :5
10
2
4
15
2
variance = 26.24
standard deviation = 5.122499

```

3.10: An electricity board charges the following rates to domestic users to discourage large consumption of energy:

For the first 100 units – 60P per unit

For the first 200 units – 80P per unit

For the first 300 units – 90P per unit

All users are charged a minimum of Rs. 50.00. If the total amount is more than Rs. 300.00 then an additional surcharge of 15% is added.

Write a program to read the names of users and number of units consumed and print out the charges with names.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 int main()
4 {
5     int unit;
6     float charge,additional;
7     char name[40];
8     while(1)
9
10    {
11        input:
12        cout<<" Enter consumer name & unit consumed :";
13        cin>>name>>unit;
14        if(unit<=100)
15            {
16                charge=50+(60*unit)/100;
17            }
18        else if(unit<=300 && unit>100)
19            {
20                charge=50+(80*unit)/100;
21            }
22        else if(unit>300)
23            {
24                charge=50+(90*unit)/float(100);
25                additional=(charge*15)/100;
26                charge=charge+additional;
27            }
28        cout<<setw(15)<<"Name"<<setw(20)<<"Charge"<<endl;
29        cout<<setw(15)<<name<<setw(20)<<charge<<endl;
30        cout<<" Press o for exit / press 1 to input again :";
31        int test;
32        cin>>test;
33        if(test==1)
34            goto input;
35        else if(test==0)
36            break;
37    }
```

```
38 return 0;
39 }
```

output

Enter consumer name & unit consumed :sattar 200

Name	Charge
------	--------

sattar	210
--------	-----

Press 0 for exit / press 1 to input again :1

Enter consumer name & unit consumed :santo 300

Nmae	Charge
------	--------

santo	290
-------	-----

Press 0 for exit / press 1 to input again : 0

Chapter 4

Review Questions

4.1: State whether the following statements are TRUE or FALSE.

- (a) A function argument is a value returned by the function to the calling program.
- (b) When arguments are passed by value, the function works with the original arguments in the calling program.
- (c) When a function returns a value, the entire function call can be assigned to a variable.
- (d) A function can return a value by reference.
- (e) When an argument is passed by reference, a temporary variable is created in the calling program to hold the argument value.
- (f) It is not necessary to specify the variable name in the function prototype.

Ans:

- | | |
|-----------|-----------|
| (a) FALSE | (d) TRUE |
| (b) FALSE | (e) FALSE |
| (c) TRUE | (f) TRUE |

4.2: What are the advantages of function prototypes in C++?

Ans: Function prototyping is one of the major improvements added to C++ functions. The prototype describes the function interface to the compiler by giving details such as the number and type of arguments and the type of return values.

4.3: Describe the different styles of writing prototypes.

Ans:

General form of function prototyping :

return_type function_name (argument_list)

Example :

int do_something (void);

float area (float a, float b);

float area (float, float);

4.4: Find errors, if any, in the following function prototypes.

(a) float average(x,y);

(b) int mul(int a,b);

(c) int display(...);

(d) void Vect(int? &V, int & size);

(e) void print(float data[], size = 201);

Ans:

No.	Error	Correction
(a)	Undefined symbol x, y	float average (float x, float y)
(b)	Undefined symbol b	int mul (int a, int b);
(c)	No error	
(d)	invalid character in variable name	void vect (int &v, int &size);
(e)	Undefined symbol 's'	void print (float data [], int size = 20);

4.5: What is the main advantage of passing arguments by reference?

Ans: When we pass arguments by reference, the formal arguments in the called function become aliases to the 'actual' arguments in the calling function.

4.6: When will you make a function inline? Why?

Ans: When a function contains a small number of statements, then it is declared as inline function.

By declaring a function inline the execution time can be minimized.

4.7: How does an inline function differ from a preprocessor macro?

Ans: The macros are not really functions and therefore, the usual error checking does not occur during compilation. But using inline-function this problem can be solved.

4.8: When do we need to use default arguments in a function?

Ans: When some constant values are used in a user defined function, then it is needed to assign a default value to the parameter.

Example :

```
1 float area (float r, float PI = 3.1416)
2 {
3     return PI*r*r;
4 }
```

4.9: What is the significance of an empty parenthesis in a function declaration?

Ans: An empty parentheses implies arguments is void type.

4.10: What do you mean by overloading of a function? When do we use this concept?

Ans: Overloading of a function means the use of the same thing for different purposes. When we need to design a family of functions-with one function name but with different argument lists, then we use this concept.

4.11: Comment on the following function definitions:

(a)

```
1 int *f( )
2 {
3     int m = 1;
4     ....
5     ....
6     return(&m);
7 }
```

(b)

```
1 double f()
2 {
3     .....
4     .....
5     return(1);
6 }
```

(c)

```
1 int & f()
2 {
3     int n = 10;
4     .....
5     .....
6     return(n);
7 }
```

Ans:

No.	Comment
(a)	This function returns address of m after execution this function.
(b)	This function returns 1 after execution.
(c)	returns address of n

Debugging Exercises

4.1: Identify the error in the following program.

```
#include <iostream.h>
int fun()
{
    return 1;
}
float fun()
{
    return 10.23;
}
void main()
{
    cout <<(int)fun() << ' ';
    cout << (float)fun() << ' ';
}
```

Solution: Here two function are same except return type. Function overloading can be used using different argument type but not return type.

Correction : This error can be solved as follows :

```
#include<iostream.h>

int fun()
{
    return 1;
}
float fun1()
{
    return 10.23;
}

void main()
{
    cout<<fun()<<" ";
    cout<<fun1()<<" ";
}
```

4.2: Identify the error in the following program.

```
#include <iostream.h>
void display(const Int const1=5)
{
    const int const2=5;
    int array1[const1];
    int array2[const2];
    for(int i=0; i<5; i++)
    {
        array1[i] = i;
        array2[i] = i*10;
        cout <<array1[i]<< ' ' << array2[i] << ' ';
    }
}
void main()
{
    display(5);
}
```

Solution:

```
#include<iostream.h>
```

```

void display()
{
    const int const1=5;
    const int const2=5;
    int array1[const1];
    int array2[const2];

    for(int i=0;i<5;i++)
    {
        array1[i]=i;
        array2[i]=i*10;
        cout<<array1[i]<<" "<<array2[i]<<" ";
    }
}

void main()
{
    display();
}

```

4.3: Identify the error in the following program.

```

#include <iostream.h>
int gValue=10;
void extra()
{
    cout << gValue << ' ';
}
void main()
{
    extra();
    {
        int gValue = 20;
        cout << gValue << ' ';
        cout << : gValue << ' ';
    }
}

```

Solution:

Here cout << : gvalue << " "; replace with cout <<::gvalue<< " ";

```

#include <iostream.h>
int gValue=10;
void extra()
{

```



```

    cout << gValue << ' ';
}
void main()
{
    extra();
    {
        int gValue = 20;
        cout << gValue << ' ';
        cout <<::gvalue<< " ";
    }
}

```

4.4: Find errors, if any, in the following function definition for displaying a matrix: void display(int A[][], int m, int n)

```

{
    for(i=0; i<m; i++)
        for(j=0; j<n; j++)
            cout<<" "<<A[i][j];
        cout<<"\n";
}

```

Solution:

First dimension of an array may be variable but others must be constant.

Here int A [] [] replace by the following code:

```

int A [ ] [10];
int A[10] [10];
int A [ ] [size];
int A [size] [size];

```

Where const int size = 100;
any other numerical value can be assigned to size.

Programming Exercises

4.1: Write a function to read a matrix of size m*n from the keyboard.

Solution:

```

1 #include<iostream.h>
2 #include<iomanip.h>
3
4 void matrix(int m,int n)

```

```

5 {
6  float **p;
7  p=new float*[m];
8  for(int i=0;i<m;i++)
9  {
10   p[i]=new float[n];
11  }
12  cout<<" Enter "<<m<<"by"<<n<<" matrix elements one by one "<<endl;
13  for(i=0;i<m;i++)
14  {
15   for(int j=0;j<n;j++)
16   {
17    float value;
18    cin>>value;
19    p[i][j]=value;
20   }
21  }
22  cout<<" The given matrix is :"<<endl;
23  for(i=0;i<m;i++)
24  {
25   for(int j=0;j<n;j++)
26   {
27    cout<<p[i][j]<<" ";
28   }
29   cout<<"\n";
30  }
31}
32
33int main()
34{
35  int r,c;
36  cout<<" Enter size of matrix : ";
37  cin>>r>>c;
38  matrix(r,c);
39  return 0;
40}

```

output

Enter size of matrix : 3 4

Enter 3 by 4 matrix elements one by one

1 2 3 4

2 3 4 5

3 4 5 6

The given matrix is :

```
1 2 3 4
2 3 4 5
3 4 5 6
```

4.2: Write a program to read a matrix of size $m*n$ from the keyboard and display the same on the screen using function.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3
4 void matrix(int m,int n)
5 {
6     float **p;s
7     p=new float*[m];
8     for(int i=0;i<m;i++)
9     {
10        p[i]=new float[n];
11    }
12    cout<<" Enter "<<m<<" by "<<n<<" matrix elements one by one "<<endl;
13    for(i=0;i<m;i++)
14    {
15        for(int j=0;j<n;j++)
16        {
17            float value;
18            cin>>value;
19            p[i][j]=value;
20        }
21    }
22    cout<<" The given matrix is : "<<endl;
23    for(i=0;i<m;i++)
24    {
25        for(int j=0;j<n;j++)
26        {
27            cout<<p[i][j]<<" ";
28        }
29        cout<<"\n";
30    }
31}
32
33int main()
34{
35    int r,c;
36    cout<<" Enter size of matrix : ";
```

```

37 cin>>r>>c;
38 matrix(r,c);
39 return 0;
40}

```

output

Enter size of matrix : 4 4

Enter 4 by 4 matrix elements one by one

1 2 3 4 7

2 3 4 5 8

3 4 5 6 9

The given matrix is :

1 2 3 4 7

2 3 4 5 8

3 4 5 6 9

4.3: Rewrite the program of Exercise 4.2 to make the row parameter of the matrix as a default argument.

Solution:

```

1 #include<iostream.h>
2 #include<iomanip.h>
3
4 void matrix(int n,int m=3)
5 {
6     float **p;
7     p=new float*[m];
8     for(int i=0;i<m;i++)
9     {
10        p[i]=new float[n];
11    }
12    cout<<" Enter "<<m<<" by "<<n<<" matrix elements one by one "<<endl;
13    for(i=0;i<m;i++)
14    {
15        for(int j=0;j<n;j++)
16        {

```

```

17     float value;
18     cin>>value;
19     p[i][j]=value;
20 }
21 }
22 cout<<" The given matrix is :"<<endl;
23 for(i=0;i<m;i++)
24 {
25     for(int j=0;j<n;j++)
26     {
27         cout<<p[i][j]<<" ";
28     }
29     cout<<"\n";
30 }
31}
32
33int main()
34{
35     int c;
36     cout<<" Enter column of matrix : ";
37     cin>>c;
38     matrix(c);
39     return 0;
40}

```

output

Enter column of matrix : 3

Enter 3 by 3 matrix elements one by one

1 2 3

2 3 4

3 4 5

The given matrix is :

1 2 3

2 3 4

3 4 5

4.4: The effect of a default argument can be alternatively achieved by overloading. Discuss with examples.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3
4 void matrix(int m,int n)
5 {
6     float **p;
7     p=new float*[m];
8     for(int i=0;i<m;i++)
9     {
10        p[i]=new float[n];
11    }
12    cout<<" Enter "<<m<<"by"<<n<<" matrix elements one by one "<<endl;
13    for(i=0;i<m;i++)
14    {
15        for(int j=0;j<n;j++)
16        {
17            float value;
18            cin>>value;
19            p[i][j]=value;
20        }
21    }
22    cout<<" The given matrix is :"<<endl;
23    for(i=0;i<m;i++)
24    {
25        for(int j=0;j<n;j++)
26        {
27            cout<<p[i][j]<<" ";
28        }
29        cout<<"\n";
30    }
31}
32void matrix(int m,long int n=3)
33{
34    float **p;
35    p=new float*[m];
36
37    for(int i=0;i<m;i++)
38    {
39        p[i]=new float[n];
40    }
41    cout<<" Enter "<<m<<" by "<<n<<" matrix elements one by one "<<endl;
42    for(i=0;i<m;i++)
43    {
44        for(int j=0;j<n;j++)
45        {
46            float value;
47            cin>>value;
48            p[i][j]=value;
```



```

49     }
50   }
51   cout<<" The given matrix is :"<<endl;
52   for(i=0;i<m;i++)
53   {
54     for(int j=0;j<n;j++)
55     {
56       cout<<p[i][j]<<" ";
57     }
58     cout<<"\n";
59   }
60}
61
62int main()
63{
64   int r;
65   cout<<" Enter row of matrix : ";
66   cin>>r;
67   matrix(r);
68   return 0;
69}

```

output

Enter column of matrix : 2

Enter 2 by 3 matrix elements one by one

1 0 1

0 2 1

The given matrix is :

1 0 1

0 2 1

4.5: Write a macro that obtains the largest of the three numbers.

Solution:

```

1 #include<iostream.h>
2 #include<iomanip.h>
3
4 float large(float a,float b,float c)
5 {

```

```

6  float largest;
7  if(a>b)
8  {
9      if(a>c)
10     largest=a;
11     else
12     largest=c;
13 }
14 else
15 {
16     if(b>c)
17     largest=b;
18     else
19     largest=c;
20 }
21 return largest;
22}
23
24int main()
25{
26 float x,y,z;
27 cout<<" Enter three values : ";
28 cin>>x>>y>>z;
29 float largest=large(x,y,z);
30 cout<<" large = "<<largest<<endl;
31 return 0;
32}

```

output

Enter three values : 4 5 8

large = 8

4.6: Redo Exercise 4.16 using inline function. Test the function using a main function.

Solution:

Blank

4.7: Write a function power() to raise a number m to power n. The function takes a double value for m and int value for n and returns the result correctly. Use a default value of 2 for n to make the function to calculate the squares when this argument is omitted. Write a main that gets the values of m and n from the user to test the function.

Solution:

```

1 #include<iostream.h>
2 #include<iomanip.h>
3 #include<math.h>
4
5 long double power(double m,int n)
6 {
7     long double mn=pow(m,n);
8     return mn;
9 }
10 long double power(double m,long int n=2)
11 {
12     long double mn=pow(m,n);
13     return mn;
14 }
15 int main()
16 {
17     long double mn;
18     double m;
19     int n;
20
21     cout<<" Enter the value of m & n"<<endl;
22     cin>>m>>n;
23     mn=power(m,n);
24     cout<<" m to power n : "<<mn<<endl;
25     mn=power(m);
26     cout<<" m to power n : "<<mn<<endl;
27     return 0;
28 }

```

output

Enter the value of m & n

12 6

m to power n : 2985984

m to power n: 144

4.6: Redo Exercise 4.16 using inline function. Test the function using a main function.

Solution:

Blank

4.7: Write a function power() to raise a number m to power n. The function takes a double value for m and int value for n and returns the result correctly. Use a default value of 2 for

n to make the function to calculate the squares when this argument is omitted. Write a main that gets the values of m and n from the user to test the function.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 #include<math.h>
4
5 long double power(double m,int n)
6 {
7     long double mn=pow(m,n);
8     return mn;
9 }
10 long double power(double m,long int n=2)
11 {
12     long double mn=pow(m,n);
13     return mn;
14 }
15 int main()
16 {
17     long double mn;
18     double m;
19     int n;
20
21     cout<<" Enter the value of m & n"<<endl;
22     cin>>m>>n;
23     mn=power(m,n);
24     cout<<" m to power n : "<<mn<<endl;
25     mn=power(m);
26     cout<<" m to power n : "<<mn<<endl;
27     return 0;
28 }
```

output

Enter the value of m & n

12 6

m to power n : 2985984

m to power n: 144

4.8: Write a function that performs the same operation as that of Exercise 4.18 but takes an int value for m. Both the functions should have the same name. Write a main that calls both the functions. Use the concept of function overloading.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 #include<math.h>
4
5 long double power(int m,int n)
6 {
7     long double mn= (long double)pow(m,n);
8     return mn;
9 }
10long double power(int m,long int n=2)
11{
12     long double mn=(long double)pow(m,n);
13     return mn;
14}
15int main()
16{
17     long double mn;
18     int m;
19     int n;
20
21     cout<<" Enter the value of m & n"<<endl;
22     cin>>m>>n;
23     mn=power(m,n);
24     cout<<" m to power n : "<<mn<<endl;
25     mn=power(m);
26     cout<<" m to power n : "<<mn<<endl;
27     return 0;
28}
```

output

Enter the value of m & n

15 16

m to power n : 6.568408e+18

m to power n: 225

Chapter 5

Review Questions

5.1: How do structures in C and C++ differ?

Ans:

C structure member functions are not permitted but in C++ member functions are permitted.

5.2: What is a class? How does it accomplish data hiding?

Ans:

A class is a way to bind the data and its associated functions together. In class we can declare a data as private for which the functions accomplish data—outside the class can not access the data and thus if hiding.

5.3: How does a C++ structure differ from a C++ class?

Ans:

Initially (in C) a structure was used to bundle different of data types together to perform a particular functionality C++ extended the structure to contain functions also. The difference is that all declarations inside a structure are default public.

5.4: What are objects? How are they created?

Ans:

Object is a member of class. Let us consider a simple example. `int a;` here a is a variable of int type. Again consider class fruit.

```
{  
}
```

here fruit is the class-name. We can create an object as follows:

```
fruit mango;
```

here mango is a object.

5.5: How is a member function of a class defined?

Ans:

member function of a class can be defined in two places:

- * Outside the class definition.

- * Inside the class definition.

Inside the class definition : same as other normal function.

Outside the class definition : general form:

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


```
{  
function body  
}
```

5.6: Can we use the same function name for a member function of a class and an outside function in the same program file? If yes, how are they distinguished? If no, give reasons.

Ans:

Yes, We can distinguish them during calling to main () function. The following example illustrates this:

```
1 #include<iostream.h>  
2 void f()  
3 {  
4 cout<<"Outside Of class \n";  
5 }  
6  
7 class santo  
8 {  
9 public:  
10void f()  
11{  
12     cout<<"Inside of class \n";  
13}  
14};  
15  
16void main()  
17{  
18f(); // outside f() is calling.  
19santo robin;  
20robin.f(); // Inside f() is calling.  
21}
```

5.7: Describe the mechanism of accessing data members and member functions in the following cases:

- (a) Inside the main program.
- (b) Inside a member function of the same class.
- (c) Inside a member function of another class.

Ans:

- (a) Using object and dot membership operator.
- (b) Just like accessing a local variable of a function.
- (c) Using object and dot membership operator.

The following example explains how to access data members and member functions inside a member function of another class.

```
1 #include<iostream.h>
2
3 class a
4 {
5     public:
6         int x;
7         void display()
8         {
9             cout<<"This is class a \n";
10            x=111;
11        }
12};
13
14class b
15{
16    public:
17    void display()
18    {
19        a s;
20        cout<<" Now member function 'display()' of class a is calling from class b \n";
21        s.display();
22        cout<<" x = "<<s.x<<"\n";
23    }
24};
25
26void main()
27{
28    b billal; // billal is a object of class b.
29    billal.display();
30}
```

5.8: When do we declare a member of a class static?

Ans:

When we need a new context of a variable the n we declare this variable as static.

5.9: What is a friend function? What are the merits and demerits of using friend functions?

Ans:

A function that acts as a bridge among different classes, then it is called friend function.

Merits :

We can access the other class members in our class if we use friend keyword. We can access the members without inheriting the class.

demerits :

Maximum size of the memory will occupied by objects according to the size of friend members.

5.10: State whether the following statements are TRUE or FALSE.

- (a) Data items in a class must always be private.
- (b) A function designed as private is accessible only to member functions of that class.
- (c) A function designed as public can be accessed like any other ordinary functions.
- (d) Member functions defined inside a class specifier become inline functions by default.
- (e) Classes can bring together all aspects of an entity in one place.
- (f) Class members are public by default.
- (g) Friend junctions have access to only public members of a class.
- (h) An entire class can be made a friend of another class.
- (i) Functions cannot return class objects.
- (j) Data members can be initialized inside class specifier.

Ans:

- (a) FALSE
- (b) TRUE
- (c) FALSE

*A function designed as public can be accessed like any other ordinary functions from the member function of same class.

- (d) TRUE
- (e) TRUE
- (f) FALSE
- (g) FALSE
- (h) TRUE
- (i) FALSE
- (j) FALSE

Debugging Exercises

5.1: Identify the error in the following program

```
1 #include <iosream.h>
2 struct Room
3 {
4     int width;
5     int length;
6     void setValue(int w, int l)
7     {
8         width = w;
9         length = l;
```

```

10 }
11);
12void main()
13{
14Room objRoom;
15objRoom.setValue(12, 1,4);
16}

```

Solution:

Void setValue (in w, int l) function must be public.

5.2: Identify the error in the following program

```

1 #include <iosream.h>
2 class Room
3 {
4     int width, int length;
5     void setValue(int w, int h)
6     {
7         width = w;
8         length = h;
9     }
10};
11void main()
12{
13Room objRoom;
14objRoom.width=12;
15}

```

Solution:

Void setValue (int w, int l) function must be public.

5.3: Identify the error in the following program

```

1 #include <iosream.h>
2 class Item
3 {
4     private:
5         static int count;
6     public:
7         Item()
8         {
9             count++;
10}

```

```

11 int getCount()
12 {
13     return count;
14 }
15 int* getCountAddress()
16 {
17     return count;
18 }
19 };
20 int Item::count = 0;
21 void main()
22 {
23     Item objItem1;
24     Item objItem2;
25
26     cout << objItem1.getCount() << '\n';
27     cout << objItem2.getCount() << '\n';
28
29     cout << objItem1.getCountAddress() << '\n';
30     cout << objItem2.getCountAddress() << '\n';
31 }

```

Solution:

```

1
2 int* getCountAddress ( )
3 {
4     return &count;
5 }

```

Note: All other code remain unchanged.

5.4: Identify the error in the following program

```

1 #include <iostream.h>
2 class staticfunction
3 {
4     static int count;
5 public:
6     static void setCountto()
7     {
8         count++;
9     }
10    void displayCount()
11    {
12        cout << count;
13    }

```

```

14);
15int staticFunction::count = 10;
16void main()
17{
18  staticFunction obj1;
19  obj1.setcount(5);
20  staticFunction::setCount();
21  obj1.displayCount();
22}

```

Solution:

setCount () is a void argument type, so here obj1.setCount (5); replace with obj1.setcount();

5.5: Identify the error in the following program

```

1 #include <iostream.h>
2 class Length
3 {
4   int feet;
5   float inches;
6 public:
7   Length()
8   {
9     feet = 5;
10    inches = 6.0;
11  }
12  Length(int f, float in)
13  {
14    feet = f;
15    inches=in;
16  }
17  Length addLength(Length l)
18  {
19
20    l.inches this->inches;
21    l.feet += this->feet;
22    if(l.inches>12)
23    {
24
25      l.inches-=12;
26      l.feet++;
27    }
28    return l;
29  }
30  int getFeet()
31  {
32    return feet;
33  }

```



```

34 float getInches()
35 {
36     return inches;
37 }
38};
39void main()
40{
41
42     Length objLength1;
43     Length objLenngh1(5, 6.5);
44     objLength1 = objLength1.addLength(objLength2);
45     cout << objLenth1.getFeet() << ' ';
46     cout << objLength1.getInches() << ' ';
47}

```

Solution:

Just write the main function like this:

```

1 #include<iostream.h>
2
3 void main()
4 {
5     Length objLength1;
6     Length objLength2(5,6.5);
7     objLength1=objLength1.addLenghth(objLenghth2);
8
9     cout<<objLength1.getFeet()<<" ";
10    cout<<objLength1.getInches()<<" ";
11}

```

5.6: Identify the error in the following program

```

1 #include <iosream.h>
2 class Room
3 void Area()
4 {
5     int width, height;
6     class Room
7     {
8         int width, height;
9         public:
10        void setvalue(int w, int h)
11        {
12            width = w;
13            height = h;
14        }

```

```

15 void displayvalues()
16 {
17     cout << (float)width << ' ' << (float)height;
18 }
19 };
20 Room objRoom1;
21 objRoom1.setValue(12, 8);
22 objRoom1.displayvalues();
23 }
24
25 void main()
26 {
27     Area();
28     Room objRoom2;
29 }

```

Solution:

Undefined structure Room in main () function.

Correction : Change the main () Function as follow:

```

1 void main()
2 {
3     Area();
4 }

```

Programming Exercises

5.1: **Define a class to represent a bank account. Include the following members:**

Data members:

1. Name of the depositor.
2. Account number.
3. Type of account.
4. Balance amount in the account.

Member functions:

1. To assign initial values.
2. To deposit an amount.
3. To withdraw an amount after checking the balance.
4. To display the name and balance.

Write a main program to test the program.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 class bank
4 {
5     char name[40];
6     int ac_no;
7     char ac_type[20];
8     double balance;
9 public:
10    int assign(void);
11    void deposit(float b);
12    void withdraw(float c);
13    void display(void);
14};
15
16int bank::assign(void)
17{
18    float initial;
19    cout<<" You have to pay 500 TK to open your account \n"
20    <<" You have to store at least 500 TK to keep your account active\n"
21    <<"Would you want to open a account????\n"
22    <<" If Yes press 1 \n"
23    <<" If No press 0 : ";
24    int test;
25    cin>>test;
26    if(test==1)
27    {
28        initial=500;
29        balance=initial;
30        cout<<" Enter name ,account number & account type to creat account : \n";
31        cin>>name>>ac_no>>ac_type;
32    }
33    else
34    ;// do nothing
35
36    return test;
37
38}
39void bank::deposit(float b)
40{
41    balance+=b;
42}
43void bank::withdraw(float c)
44{
45    balance-=c;
46    if(balance<500)
47    {
48        cout<<" Sorry your balance is not sufficient to withdraw "<<<<"TK\n"
```

```

49     <<" You have to store at least 500 TK to keep your account active\n";
50     balance+=c;
51 }
52}
53void bank::display(void)
54{
55     cout<<setw(12)<<"Name"<<setw(20)<<"Account type"<<setw(12)<<"Balance"<<endl;
56     cout<<setw(12)<<name<<setw(17)<<ac_type<<setw(14)<<balance<<endl;
57}
58
59int main()
60{
61     bank account;
62
63     int t;
64     t=account.assign();
65     if(t==1)
66     {
67         cout<<" Would you want to deposite: ?"<<endl
68         <<"If NO press 0(zero)"<<endl
69         <<"If YES enter deposite ammount :"<<endl;
70         float dp;
71         cin>>dp;
72         account.deposite(dp);
73         cout<<" Would you want to withdraw : ?"<<endl
74         <<"If NO press 0(zero)"<<endl
75         <<"If YES enter withdrawal ammount :"<<endl;
76         float wd;
77         cin>>wd;
78         account.withdraw(wd);
79         cout<<" see details :"<<endl<<endl;
80         account.display();
81     }
82     else if(t==0)
83     cout<<" Thank you ,see again\n";
84     return 0;
85}

```

output

```

You have to pay 500 TK to open your account
You have to store at least 500 TK to keep your account active
Would you want to open a account????
If Yes press 1
If No press 0 : 0
Thank you ,see again

```

5.2: Write a class to represent a vector (a series of float values). Include member functions to perform the following tasks:

- (a) To create the vector.
- (b) To modify the value of a given element.
- (c) To multiply by a scalar value.
- (d) To display the vector in the form (10, 20, 30 ...)

Write a program to test your class.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 class vector
4 {
5     float *p;
6     int size;
7 public:
8     void creat_vector(int a);
9     void set_element(int i,float value);
10    void modify(void);
11    void multiply(float b);
12    void display(void);
13};
14
15void vector::creat_vector(int a)
16{
17    size=a;
18    p=new float[size];
19}
20void vector::set_element(int i,float value)
21{
22    p[i]=value;
23}
24void vector :: multiply(float b)
25{
26    for(int i=0;i<size;i++)
27        p[i]=b*p[i];
28}
29void vector:: display(void)
30{
31    cout<<"p["<<size<<"] = ( ";
32    for(int i=0;i<size;i++)
33        {
34            if(i==size-1)
```

```

35     cout<<p[i];
36     else
37     cout<<p[i]<<" , ";
38
39 }
40 cout<<")"<<endl;
41}
42
43void vector::modify(void)
44{
45     int i;
46     cout<<" to edit a given element enter position of the element : ";
47     cin>>i;
48     i--;
49     cout<<" Now enter new value of "<<i+1<<"th element : ";
50     float v;
51     cin>>v;
52     p[i]=v;
53     cout<<" Now new contents : "<<endl;
54     display();
55
56     cout<<" to delete an element enter position of the element :";
57     cin>>i;
58     i--;
59
60     for(int j=i;j<size;j++)
61     {
62         p[j]=p[j+1];
63     }
64     size--;
65     cout<<" New contents : "<<endl;
66     display();
67}
68
69int main()
70{
71     vector santo;
72     int s;
73     cout<<" enter size of vector : ";
74     cin>>s;
75     santo.creat_vector(s);
76     cout<<" enter "<<s<<" elements one by one : "<<endl;
77     for(int i=0;i<s;i++)
78     {
79         float v;
80         cin>>v;
81         santo.set_element(i,v);
82     }
83     cout<<" Now contents : "<<endl;
84     santo.display();
85     cout<<" to multiply this vector by a scalar quantity enter this scalar quantity : ";

```



```

86 float m;
87 cin>>m;
88 santo.multiply(m);
89 cout<<" Now contents : "<<endl;
90 santo.display();
91 santo.modify();
92 return 0;
93}

```

output

enter size of vector : 5

enter 5 elements one by one :

11 22 33 44 55

Now contents p[5] = (11 , 22 , 33 , 44 , 55)

to multiply this vector by a scalar quantity enter this scalar quantity : 2

Now contents :

p[5] = (22 , 44 , 66 , 88 , 110)

to edit a given element enter position of the element : 3

Now enter new value of 3th element : 100

Now new contents :

p[5] = (22 , 44 , 100 , 88 , 110)

to delete an element enter position of the element :2

New contents :

p[4] = (22 , 100 , 88 , 110)

5.3: Modify the class and the program of Exercise 5.1 for handling 10 customers.

Solution:

```

1 #include<iostream.h>
2 #include<iomanip.h>
3 #define size 10
4 char *serial[size]={" FIRST "," SECOND "," THIRD "," 4th "," 5th "," 6th "," 7th "," 8th ","

```

```

5  9th ","10th");
6
7  class bank
8  {
9      char name[40];
10     int ac_no;
11     char ac_type[20];
12     double balance;
13 public:
14     int assign(void);
15     void deposit(float b);
16     void withdraw(float c);
17     void displayon(void);
18         void displayoff(void);
19 };
20
21 int bank::assign(void)
22 {
23     float initial;
24     cout<<" You have to pay 500 TK to open your account \n"
25     <<" You have to store at least 500 TK to keep your account active\n"
26     <<"Would you want to open a account????\n"
27     <<" If Yes press 1 \n"
28     <<" If No press 0 : ";
29     int test;
30     cin>>test;
31     if(test==1)
32     {
33         initial=500;
34         balance=initial;
35     cout<<" Enter name ,account number & account type to create account : \n";
36         cin>>name>>ac_no>>ac_type;
37     }
38     else
39     ;// do nothing
40
41     return test;
42
43 }
44 void bank::deposit(float b)
45 {
46     balance+=b;
47 }
48 void bank::withdraw(float c)
49 {
50     balance-=c;
51     if(balance<500)
52     {
53     cout<<" Sorry your balance is not sufficient to withdraw "<<c<<"TK\n"
54     <<" You have to store at least 500 TK to keep your account active\n";
55         balance+=c;

```

```

56 }
57 }
58 void bank::displayon(void)
59 {
60     cout<<setw(12)<<name<<setw(17)<<ac_type<<setw(14)<<balance<<endl;
61 }
62 void bank::displayoff(void)
63 {     cout<<" Account has not created"<<endl; }
64 int main()
65 {
66     bank account[size];
67     int t[10];
68     for(int i=0;i<size;i++)
69     {
70         cout<<" Enter information for "<<serial[i]<<"customer : "<<endl;
71         t[i]=account[i].assign();
72         if(t[i]==1)
73         {
74             cout<<" Would you want to deposit: ?"<<endl
75             <<"If NO press 0(zero)"<<endl
76             <<"If YES enter deposit amount :"<<endl;
77             float dp;
78             cin>>dp;
79             account[i].deposit(dp);
80             cout<<" Would you want to with draw : ?"<<endl
81             <<"If NO press 0(zero)"<<endl
82             <<"If YES enter withdrawal amount :"<<endl;
83             float wd;
84             cin>>wd;
85             account[i].withdraw(wd);
86             cout<<endl<<endl;
87         }
88         else if(t[i]==0)
89             cout<<"Thank you , see again \n";
90     }
91 }
92
93 cout<<" see details :"<<endl<<endl;
94 cout<<setw(12)<<"Name"<<setw(20)<<"Account type"
95     <<setw(12)<<"Balance"<<endl;
96
97 for(i=0;i<size;i++)
98 {
99     if(t[i]==1)
100     account[i].displayon();
101     else if(t[i]==0)
102     account[i].displayoff();
103 }
104 return 0;
}

```

Note: Here we will show output only for **Three** customers. But when you run this program you can see output for 10 customer.

output

Enter information for FIRST customer :

You have to pay 500 TR to open your account

You have to store at least 500 TR to keep your account active Would you want to open a account????

If Yes press 1

If No press 0 : 0

Thank you , see again

Enter information for SECOND customer :

You have to pay 500 TR to open your account

You have to store at least 500 TR to keep your account active Would you want to open a account????

If Yes press 1

If No press 0 : 1

Enter name ,account number & account type to create account :

Robin 11123 saving

Would you want to deposit: ?

If HO press 0(zero)

If YES enter deposit amount :

0

Would you want to with draw : ?

If HO press 0(zero)

If YES enter withdrawal amount :

0

Enter information for 3rd customer :

You have to pay 500 TK to open your account

You have to store at least 500 TK to keep your account active Would you want to open a account????

If Yes press 1

If No press 0 : 1

Enter name ,account number & account type to create account :

Billal 11123 fixed

Would you want to deposit: ?

If HO press 0(zero)

If YES enter deposit amount :

1000000

Would you want to with draw : ?

If HO press 0(zero)

If YES enter withdrawal amount :

100000

see details :

Name	Account type	Balance
------	--------------	---------

Account has not created

Robin	saving	500
Billal	fixed	900500

5.4: Modify the class and the program of Exercise 5.12 such that the program would be able to add two vectors and display the resultant vector. (Note that we can pass objects as function arguments)

Solution:

```

1 #include<iostream.h>
2 #include<iomanip.h>
3 #define size 8
4 class vector
5 {
6     float *p;
7
8 public:
9     void creat_vector(void);
10    void set_element(int i,float value);
11    friend void add(vector v1,vector v2);
12
13};
14void vector::creat_vector(void)
15{
16    p=new float[size];
17}
18void vector::set_element(int i,float value)
19{
20    p[i]=value;
21}
22void add(vector v1,vector v2)
23{
24
25    float *sum;
26    cout<<"sum["<<size<<"] = (";
27    sum= new float[size];
28
29    for(int i=0;i<size;i++)
30    {
31        sum[i]=v1.p[i]+v2.p[i];
32        if(i==size-1)
33            cout<<sum[i];
34        else
35            cout<<sum[i]<<" , ";
36    }
37    cout<<")"<<endl;

```

```

38
39}
40
41int main()
42{
43  vector x1,x2,x3;
44  x1.creat_vector();
45  x2.creat_vector();
46  x3.creat_vector();
47  cout<<" Enter "<<size<<" elements of FIRST vector : ";
48  for(int i=0;i<size;i++)
49  {
50      float v;
51      cin>>v;
52      x1.set_element(i,v);
53  }
54
55  cout<<" Enter "<<size<<" elements of SECOND vector : ";
56  for(i=0;i<size;i++)
57  {
58      float v;
59      cin>>v;
60      x2.set_element(i,v);
61  }
62  add(x1,x2);
63
64  return 0;
65}

```

output

Enter 8 elements of FIRST vector : 4 7 8 2 4 3 2 9

Enter 8 elements of SECOND vector : 1 2 3 4 5 6 7 8

sum[8] = (5 , 9 , 11 , 6 , 9 , 9 , 9 , 17)

5.5: Create two classes DM and DB which store the value of distances. DM stores distances in meters and centimeters and DB in feet and inches. Write a program that can read values for the class objects and add one object of DM with another object of DB.

Use a friend function to carry out the addition operation. The object that stores the results may be a DM object or DB object, depending on the units in which the results are required. The display should be in the format of feet and inches or meters and centimeters depending on the object on display.

Solution:

```

1 #include<iostream.h>
2 #define factor 0.3048
3 class DB;
4 class DM
5 {
6     float d;
7     public:
8     void store(float x){d=x;}
9     friend void sum(DM,DB);
10    void show();
11};
12class DB
13{
14    float d1;
15    public:
16    void store(float y){d1=y;}
17    friend void sum(DM,DB);
18    void show();
19};
20
21void DM::show()
22{
23
24    cout<<"\n Distance = "<<d<<" meter or "<<d*100<<" centimeter\n";
25}
26
27void DB::show()
28{
29
30    cout<<"\n Distance = "<<d1<<" feet or "<<d1*12<<" inches \n";
31}
32void sum(DM m,DB b)
33{
34
35    float sum;
36
37    sum=m.d+b.d1*factor;
38    float f;
39    f=sum/factor;
40    DM m1;
41    DB b1;
42
43    m1.store(sum);
44    b1.store(f);
45
46    cout<<" press 1 to display result in meter\n"
47    <<" press 2 to display result in feet \n"
48    <<" What is your option ? : ";
49    int test;
50    cin>>test;
51

```



```

52     if(test==1)
53         m1.show();
54     else if(test==2)
55         b1.show();
56
57
58}
59
60
61int main()
62{
63     DM dm;
64     DB db;
65     dm.store(10.5);
66     db.store(12.3);
67     sum(dm,db);
68     return 0;
69}

```

output

Press 1 to display result in meter
 Press 2 to display result in feet
 What is your option ? 1
 Distance = 14.24904 meter or 1424.903931 centimeter

Chapter 6

Review Questions

6.1: What is a constructor? Is it mandatory to use constructors in a class?

Ans:A constructor is a 'special' member function whose task is to initialize the object of its class. It is not mandatory to use constructor in a class.

6.2: How do we invoke a constructor function?

Ans:Constructor function are invoked automatically when the objects are created.

6.3: List some of the special properties of the constructor functions.

Ans:Special properties of the constructor functions:

1. They should be declared in the public section.
2. They are invoked automatically when the objects are created.
3. They do not have return type, not even void.
4. They cannot be inherited.
5. Like other C++ functions, they can have default arguments.
6. Constructors cannot be virtual.

6.4: What is a parameterized constructor?

Ans: The constructors that can take arguments are called parameterized constructors.

6.5: Can we have more than one constructors in a class? If yes, explain the need for such a situation.

Ans: Yes, we have when we need to overload the constructor, then we have to do this.

6.6: What do you mean by dynamic initialization of objects? Why do we need to this?

Ans: Initializing value of object during run time is called dynamic initialization of objects. One advantage of dynamic initialization is that we can provide various initialization formats using overloaded constructors.

6.7: How is dynamic initialization of objects achieved?

Ans: Appropriate function of a object is invoked during run-time and thus dynamic initialization of object is achieved.

Consider following constructor:

santo (int p, int q, float r);

santo (int p, int q, int r);

If two int type value and one float type value are passed then sant (int p, int q, float r) is invoked.

If three int type value are passed then santo (int p, into q, int r) is invoked.

6.8: Distinguish between the following two statements:

time T2(T1);

time T2 = T1;

T1 and T2 are objects of time class.

Ans:

time T2 (T1); ==> explicitly called of copy constructor

time T2 = T1; ==> implicitly called of copy constructor.

6.9: Describe the importance of destructors.

Ans: Destructors are important to release memory space for future use.

6.10: State whether the following statements are TRUE or FALSE.

- (a) Constructors, like other member functions, can be declared anywhere in the class.
- (b) Constructors do not return any values.
- (c) A constructor that accepts no parameter is known as the default constructor.
- (d) A class should have at least one constructor.
- (e) Destructors never take any argument.

Ans:

- (a) FALSE
- (b) TRUE
- (c) TRUE
- (d) TRUE
- (e) TRUE

Debugging Exercises

6.1: Identify the error in the following program.

```
1 #include <iostream.h>
2 class Room
3 {
4
5     int length;
6     int width;
7 public:
8     Room(int l, int w=0):
9         width(w),
10        length(l)
11 {
12 }
13};
14void main()
15{
16Room objRoom1;
17Room objRoom2(12, 8);
18}
19
```

20
21Solution: Here there is no default constructor, so object could not be
22written without any argument.

```

23<strong>Correction :</strong>
241
25 Void main ( )
26 {
27     Room Objroom2(12,8);
    }.

```

6.2: Identify the error in the following program.

```

1 #include <iostream.h>
2 class Room
3 {
4
5     int length;
6     int width;
7 public:
8     Room()
9     {
10        length=0;
11        width=0;
12    }
13Room(int value=8)
14 {
15     length = width =8;
16 }
17void display()
18 {
19     cout<<length<< ' ' <<width;
20 }
21};
22void main()
23{
24Room objRoom1;
25objRoom1.display();

```

Solution:Room() and Room(int value=8) Functions are same, so it show Ambiguity error.
Correction : Erase Room() function and then error will not show.

6.3: Identify the error in the following program.

```

1 #include <iostream.h>
2 class Room
3 {
4     int width;
5     int height;

```

```

6 public:
7 void Room()
8 {
9     width=12;
10    height=8;
11 }
12Room(Room& r)
13 {
14    width =r.width;
15    height=r.height;
16    copyConsCount++;
17 }
18void displayConsCount()
19 {
20    cout<<copyConsCount;
21 }
22};
23int Room::copyConsCount = 0;
24void main()
25{
26Room objRoom1;
27Room objRoom2(objRoom1);
28Room objRoom3 = objRoom1;
29Room objRoom4;
30objRoom4 = objRoom3;
31objRoom4.displayConsCount();
32}

```

Solution: Just erase “objRoom4 = objRoom3; invalid to call copy constructor.” for successfully run.

6.4: Identify the error in the following program.

```

1 #include <iostream.h>
2 class Room
3 {
4     int width;
5     int height;
6     static int copyConsCount;
7 public:
8 void Room()
9 {
10    width=12;
11    height=8;
12 }
13Room(Room& r)
14 {
15    width =r.width;

```

```

16     height=r.height;
17     copyConsCount++;
18 }
19 void displayConsCount()
20 {
21     cout<<copyConsCount;
22 }
23};
24 int Room::copyConsCount = 0;
25 void main()
26 {
27     Room objRoom1;
28     Room objRoom2(objRoom1);
29     Room objRoom3 = objRoom1;
30     Room objRoom4;
31     objRoom4 = objRoom3;
32     objRoom4.displayConsCount();
33 }

```

Solution: Same as 6.3 problem solution.

Programming Exercises

6.1: Design constructors for the classes designed in Programming Exercise 5.1 through 5.5 of Chapter 5.

Solution: Study on Constructor and then see solution of chapter 5.

6.2: Define a class String that could work as a user-defined string type. Include constructors that will enable us to create an uninitialized string:

String s1; // string with length 0

And also initialize an object with a string constant at the time of creation like

String s2("Well done!");

Include a function that adds two strings to make a third string. Note that the statement

S2 = s1;

will be perfectly reasonable expression to copy one string to another.

Write a complete program to test your class to see that it does the following tasks:

- (a) Creates uninitialized string objects.
- (b) Creates objects with string constants.
- (c) Concatenates two strings properly.
- (d) Displays a desired string object.

Solution:

```
1 #include
2 #include
3 class string
4 {
5 char *str;
6 int length;
7
8 public:
9 string()
10{
11length = 0;
12str = new char [length + 1] ;
13}
14string(char *s);
15void concat(string &m,string &n);
16string(string &x);
17void display();
18
19};
20string::string(string &x)
21{
22length = x.length + strlen(x.str);
23str = new char[length + 1];
24strcpy(str, x.str);
25
26}
27void string:: concat(string &m,string &n)
28{
29length=m.length+n.length;
30delete str;
31str=new char[length+1];
32strcpy(str,m.str);
33strcat(str,n.str);
34}
35void string:: display()
36{
37cout<<&str<&lt;"\n";
38}
39string::string(char *s)
40{
41length = strlen(s);
42str = new char[length + 1];
43strcpy(str,s);
44}
45
46int main()
47{
48string s1;
```



```

49string s2(" Well done ");
50string s3(" Badly done ");
51s2.display();
52s1.concat(s2,s3);
53s2=s3;
54s2.display();
55s1.display();
56return 0;
57}

```

output

```

Well done
Badly done
Well done Badly done

```

6.3: A book shop maintains the inventory of books that are being sold at the shop. The list includes details such as author, title, price, publisher and stock position. Whenever a customer wants a book, the sales person inputs the title and author and the system searches the list and displays whether it is available or not. If it is not, an appropriate message is displayed. If it is, then the system displays the book details and requests for the number of copies required. If the requested copies are available, the total cost of the requested copies is displayed; otherwise “Required copies not in stock” is displayed. Design a system using a class called books with suitable member functions and constructors. Use new operator in constructors to allocate memory space required.

Solution:

```

1  #include
2  #include
3  #include
4  #include
5  #include
6
7  class book
8  {
9  char **author;
10 char **title;
11 float *price;
12 char **publisher;
13 int *stock_copy;
14 int size;
15
16 public:
17 book();
18 void book_detail(int i);
19 void buy(int i);

```

```

20 int search();
21 };
22
23 book :: book()
24 {
25 size=4;
26 author=new char*[80];
27 title=new char*[80];
28 publisher=new char*[80];
29
30 for(int i=0;i<size;i++)
31 {
32 author[i]=new char[80];
33 title[i]=new char[80];
34 publisher[i]=new char[80];
35 }
36 stock_copy=new int[size];
37 price=new float[size];
38
39 title[0]="object oriented programming with c++";
40 title[1]="programming in ANCI";
41 title[2]="electronic circuit theory";
42 title[3]="computer algorithm";
43
44 author[0]="balagurusamy";
45 author[1]="balagurusamy";
46 author[2]="boyelstade";
47 author[3]="shahani";
48
49 stock_copy[0]=200;
50 stock_copy[1]=150;
51 stock_copy[2]=50;
52 stock_copy[3]=80;
53
54 price[0]=120.5;
55 price[1]=115.75;
56 price[2]=140;
57 price[3]=180.5;
58
59 }
60 void book::book_detail(int i)
61 {
62 cout<<<" *****book detail *****\n";
63 cout<<<setw(12)<<<"Title"<<<setw(25)<<<"Author Name"
64 <<<setw(18)<<<"Stock copy\n";
65 cout<<<setw(15)<<<title[i]<<<setw(16)<<<author[i]<<<setw(15)
66 <<<stock_copy[i]<<<"\n";
67
68 }
69 int book::search()
70 {

```

```

71 char name[80],t[80];
72 cout<<<"Enter author name : ";
73
74 gets(name);
75 cout<<<"and title of book in small letter : ";
76 gets(t);
77
78 int count=-1;
79 int a,b;
80 for(int i=0;i<<size;i++)
81 {
82
83 a=strcmp(name,author[i]);
84 b=strcmp(t,title[i]);
85 if(a==0 && b==0)
86
87 count=i;
88
89 }
90
91 return count;
92 }
93
94 void book::buy(int i)
95 {
96 if(i<<0)
97 cout<<<" This book is not available \n";
98
99 else
100{
101book_detail(i);
102cout<<<" How many copies of this book is required : ? "; int copy; cin>>copy;
103int remaining_copy;
104if(copy<=stock_copy[i])
105{
106remaining_copy=stock_copy[i]-copy;
107float total_price;
108total_price=price[i]*copy;
109cout<<<"Total price = "<<total_price<<" TK\n";
110}
111else
112cout<<<" Sorry your required copies is not available \n";
113}
114}
115
116int main()
117{
118book b1;
119int result;
120
121result=b1.search();

```

```
122b1.buy(result);
123return 0;
124}
```

output

Enter author name : shahani

and title of book in small latter : computer algorithm

*****book detail *****

Title	Author Name	Stock copy
computer algorithm	shahani	80

How many copies of this book is required : ? 78

Total price = 14079 TK

6.4: Improve the system design in Exercise 6.3 to incorporate the following features:

- (a) The price of the books should be updated as and when required. Use a private member function to implement this.**
- (b) The stock value of each book should be automatically updated as soon as a transaction is completed.**
- (c) The number of successful and unsuccessful transactions should be recorded for the purpose of statistical analysis. Use static data members to keep count of transactions.**

Solution:

```
1 #include
2 #include
3 #include
4 #include
5 #include
6
7 class book
8 {
9     static int successful,unsuccessful;
10    char **author;
11    char **title;
12    float *price;
13    char **publisher;
14    int *stock_copy;
15    int size;
16
17    public:
```

```

18 book();
19 void book_detail(int i);
20 void buy(int i);
21 int search();
22 void showtransaction();
23 void showdetail();
24 void edit_price();
25 };
26 int book::successful=0;
27 int book::unsuccessful=0;
28
29 book :: book()
30 {
31 size=5;
32 author=new char*[80];
33 title=new char*[80];
34 publisher=new char*[80];
35
36 for(int i=0;i<&size;i++)
37 {
38 author[i]=new char[80];
39 title[i]=new char[80];
40 publisher[i]=new char[80];
41 }
42 stock_copy=new int[size];
43
44 price=new float[size];
45
46 title[0]="object oriented programming with c++";
47 title[1]="programming in ANCI";
48 title[2]="electronic circuit theory";
49 title[3]="computer algorithm";
50 title[4]="complete solution of balagurusamy(c++)";
51
52 author[0]="balagurusamy";
53 author[1]="balagurusamy";
54 author[2]="boyelstade";
55 author[3]="shahani";
56 author[4]="abdus sattar";
57
58 stock_copy[0]=200;
59 stock_copy[1]=150;
60 stock_copy[2]=50;
61 stock_copy[3]=80;
62 stock_copy[4]=300;
63
64 price[0]=120.5;
65 price[1]=115.75;
66 price[2]=140;
67 price[3]=180.5;
68 price[4]=120;

```

```

69
70 }
71
72 void book::book_detail(int i)
73 {
74 cout<<<" *****book detail *****\n";
75 cout<<<setw(25)<<<"Title"<<<setw(30)<<<"Author Name"
76 <<<setw(18)<<<"Stock copy\n";
77 cout<<<setw(15)<<<title[i]<<<setw(16)<<<author[i]<<<setw(15)
78 <<<stock_copy[i]<<<"\n";
79
80 }
81
82 int book::search()
83 {
84 char name[80],t[80];
85 cout<<<"Enter author name in small letter : ";
86 gets(name);
87 cout<<<" title of book in small letter : ";
88 gets(t);
89
90 int count=-1;
91 int a,b;
92 for(int i=0;i<<size;i++)
93 {
94
95 a=strcmp(name,author[i]);
96 b=strcmp(t,title[i]);
97 if(a==0 && b==0)
98
99 count=i;
100
101 }
102
103return count;
104}
105
106void book::buy(int i)
107{
108if(i<<0)
109{
110cout<<<" This book is not available \n";
111unsuccessful++;
112}
113
114else
115{
116book_detail(i);
117cout<<<" How many copies of this book is required : ? "; int copy; cin<>>copy;
118
119if(copy<=stock_copy[i)

```

```

120{
121stock_copy[i]=stock_copy[i]-copy;
122float total_price;
123total_price=price[i]*copy;
124cout<<<"Total price = "<<<total_price<<<" TK\n";
125successful++;
126}
127else
128{
129cout<<<" Sorry your required copies is not available \n";
130unsuccessful++;
131}
132}
133}
134
135void book::edit_price()
136{
137cout<<<" To edit price ";
138int i;
139i=search();
140cout<<<"Enter new price : "; float p; cin>>>p;
141price[i]=p;
142}
143void book::showdetail()
144{
145cout<<<setw(22)<<<"Title"<<<setw(30)<<<" stock copy "<<<setw(20)
146<<<" Price per book "<<<endl;
147for(int i=0;i<<size;i++)
148{
149cout<<<setw(35)<<<title[i]<<<setw(10)<<<stock_copy[i]
150<<<setw(18)<<<price[i]<<<endl;
151}
152}
153void book::showtransaction()
154{
155cout<<<setw(22)<<<"Successful transaction"<<<setw(34)
156<<<"unsuccessful transaction "<<<endl<<<setw(10)
157<<<successful<<<setw(32)<<<unsuccessful<<<endl;
158}
159
160int main()
161{
162book b1;
163int result;
164
165result=b1.search();
166b1.buy(result);
167b1.showdetail();
168b1.showtransaction();
169b1.edit_price();
170cout<<<"*****details after edit price

```



```

171 *****" &lt;&lt;endl;
172 b1.showdetail();
173
174 return 0;
175 }

```

output

Enter author name in small letter : abdu s attar

title of book in small letter : complete solution of balagurusamy(c++)

*****book detail *****

Title	Author Name	Stock copy
complete solution of balagurusamy(c++)	abdu s attar	300

How many copies of this book is required : ? 100

Total price = 12000 TK

Title	stock copy	Price per book
object oriented programming with c++	200	120.5
programming in ANCI	150	115.75
electronic circuit theory	50	140
computer algorithm	80	180.5
complete solution of balagurusamy(c++)	200	120

Successful transaction	unsuccessful transaction
1	0

To edit price Enter author name in small letter : shahani

title of book in small letter : computer algorithm

Enter new price : 200

*****details after edit price*****

Title	stock copy	Price per book
-------	------------	----------------

object oriented programming with c++	200	120.5
programming in ANCI	150	115.75
electronic circuit theory	50	140
computer algorithm	80	200
complete solution of balagurusamy(c++)	200	120

Chapter 7

Review Questions

7.1: What is operator overloading?

Ans: The mechanism of giving special meaning to an operator is known as operator overloading.

7.2: Why is it necessary to overload an operator?

Ans: We can almost create a new language of our own by the creative use of the function and operator overloading techniques.

7.3: What is an operator function? Describe the syntax of an operator function.

Ans: 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. By which function this is done, is called operator function. Syntax of operator function:

```
return type class name : : operator OP (argument list)
{
function body // task defined
}
```

7.4: How many arguments are required in the definition of an overloaded unary operator?

Ans: No arguments are required.

7.5: A class alpha has a constructor as follows:

alpha(int a, double b);

Can we use this constructor to convert types?

Ans: No. The constructors used for the type conversion take a single argument whose type is to be converted.

7.6: What is a conversion function How is it created Explain its syntax.

Ans: C++ allows us to define an overloaded casting operator that could be used to convert a class type data to a basic type. This is referred to conversion function.

Syntax:

```
Operator type name ( )  
{  
    (Function Statements)  
}
```

7.7: A friend function cannot be used to overload the assignment operator =. Explain why?

Ans: A friend function is a non-member function of the class to which it has been defined as friend. Therefore it just uses the functionality (functions and data) of the class. So it does not consist the implementation for that class. That's why it cannot be used to overload the assignment operator.

7.8: When is a friend function compulsory? Give an example.

Ans: When we need to use two different types of operands for a binary operator, then we must use friend function.

Example:

A = B + 2;

or

A = B * 2;

is valid

But A = 2 + B

or

A = 2 * B will not work.

Because the left hand operand is responsible for invoking the member function. In this case friend function allows both approaches.

7.9: We have two classes X and Y. If a is an object of X and b is an object of Y and we want to say a = b; What type of conversion routine should be used and where?

Ans: We have to use one class to another class type conversion. The type-conversion function to be located in the source class or in the destination class.

7.10: State whether the following statements are TRUE or FALSE.

- (a) Using the operator overloading concept, we can change the meaning of an operator.
- (b) Operator overloading works when applied to class objects only.
- (c) Friend functions cannot be used to overload operators.
- (d) When using an overloaded binary operator, the left operand is implicitly passed to the member function.
- (e) The overloaded operator must have at least one operand that is user-defined type.
- (f) Operator functions never return a value.
- (g) Through operator overloading, a class type data can be converted to a basic type data.
- (h) A constructor can be used to convert a basic type to a class type data.

Ans:

- (a) FALSE
- (b) TRUE
- (c) FALSE
- (d) FALSE
- (e) TRUE
- (f) FALSE
- (g) TRUE
- (h) TRUE

Debugging Exercises

7.1: Identify the error in the following program.

```
#include <iostream.h>
class Space
{
    int mCount;
public:
    Space()
    {
        mCount = 0;
    }

    Space operator ++()
    {
        mCount++;
        return Space(mCount);
    }
}
```

```

    }
};
void main()
{
    Space objSpace;
    objSpace++;
}

```

Solution: The argument of Space() function is void type, so when this function is called there are no argument can send to it. But 'mCount' argument is sending to Space() function through return space(mCount); Statement.

Here return space (mCount); replaced by return space();

7.2: Identify the error in the following program.

```

#include <iostream.h>
enum WeekDays
{
    mSunday'
    mMonday,
    mtuesday,
    mWednesday,
    mThursday,
    mFriday,
    mSaturday
};
bool op==(WeekDays& w1, WeekDays& w2)
{
    if(w1== mSunday && w2==mSunday)
        return 1;
    else if(w1==mSunday && w2==mSunday)
        return 1;
    else if(w1==mSunday && w2==mSunday)
        return 1;
    else if(w1==mSunday && w2==mSunday)
        return 1;
    else if(w1==mSunday && w2==mSunday)
        return 1;
    else if(w1==mSunday && w2==mSunday)
        return 1;
    else if(w1==mSunday && w2==mSunday)
        return 1;
    else
        return 0;
}
void main()
{
    WeekDays w1 = mSunday, w2 = mSunday;
}

```

```

    if(w1==w2)
    cout<<"Same day";
    else
    cout<<"Different day";
}

```

Solution: bool OP == (WeekDays & w1, WeekDays & w2) replaced by bool operator == (Weekdays & w1, WeekDays & w2). All other code will remain same.

7.3: Identify the error in the following program.

```

#include <iostream.h>
class Room
{
    float mWidth;
    float mLength;
public:
    Room()
    {
    }
    Room(float w, float h)
        :mWidth(w), mLength(h)
    {
    }
    operator float ()
    {
        return (float)mWidth * mLength;
    }

    float getWidth()
    {
    }
    float getLength()
    {
        return mLength;
    }
};

void main()
{
    Room objRoom1(2.5, 2.5)
    float fTotalArea;
    fTotalArea = objRoom1;
    cout<< fTotalArea;
}

```

Solution: The float getWidth() function return float type data, but there is no return statement in getWidth() function. So it should write as follows.

```
float getWidth()
{
    return mWidth;
}
```

All other code will remain unchanged.

Programming Exercises

7.1: Crate a class FLOAT that contains one float data member. Overload all the four arithmetic operators so that they operate on the objects of FLOAT.

Solution:

```
1 #include<iostream.h>
2
3 class FLOAT
4 {
5     float data;
6     public:
7     FLOAT(){};
8     FLOAT(float d)
9     { data=d;}
10    FLOAT operator+(FLOAT f1);
11    FLOAT operator-(FLOAT f2);
12    FLOAT operator*(FLOAT f3);
13    FLOAT operator/(FLOAT f4);
14    void display();
15};
16FLOAT FLOAT::operator+(FLOAT f1)
17{
18    FLOAT temp;
19    temp.data=data+f1.data;
20    return (temp);
21}
22FLOAT FLOAT::operator-(FLOAT f2)
23{
24    FLOAT temp;
25    temp.data=data-f2.data;
26    return temp;
27}
28FLOAT FLOAT::operator*(FLOAT f3)
```



```

29{
30    FLOAT temp;
31    temp.data=data*f3.data;
32    return temp;
33}
34FLOAT FLOAT::operator/(FLOAT f4)
35{
36    FLOAT temp;
37    temp.data=data/f4.data;
38    return (temp);
39}
40void FLOAT:: display()
41{
42    cout<<data<<"\n";
43}
44int main()
45{
46    FLOAT F1,F2,F3,F4,F5,F6;
47    F1=FLOAT(2.5);
48    F2=FLOAT(3.1);
49    F3=F1+F2;
50    F4=F1-F2;
51    F5=F1*F2;
52    F6=F1/F2;
53    cout<<" F1 = ";
54    F1.display();
55    cout<<" F2 = ";
56    F2.display();
57    cout<<" F1+F2 = ";
58    F3.display();
59    cout<<" F1-F2 = ";
60    F4.display();
61    cout<<" F1*F2 = ";
62    F5.display();
63    cout<<" F1/F2= ";
64    F6.display();
65    return 0;
66}

```

output

```

F1 = 2.5
F2 = 3.1
F1+F2 = 5.6
F1-F2 = -0.6
F1*F2 = 7.75
F1/F2= 0.806452

```

7.2: Design a class Polar which describes a point in the plane using polar coordinates radius and angle. A point in polar coordinates is shown below figure 7.3

Use the overload + operator to add two objects of Polar.

Note that we cannot add polar values of two points directly. This requires first the conversion of points into rectangular coordinates, then adding the respective rectangular coordinates and finally converting the result back into polar coordinates. You need to use the following trigonometric formula:

$$x = r * \cos(a);$$

$$y = r * \sin(a);$$

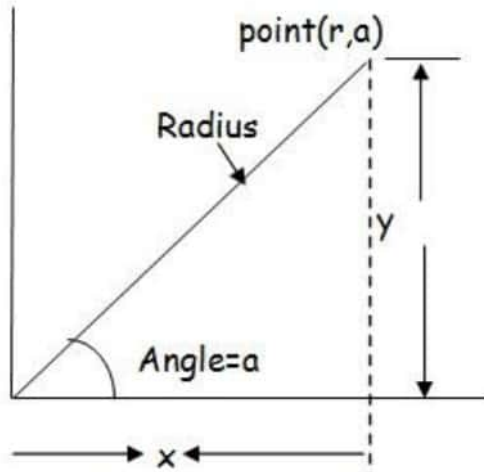


fig: polar coordinates of a point

$$y = r * \sin(a);$$

$$a = \text{atan}(y/x); // \text{arc tangent}$$

$$r = \text{sqrt}(x*x + y*y);$$

Solution:

```
1 #include<iostream.h>
2 #include<math.h>
3 #define pi 3.1416
4 class polar
5 {
6     float r,a,x,y;
7     public:
8     polar(){};
9     polar(float r1,float a1);
10    polar operator+(polar r1);
11    void display(void);
12};
13
14polar :: polar(float r1,float a1)
15{
16    r=r1;
17    a=a1*(pi/180);
```

```

18     x=r*cos(a);
19     y=r*sin(a);
20}
21
22polar polar :: operator+(polar r1)
23{
24     polar R;
25
26     R.x=x+r1.x;
27     R.y=y+r1.y;
28     R.r=sqrt(R.x*R.x+R.y*R.y);
29     R.a=atan(R.y/R.x);
30
31     return R;
32}
33
34void polar::display()
35{
36     cout<<"radius = "<<r<<"\n angle = "<<a*(180/pi)<<"\n";
37}
38
39int main()
40{
41     polar p1,p2,p3;
42     float r,a;
43     cout<<" Enter radius and angle : ";
44     cin>>r>>a;
45     p1=polar(r,a);
46     p2=polar(8,45);
47     p3=p1+p2;
48     cout<<" p1 : \n";
49     p1.display();
50     cout<<" p2 : \n ";
51     p2.display();
52     cout<<" p3 : \n ";
53     p3.display();
54     return 0;
55}

```

output

```

Enter radius and angle : 10 45
P1:
radius = 10
angle = 44.999998
P2 :
radius = 8
angle = 44.999998
P3 :
radius = 18
angle = 44.999998

```

7.3: Create a class MAT of size m * n. Define all possible matrix operations for MAT type objects.

Solution:

```
1  #include<iostream.h>
2  #include<iomanip.h>
3
4  class mat
5  {
6      float **m;
7      int rs,cs;
8      public:
9      mat(){}
10     void creat(int r,int c);
11     friend istream & operator >>(istream &,mat &);
12     friend ostream & operator <<(ostream &,mat &);
13     mat operator+(mat m2);
14     mat operator-(mat m2);
15     mat operator*(mat m2);
16 };
17
18 void mat::creat(int r,int c)
19 {
20     rs=r;
21     cs=c;
22     m=new float *[r];
23     for(int i=0;i<r;i++)
24         m[i]=new float[cs];
25 }
26
27 istream & operator>>(istream &din, mat &a)
28 {
29     int r,c;
30     r=a.rs;
31     c=a.cs;
32     for(int i=0;i<r;i++)
33     {
34         for(int j=0;j<c;j++)
35         {
36             din>>a.m[i][j];
37         }
38     }
39     return (din);
40 }
41 ostream & operator<<(ostream &dout,mat &a)
42 {
43     int r,c;
```

```

44     r=a.rs;
45     c=a.cs;
46     for(int i=0;i<r;i++)
47     {
48         for(int j=0;j<c;j++)
49         {
50             dout<<setw(5)<<a.m[i][j];
51         }
52         dout<<"\n";
53     }
54 return (dout);
55 }
56 mat mat::operator+(mat m2)
57 {
58     mat mt;
59     mt.creat(rs,cs);
60     for(int i=0;i<rs;i++)
61     {
62         for(int j=0;j<cs;j++)
63         {
64             mt.m[i][j]=m[i][j]+m2.m[i][j];
65         }
66     }
67     return mt;
68 }
69
70 mat mat::operator-(mat m2)
71 {
72     mat mt;
73     mt.creat(rs,cs);
74     for(int i=0;i<rs;i++)
75     {
76         for(int j=0;j<cs;j++)
77         {
78             mt.m[i][j]=m[i][j]-m2.m[i][j];
79         }
80     }
81     return mt;
82 }
83
84 mat mat::operator*(mat m2)
85 {
86     mat mt;
87     mt.creat(rs,m2.cs);
88
89     for(int i=0;i<rs;i++)
90     {
91         for(int j=0;j<m2.cs;j++)
92         {
93             mt.m[i][j]=0;
94             for(int k=0;k<m2.rs;k++)

```

```

95         mt.m[i][j]+=m[i][k]*m2.m[k][j];
96     }
97 }
98
99     return mt;
100 }
101 int main()
102 {
103     mat m1,m2,m3,m4,m5;
104     int r1,c1,r2,c2;
105     cout<<" Enter first matrix size : ";
106     cin>>r1>>c1;
107     m1.creat(r1,c1);
108     cout<<"m1 = ";
109     cin>>m1;
110     cout<<" Enter second matrix size : ";
111     cin>>r2>>c2;
112     m2.creat(r2,c2);
113     cout<<"m2 = ";
114     cin>>m2;
115     cout<<" m1:"<<endl;
116     cout<<m1;
117     cout<<" m2: "<<endl;
118     cout<<m2;
119     cout<<endl<<endl;
120     if(r1==r2 && c1==c2)
121     {
122         m3.creat(r1,c1);
123         m3=m1+m2;
124         cout<<" m1 + m2: "<<endl;
125         cout<<m3<<endl;
126         m4.creat(r1,c1);
127
128         m4=m1-m2;
129         cout<<" m1 - m2:"<<endl;
130         cout<<m4<<endl<<endl;
131
132     }
133     else
134         cout<<" Summation & subtraction are not possible n"<<endl
135         <<"Two matrices must be same size for summation & subtraction "<<endl<<endl;
136     if(c1==r2)
137     {
138         m5=m1*m2;
139         cout<<" m1 x m2: "<<endl;
140         cout<<m5;
141     }
142     else
143         cout<<" Multiplication is not possible "<<endl
144         <<" column of first matrix must be equal to the row of second matrix ";
145     return 0;

```

146}

output

Enter first matrix size : 2 2

m1 =

1 2

3 4

Enter second matrix size : 2 2

m2 =

5 6

7 8

m1 =

1 2

3 4

m2 =

5 6

7 8

m1+m2:

6 8

10 12

m1-m2:

-4 -4

-4 -4

m1 x m2:

19 22

7.4: Define a class String. Use overload == operator to compare two strings.

Solution:

```
1 #include<iostream.h>
2 #include<string.h>
3 #include<stdio.h>
4
5 class string
6 {
7     char str[1000];
8     public:
9     void input(){ gets(str);}
10    int operator==(string s2);
11};
12int string::operator==(string s2)
13{
14    int t= strcmp(str,s2.str);
15    if(t==0)
16        t=1;
17    else
18        t=0;
19    return t;
20}
21
22int main()
23{
24
25    char st1[1000],st2[1000];
26    string s1,s2;
27    cout<<" Enter 1st string : ";
28    s1.input();
29    cout<<" enter 2nd string : ";
30    s2.input();
31
32    if(s1==s2)
33        cout<<" Two strings are equal ";
34    else
35        cout<<" Two string are not equal ";
36    return 0;
37}
```

output

Enter 1st string : our sweetest songs tel our saddest thought
enter 2nd string : a burning desire lead to success.
Two string are not equal

7.5: Define two classes Polar and Rectangle to represent points in the polar and rectangle systems. Use conversion routines to convert from one system to the other.

Solution:

```
1 #include<iostream.h>
2 #include<math.h>
3 #define pi 3.1416
4 class conversion_point
5 {
6     float x,y,r,theta;
7     public:
8     void set_xy();
9     void set_r_theta();
10    void show_xy();
11    void show_r_theta();
12    void conversion(int t);
13};
14 void conversion_point::set_xy()
15{
16    cout<<"Enter the value of x & y : ";
17    cin>>x>>y;
18}
19 void conversion_point::set_r_theta()
20{
21    cout<<"Enter the value of r & theta :";
22    cin>>r>>theta;
23    theta=(pi/180)*theta;
24}
25
26 void conversion_point::show_xy()
27{
28    cout<<" CERTECIAN FORM :\n"
29        <<" x = "<<x<<"\n"
30        <<" y = "<<y<<"\n";
31}
32void conversion_point::show_r_theta()
33{
34    cout<<" POLAR FORM :\n"
35        <<" r = "<<r<<"\n"
36        <<" theta = "<<(180/pi)*theta<<" degree \n";
37}
38
39void conversion_point::conversion(int t)
```

```

40{
41    if(t==1)
42    {
43        r=sqrt(x*x+y*y);
44
45        if(x!=0)
46        {
47            theta=atan(y/x);
48            show_r_theta();
49        }
50
51        else
52        {
53            cout<<" POLAR FORM :\n"
54                <<" r = "<<r<<"\n"
55                <<" theta = 90 degree\n";
56        }
57
58    }
59    else if(t==2)
60    {
61        x=r*cos(theta);
62        y=r*sin(theta);
63        show_xy();
64    }
65}
66
67int main()
68{
69    conversion_point santo;
70    int test;
71    cout<<" press 1 to input certecian point \n"
72        <<" press 2 to input polar point \n "
73        <<" what is your input ? : ";
74    cin>>test;
75    if(test==1)
76        santo.set_xy();
77    else if(test==2)
78        santo.set_r_theta();
79    santo.conversion(test);
80
81    return 0;
82}

```

output

```

Press 1 to input certecian point
Press 2 to input polar point
what is your input ? 1
Enter the value of x & y : 4 5
POLAR FORM :

```

r = 6.403124
theta = 51.340073 degree

Chapter 8

Review Questions

8.1: What does inheritance mean in C++?

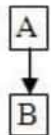
Ans: The mechanism of deriving a new class from an old one is called inheritance. The old class is referred to as the base class and the new one is called derived class.

8.2: What are the different forms of inheritance? Give an example for each.

Ans: Different forms of inheritance:

1. Single inheritance : Only one derived class inherited from only one base class is called single inheritance.

Example: Let A is a base class
and B is a new class derived from A

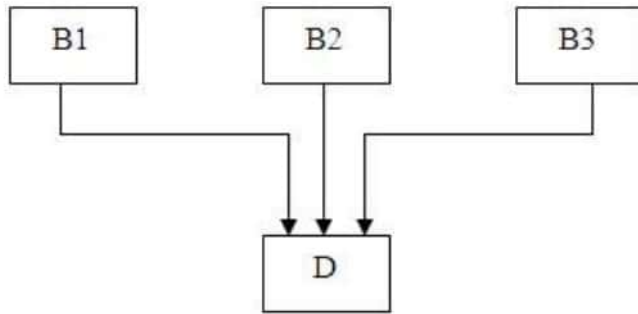


This is written in program as following:

```
class A {.....};  
class B : Public A {.....};
```

2. Multiple inheritance : A class can inherit the attributes of two or more classes. This is known as multiple inheritance.

Example :



Class D : visibilityB1, visibility.B2, visibility B3

```
{
    (Body of D)
}
```

* visibility may be public or private.

3.Multilevel inheritance : It a class is derived from a base class, and another class is derived from this derived class and so on, then it is called multilevel inheritance.

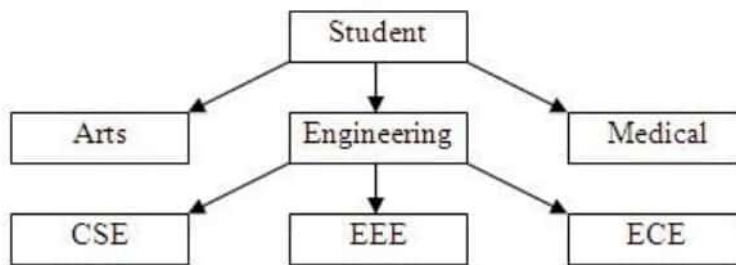


Example :

```
Class A { ..... };
Class B : Public A { ..... };
Class C : Private B { ..... };
```

4.Hierarchical inheritance: It the inheritance follows the hierarchical design of a program, then it is called hierarchical inheritance.

Example :



This design is implemented in program as follows:

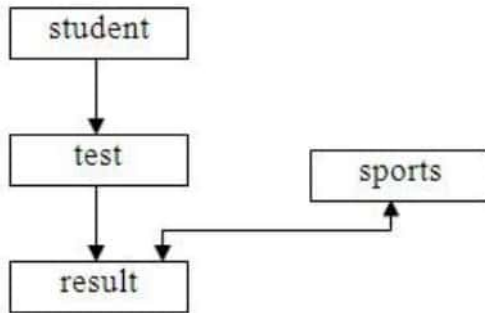
```
Class student { ..... }; // base class,
Class Arts : Public student { ..... };
Class Medical : Public student { ..... };
Class Engineering : Public student { ..... };
Class CSE : Public Engineering { ..... };
```

```
Class EEE : Public Engineering { ..... };  
Class ECE : Public Engineering { ..... };
```

* here all inheritance are considered as public you can private inheritance also. as you wish.

5. Hybrid inheritance: When multi level and multiple inheritances are applied to an inheritance, then it is called Hybrid inheritance.

Example :



In program :

```
Class student {..... };  
Class test : public student {..... };  
Class result : public test {..... };  
Class result : public sports {..... };
```

8.3: Describe the syntax of the single inheritance in C++.

Ans: Syntax of single inheritance:

Class Derived name : visibility mode Base_class name

```
{  
    Body of derived class  
};
```

* visibility mode may be public or private.
or protected

8.4: We know that a private member of a base class is not inheritable. Is it anyway possible for the objects of a derived class to access the private members of the base class? If yes, how? Remember, the base class cannot be modified.

Ans: Yes. It is possible for the objects of derived class to access the private member of the base class by a member function of base class which is public. The following example explains this :

```
#include<iostream.h>  
class B  
{
```

```

    int a; // a is private that can not be inherited.
    public:
        int get_a();
        void set_a();
};
class D:public B
{
    int b;
    public:
        void display_a();
};

void D :: display_a()
{
    cout<<" a = "<<get_a()<<"\n"; // a is accessed by member function get_a().
}

void B :: set_a()
{
    a=156271;
}

int B :: get_a()
{
    return a;
}

void main()
{
    D d;
    d.set_a();
    d.display_a();
}

```

8.5: How do the properties of the following two derived classes differ?

- (a) class D1: private B(// ...);
- (b) class D2: public B(//...);

Ans:(a) Private member of B can not be inherited in D1 Protected member of B become private in D1 public member of B become private in D1.

(b) Private member of B can not be inherited in D2 Protected member of B remains protected in D2 Public member of B remains public in D2

8.6: When do we use the protected visibility specifier to a class member?

Ans:When we want access a data member by the member function within its class and by the

member functions immediately derived from it, then we use visibility modifier protected.

8.7: Describe the syntax of multiple inheritance. When do we use such an inheritance?

Ans: **Syntax :**

```
Class D : Visibility B1, Visibility B2, ... ..  
{  
    (Body of D)  
}
```

Then we want of combine several feature to a single class then we use multiple inheritance.

8.8: What are the implications of the following two definitions?

- (a) class A: public B, public C(//....);
- (b) class A: public C, public B(//....);

Ans:Two are same.

8.9: WWhat is a virtual base class?

Ans:Whey multiple paths between a bass class and a derived class are exist then this base class is virtual base class. A base class can be made as virtual just adding 'virtual' keyword before this base class.

8.10: When do we make a class virtual?

Ans:To avoid the duplication of inherited members due to multiple paths between base and derived classes we make base class virtual.

8.11: What is an abstract class?

Ans:An abstract class is one that is not used to create objects.

8.12: In what order are the class constructors called when a derived class object is created?

Ans:According to the order of derived class header lines

8.13: Class D is derived from class B. The class D does not contain any data members of its own. Does the class D require constructors? If yes, why?

Ans:D does not require any construct or because a default constructor is always set to class by default.

8.14: What is containership? How does it differ from inheritance?

Ans:Containership is another word for composition. That is, the HAS-A relationship where A has-a member that is an object of class B.

Difference : In inheritance the derived class inherits the member data and functions from the base class and can manipulate base public/protected member data as its own data. By default a program which constructs a derived class can directly access the public members of the base class as well. The derived class can be safely down cast to the base class, because the derived is-a” base class.

Container : a class contains another object as member data. The class which contains the object cannot access any protected or private members of the contained class(unless the container it was made a friend in the definition of the contained class).The relationship between the container and the contained object is “has-a” instead of “is-a”

8.15: Describe how an object of a class that contains objects of other classes created?

Ans:By inheriting an object can be created that contains the objects of other class.

Example :

```
class A
{
    int a;
    public:
        void dosomething();
};
```

```
class B: class A
{
    int b;
    public:
        void donothing();
};
```

Now if object of B is created ; then it contains:

1. void dosomething ();
2. int b;
3. void donothing ();

8.16: **State whether the following statements are TRUE or FALSE:**

- (a) Inheritance helps in making a general class into a more specific class.
- (b) Inheritance aids data hiding.
- (c) One of the advantages of inheritance is that it provides a conceptual framework.
- (d) Inheritance facilitates the creation of class libraries.
- (e) Defining a derived class requires some changes in the base class.
- (f) A base class is never used to create objects.
- (g) It is legal to have an object of one class as a member of another class.
- (h) We can prevent the inheritance of all members of the base class by making base class virtual in the definition of the derived class.

Ans:

- (a) TRUE
- (b) FALSE
- (c) TRUE
- (d) TRUE
- (e) FALSE
- (f) TRUE
- (g) TRUE
- (h) FALSE

Debugging Exercises

8.1: **Identify the error in the following program.**

```
#include <iostream.h>
class Student {
    char* name;
    int rollNumber;
public:
    Student() {
        name = "AlanKay";
        rollNumber = 1025;
    }

    void setNumber(int no) {
        rollNumber = no;
    }
    int getRollNumber() {
        return rollNumber;
    }
};

class AnualTest: Student {
    int mark1, mark2;
```

```

public:
    AnualTest(int m1, int m2)
        :mark1(m1), mark2(m2) {
    }
    int getRollNumber() {
        return Student::getRollNumber();
    }
};
void main()
{
    AnualTest test1(92 85);
    cout<< test1.getRollNumber();
}

```

Solution: Constructor and Private (data & function) can not be inherited.

8.2: Identify the error in the following program.

```

#include <iostream.h>;
class A
{
public:
    A()
    {

        cout<< "A";
    }
};

class B: public A
{
public:
    B()
    {
        cout<< "B";
    }
};

class C: public B
{
public:
    C()
    {
        cout << "C";
    }
};

class D
{
public:

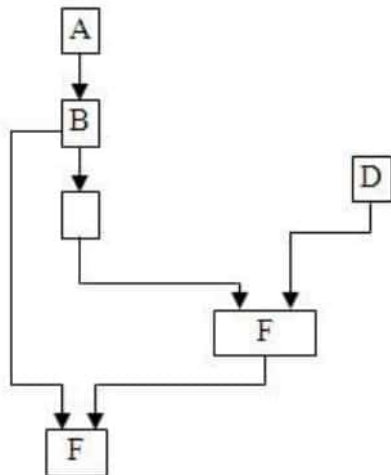
```

```

    D()
    {
        cout << "D";
    }
};
class E: public C, public D
{
public:
    E()
    {
        cout << "D";
    }
};
class F: B, virtual E
{
public:
    F()
    {
        cout << "F";
    }
};
void main()
{
    F f;
}

```

Solution: The inheritance can be represented as follows :



Here B is virtual, but not E.

8.3: Identify the error in the following program.

```
#include <iostream.h>;
```

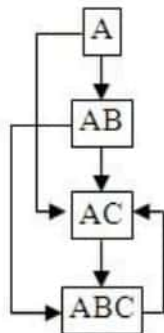
```

class A
{
    int i;
};

class AB: virtual A
{
    int j;
};
class AC: A, ABAC
{
    int k;
};
class ABAC: AB, AC
{
    int l;
};
void main()
{
    ABAC abac;
    cout << "sizeof ABAC:" << sizeof(abac);
}

```

Solution: The inheritance can be represented as follows:



Class AC: A, Here there is no identification of ABAC. If we write class ABAC; after #include it will not show any error message.

8.4: Find errors in the following program. State reasons.

```

// Program test
#include <iostream.h>
class X
{
    private:
        int x1;
    Protected:

```

```

        int x2;
    public:
        int x3;
};

class Y: public X
{
    public:
        void f()
        {
            int y1,y2,y3;
            y1 = x1;
            y2 = x2;
            y3 = x3;
        }
};

class Z: X
{
    public:
        void f()
        {
            int z1,z2,z3;
            z1 = x1;
            z2 = x2;
            z3 = x3;
        }
};

main()
{
    int m,n,p;
    Y y;
    m = y.x1;
    n = y.x2;
    p = y.x3;
    Z z;
    m = z.x1;
    n = z.x2;
    p = z.x3;
}

```

Solution: Here x1 is private, so x1 cannot be inherited.

y1 = x1; is not valid

z1 = x1; is not valid

m = y, x1; is not valid

m = z, x1; is not valid

8.5: Debug the following program.

```

// Test program
#include <iostream.h>
class B1
{
    int b1;
    public:
    void display();
    {
        cout << b1 << "\n";
    }
};

class B2
{
    int b2;
    public:
    void display();
    {
        cout << b2 << "\n";
    }
};

class D: public B1, public B2
{
    //nothing here
};

main()
{
    D d;
    d.display()
    d.B1::display();
    d.B2::display();
}

```

Programming Exercises

8.1: Assume that a bank maintains two kinds of accounts for customers, one called as savings and the other as current account. The savings account provides compound interest and withdrawal facilities but no cheque book facility. The current account provides cheque book facility but no interest. Current account holders should also maintain a minimum balance and if the balance falls below this level a service charge is imposed.

Create a class account that stores customer name, account number and type of account. From this derive the classes cur_acct and sav_acct to make them more specific to their requirements. Include necessary member functions in order to achieve the following tasks:

- (a) Accept the deposit from a customer and update the balance.
- (b) Display the balance.
- (c) Compute and deposit interest.
- (d) Permit withdrawal and update the balance.
- (e) Check for the minimum balance, impose penalty, necessary and update the balance.

Do not use any constructors. Use member functions to initialize class members.

Solution:

```
1  #include<iostream.h>
2  #include<stdio.h>
3  #include<string.h>
4  #include<math.h>
5  #define minimum 500
6  #define service_charge 100
7  #define r 0.15
8
9  class account
10 {
11     protected:
12         char name[100];
13         int ac_number;
14         char ac_type[100];
15     public:
16         void creat( char *t);
17 };
18
19 void account::creat(char *t)
20 {
21
22     cout<<" Enter customer name :";
23     gets(name);
24     strcpy(ac_type,t);
25     cout<<" Enter account number :";
26     cin>>ac_number;
27 }
28 class cur_acct: public account
29 {
30     private:
31         float balance;
32     public:
33         void deposit(float d);
34         void withdraw(float w);
35         void display();
36 };
37 void cur_acct::deposit(float d)
38 {
39     balance=d;
40 }
41
42 void cur_acct::withdraw(float w)
43 {
44     if(balance<w)
45         cout<<" sorry your balance is less than your withdrawal amount \n";
46     else
```

```

47     {
48         balance-=w;
49         if(balance<minimum)
50 cout<<"\n your current balance is :"<<balance<<" which is less than"<<minimum<<"\n
51 your account is discharged by "<<service_charge<<"TK \n"<<" You must store
52 "<<minimum<<"TK to avoid discharge \n "<<" Do you want to withdraw ? press 1 for yes
53 press 0 for no \n"<<" what is your option ?";
54         int test;
55         cin>>test;
56         if(test==0)
57             balance+=w;
58     }
59
60 }
61
62 void cur_acct::display()
63 {
64     cout<<"\n Now balance = "<<balance<<"\n";
65 }
66 class sav_acct:public account
67 {
68     float balance;
69     int d,m,y;
70     public:
71     void deposit(float d);
72     void withdraw(float w);
73     void display();
74     void set_date(int a,int b,int c){d=a;m=b;y=c;}
75     void interest();
76 };
77
78 void sav_acct::deposit(float d)
79 {
80     int x,y,z;
81     cout<<" Enter today's date(i,e day,month,year) : ";
82     cin>>x>>y>>z;
83     set_date(x,y,z);
84     balance+=d;
85 }
86
87 void sav_acct::withdraw(float w)
88 {
89     if(balance<w)
90     cout<<" sorry your balance is less than your withdrawal amount \n";
91     else
92     {
93         balance-=w;
94
95         if(balance<minimum)
96         {
97 cout<<"\n your current balance is :"<<balance<<" which is less than"<<minimum<<"\n

```

```

98 your account is discharged by "<<service_charge<<"TK \n"<<" You must store
99 "<<minimum<<"TK to avoid discharge \n "<<" Do you want to withdraw ? press 1 for yes
100press 0 for no \n"<<" what is your option ?";
101
102         int test;
103         cin>>test;
104         if(test==0)
105             balance+=w;
106     }
107 }
108
109}
110void sav_acct::display()
111{
112     cout<<"\n Now balance = "<<balance;
113}
114void sav_acct::interest()
115{
116     int D[12]={31,28,31,30,31,30,31,31,30,31,30,31};
117     int d1,y1,m1;
118     cout<<" Enter today's date :(i,e day,month,year) ";
119     cin>>d1>>m1>>y1;
120     int iday,fday;
121     iday=d;
122     fday=d1;
123     for(int i=0;i<m1;i++)
124     {
125         fday+=D[i];
126     }
127     for(i=0;i<m;i++)
128     {
129         iday+=D[i];
130     }
131     int tday;
132     tday=fday-iday;
133     float ty;
134     ty=float(tday)/365+y1-y;
135     float intrst;
136
137     intrst=ty*r*balance;
138     cout<<" Interest is : "<<intrst<<"\n";
139     balance+=intrst;
140}
141
142int main()
143{
144     sav_acct santo;
145     santo.creat("savings");
146     float d;
147     cout<<" Enter your deposit amount : ";
148     cin>>d;

```

```

149     santo.deposit(d);
150     santo.display();
151     int t;
152     cout<<"\n press 1 to see your interest : \n"
153         <<" press 0 to skip : ";
154
155     cin>>t;
156
157     if(t==1)
158         santo.interest();
159
160     cout<<"\n Enter your withdrawal amount :";
161     float w;
162     cin>>w;
163     santo.withdraw(w);
164     santo.display();
165     return 0;
166 }

```

output

```

Enter customer name :Rimo
Enter account number :10617
Enter your deposit amount : 10000
Enter today's date(i,e day,month,year) : 13 7 2010

```

```

Now balance = 10000
press 1 to see your interest :
press 0 to skip : 1
Enter today's date :(i,e day,month,year) 15 8 2010

```

```

Interest is : 135.61644
Enter your withdrawal amount :500
Now balance = 9635.616211

```

8.2: Modify the program of exercise 8.1 to include constructors for all three classes.

Solution:

```

1 #include<iostream.h>
2 #include<stdio.h>
3 #include<string.h>
4 #include<math.h>
5 #define minimum 500
6 #define service_charge 100
7 #define r 0.15
8

```

```

9  class account
10 {
11     protected:
12         char name[100];
13         int ac_number;
14         char ac_type[100];
15     public:
16         account( char *n,char *t,int no);
17 };
18 account::account(char *n,char *t,int no)
19 {
20     strcpy(name,n);
21     strcpy(ac_type,t);
22     ac_number=no;
23
24 }
25
26     class cur_acct: public account
27 {
28     private:
29         float balance,d,w;
30     public:
31         void withdraw(float ww);
32         void deposit(float d){balance=d;}
33     cur_acct(char *n,char *t,int number,float dp,float wd):
34     account(n,t,number)
35     {
36         d=dp;
37         w=wd;
38         deposit(d);
39         withdraw(w);
40
41     }
42     void display();
43 };
44
45 void cur_acct::withdraw(float ww)
46 {
47
48     if(balance<ww)
49         cout<<" sorry your balance is less than your withdrawal amount \n";
50     else
51     {
52         balance-=ww;
53         if(balance<minimum)
54         {
55             cout<<"\n your current balance is :"<<balance<<" which is less than"<<minimum<<"\n your
56             account is discharged by "<<service_charge<<"TK \n"<<" You must store
57             "<<minimum<<"TK to avoid discharge \n "<<" Do you want to withdraw ? press 1 for yes
58             press 0 for no \n"<<" what is your option ?";
59             int test;

```

```

60         cin>>test;
61         if(test==0)
62             balance+=w;
63     }
64     else
65     ;
66 }
67 }
68
69 void cur_acct::display()
70 {
71     cout<<"\n Now balance = "<<balance<<"\n";
72 }
73 class sav_acct:public account
74 {
75     float balance;
76     int d,m,y;
77     public:
78     void deposit(float d){ balance=d;set_date();}
79     void withdraw(float w);
80     void display();
81     void set_date(){d=12;m=1;y=2010;}
82     void interest();
83     sav_acct(char *n,char *t,int number,float dp,float wd):
84     account(n,t,number)
85     {
86         float d,w;
87         d=dp;
88         w=wd;
89         deposit(d);
90         interest();
91         withdraw(w);
92     }
93 };
94 };
95 void sav_acct::withdraw(float w)
96 {
97     if(balance<w)
98     cout<<" sorry your balance is less than your withdrawal amount \n";
99     else
100     {
101         balance-=w;
102         if(balance<minimum)
103         {
104 cout<<"\n your current balance is :"<<balance<<" which is less than"<<minimum<<"\n your
105account is discharged by "<<service_charge<<"TK \n"<<" You must store
106"<<minimum<<"TK to avoid discharge \n "<<" Do you want to withdraw ? press 1 for yes
107press 0 for no \n"<<" what is your option ?";
108             int test;
109             cin>>test;
110             if(test==0)

```

```

111             balance+=w;
112         }
113         else
114             ;
115     }
116
117}
118void sav_acct::display()
119{
120     cout<<"\n Now balance = "<<balance;
121}
122void sav_acct::interest()
123{
124     int D[12]={31,28,31,30,31,30,31,31,30,31,30,31};
125     int d1,y1,m1;
126     cout<<" Enter today's date :(i,e day,month,year) ";
127     cin>>d1>>m1>>y1;
128     int iday,fday;
129     iday=d;
130     fday=d1;
131     for(int i=0;i<m1;i++)
132     {
133         fday+=D[i];
134     }
135     for(i=0;i<m;i++)
136     {
137         iday+=D[i];
138     }
139     int tday;
140     tday=fday-iday;
141     float ty;
142     ty=float(tday)/365+y1-y;
143     balance=balance*pow((1+r),ty);
144}
145
146int main()
147{
148
149     float d;
150     cout<<" Enter customer name :";
151     char name[100];
152     gets(name);
153     cout<<" Enter account number :";
154     int number;
155     cin>>number;
156     cout<<" Enter your deposit amount : ";
157     cin>>d;
158
159     cout<<" Enter your withdrawal amount :";
160     float w;
161     cin>>w;

```

```
        //cur_acct s("current",name,number,d,w);
        //s.display();
        sav_acct c("savings",name,number,d,w);
        c.display();
    return 0;
}
```

output

Enter customer name :mehedi

Enter account number :1457

Enter your deposit amount : 5000

Enter your withdrawal amount :1200

Enter today's date :(i,e day,month,year) 13 7 2010

Now balance = 4160.875977

8.3: An educational institution wishes to maintain a database of its employees. The database is divided into a number of classes whose hierarchical relationships are shown in following figure. The figure also shows the minimum information required for each class. Specify all classes and define functions to create the database and retrieve individual information as and when required.

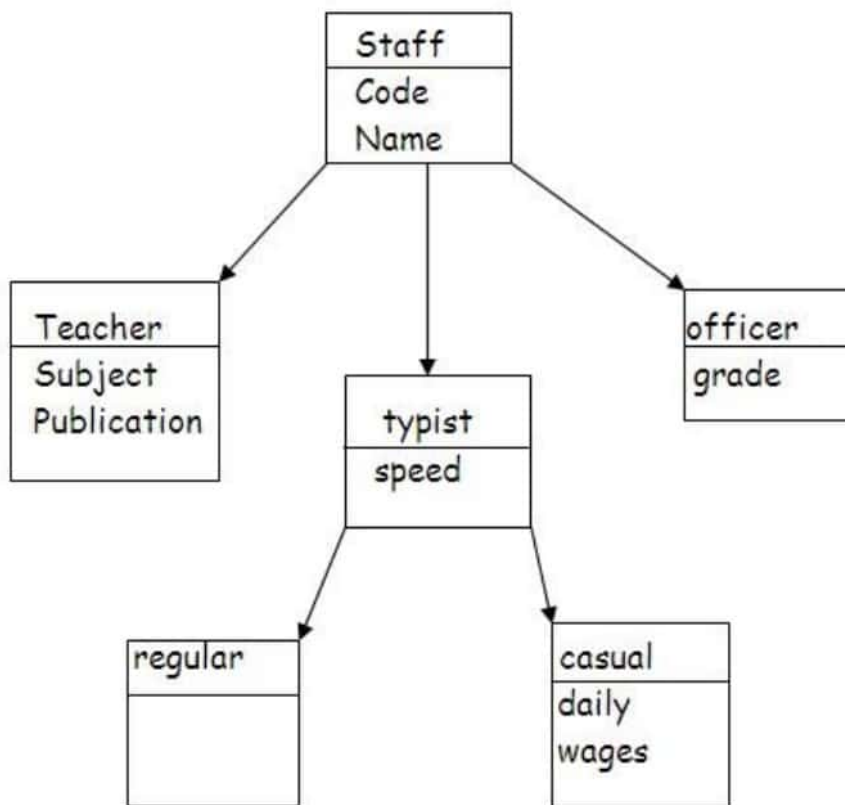


fig: class relationships (for exercise 8.3)

Solution:

```

1  #include<iostream.h>
2  #include<iomanip.h>
3  #include<string.h>
4
5  class staff
6  {
7      public:
8          int code;
9          char name[100];
10         public:
11         void set_info(char *n,int c)
12         {
13             strcpy(name,n);
14             code=c;
15         }
16 };
17
18 class teacher : public staff
  
```

```

19 {
20     protected:
21     char sub[100],publication[100];
22     public:
23     void set_details(char *s,char *p)
24     {
25         strcpy(sub,s);strcpy(publication,p);
26     }
27     void show()
28     {
29         cout<<"name"<<setw(8)<<"code"<<setw(15)<<"subject"<<setw(25)
30 <<"publication"<<endl<<name<<setw(8)<<code<<setw(25)<<sub<<setw(22)<<publication<<endl;
31     }
32 };
33
34 class officer:public staff
35 {
36     char grade[100];
37     public:
38     void set_details(char *g)
39     {
40         strcpy(grade,g);
41     }
42
43     void show()
44     {
45         cout<<" name "<<setw(15)<<"code"<<setw(15)<<"Category "<<endl
46 <<name<<setw(10)<<code<<setw(15)<<grade<<endl;
47     }
48 };
49 class typist: public staff
50 {
51     protected:
52     float speed;
53     public:
54     void set_speed(float s)
55     {
56         speed=s;
57     }
58 };
59 class regular:public typist
60 {
61     protected:
62     float wage;
63     public:
64     void set_wage(float w){wage=w;}
65     void show()
66     {
67         cout<<"name"<<setw(16)<<"code"<<setw(15)<<"speed"<<setw(15)
68 <<"wage"<<endl<<name<<setw(10)<<code<<setw(15)<<speed
69 <<setw(15)<<wage<<endl;

```

```

70     }
71 };
72 class causal:public typist
73 {
74     float wage;
75     public:
76     void set_wage(float w){wage=w;}
77     void show()
78     {
79         cout<<"name"<<setw(16)<<"code"<<setw(15)<<"speed"<<setw(15)
80 <<"wage"<<endl<<name<<setw(10)<<code<<setw(15)<<speed
81 <<setw(15)<<wage<<endl;
82     }
83
84 };
85
86 int main()
87 {
88
89     teacher t;
90     t.set_info("Ataur",420);
91     t.set_details("programming with c++"," Tata McGraw Hill");
92
93     officer o;
94     o.set_info("Md. Rashed",222);
95     o.set_details("First class");
96
97     regular rt;
98     rt.set_info("Robiul Awal",333);
99     rt.set_speed(85.5);
100    rt.set_wage(15000);
101
102    causal ct;
103ct.set_info("Kawser Ahmed",333);
104ct.set_speed(78.9);
105ct.set_wage(10000);
106    cout<<" About teacher: "<<endl;
107    t.show();
108    cout<<" About officer:"<<endl;
109    o.show();
110    cout<<" About regular typist :"<<endl;
111    rt.show();
112    cout<<" About causal typist :"<<endl;
113    ct.show();
114
115    return 0;
116}

```

output

About teacher:

name	code	subject	publication
Ataur	420	programming with c++	Tata McGraw Hill

About officer:

name	code	Category
Md. Rashed	222	First class

About regular typist :

name	code	speed	wage
Robiul Awal	333	85.5	15000

About causal typist :

name	code	speed	wage
Kawser Ahmed	333	78.900002	10000

8.4: The database created in exercise 8.3 does not include educational information of the staff. It has been decided to add this information to teachers and officers (and not for typists) which will help management in decision making with regard to training, promotions etc. Add another data class called education that holds two pieces of educational information namely highest qualification in general education and highest professional qualification. This class should be inherited by the classes teacher and officer. Modify the program of exercise 8.19 to incorporate these additions.

Solution:

```

1  #include<iostream.h>
2  #include<iomanip.h>
3  #include<string.h>
4
5  class staff
6  {
7      protected:
8          int code;
9          char name[100];
10     public:
11         void set_info(char *n,int c)
12         {
13             strcpy(name,n);
14             code=c;

```

```

15     }
16 };
17 class education:public staff
18 {
19     protected:
20         char quali[100];
21     public:
22         void set_qualification(char *q){strcpy(quali,q);}
23 };
24
25 class teacher : public education
26 {
27     protected:
28         char sub[100],publication[100];
29     public:
30         void set_details(char *s,char *p)
31         {
32             strcpy(sub,s);strcpy(publication,p);
33         }
34         void show()
35         {
36             cout<<" name "<<setw(8)<<"code"<<setw(15)
37                 <<"subject"<<setw(22)<<"publication"
38                 <<setw(25)<<"qualification"<<endl
39                 <<name<<setw(8)<<code<<setw(25)
40                 <<sub<<setw(18)<<publication<<setw(25)<<quali<<endl;
41         }
42 };
43
44 class officer:public education
45 {
46     char grade[100];
47     public:
48     void set_details(char *g)
49     {
50         strcpy(grade,g);
51     }
52
53     void show()
54     {
55         cout<<" name "<<setw(15)<<"code"<<setw(15)<<"Catagory "
56             <<setw(22)<<"Qualification"<<endl<<name<<setw(10)
57             <<code<<setw(15)<<grade<<setw(25)<<quali<<endl<<endl;
58     }
59 };
60 };
61
62 class typist: public staff
63 {
64     protected:
65         float speed;

```

```

66     public:
67     void set_speed(float s)
68     {
69         speed=s;
70     }
71 };
72 class regular:public typist
73 {
74     protected:
75     float wage;
76     public:
77     void set_wage(float w){wage=w;}
78     void show()
79     {
80         cout<<" name "<<setw(16)<<"code"<<setw(15)<<"speed"
81             <<setw(15)<<"wage"<<endl<<name<<setw(10)<<code
82             <<setw(15)<<speed<<setw(15)<<wage<<endl<<endl;
83     }
84 };
85 class causal:public typist
86 {
87     float wage;
88     public:
89     void set_wage(float w){wage=w;}
90     void show()
91     {
92         cout<<" name "<<setw(16)<<"code"<<setw(15)<<"speed"
93             <<setw(15)<<"wage"<<endl<<name<<setw(10)<<code
94             <<setw(15)<<speed<<setw(15)<<wage<<endl<<endl;
95     }
96
97 };
98
99 int main()
100{
101
102     teacher t;                t.set_info("Ataur",420);
103t.set_details("programming with c++"," Tata McGraw Hill");
104t.set_qualification("PHD from programming ");
105
106     officer o;
107
108     o.set_info("Md. Rashed",222);
109     o.set_details("First class");
110     o.set_qualification("2 years experienced");
111
112     regular rt;
113     rt.set_info("Robiul Awal",333);
114rt.set_speed(85.5);
115rt.set_wage(15000);
116

```

```

117     causal ct;
118         ct.set_info("Kawser Ahmed",333);
119         ct.set_speed(78.9);
120         ct.set_wage(10000);
121
122     cout<<" About teacher: "<<endl;
123     t.show();
124
125     cout<<" About officer:"<<endl;
126     o.show();
127
128     cout<<" About regular typist :"<<endl;
129     rt.show();
130     cout<<" About causal typist :"<<endl;
131     ct.show();
132
133     return 0;
134}

```

output

About teacher:

name	code	subject	publication	qualification
Ataur	420	programming with c++	Tata McGraw Hill	PHD from programming-

About officer:

name	code	Catagory	Qualification
Md. Rashed	222	First class	2 years experienced

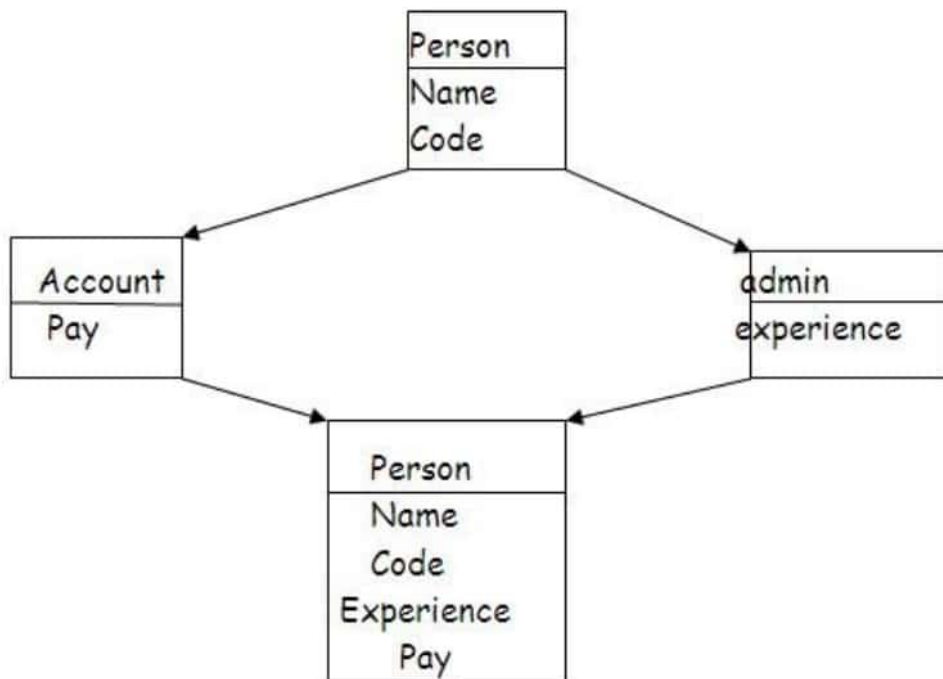
About regular typist :

name	code	speed	wage
Robiul Awal	333	85.5	15000

About causal typist :

name	code	speed	wage
Kawser	333	78.900002	10000

8.5: Consider a class network of the following figure. The class master derives information from both account and admin classes which in turn derives information from the class person. Define all the four classes and write a program to create, update and display the information contained in master objects.



Solution:

```

1 #include<iostream.h>
2 #include<iomanip.h>
3 #include<string.h>
4
5 class staff
6 {
7     protected:
8     int code;
9     char name[100];
10    public:
11    void set_info(char *n,int c)
12    {
13        strcpy(name,n);
14        code=c;
15    }
16 };
17
18 class education:public staff
19 {
20     protected:
21     char quali[100];
22     public:
23     void set_qualification(char *q){strcpy(quali,q);}
24 };
25
  
```



```

26 class teacher : public education
27 {
28     protected:
29     char sub[100],publication[100];
30     public:
31     void set_details(char *s,char *p)
32     {
33         strcpy(sub,s);strcpy(publication,p);
34     }
35 }
36
37     void show()
38     {
39         cout<<"name"<<setw(8)<<"code"<<setw(15)<<"subject"<<setw(22)
40 <<"publication"<<setw(25)<<"qualification"<<endl<<name<<setw(8)
41 <<code<<setw(25)<<sub<<setw(18)<<publication<<setw(25)<<quail
42 <<endl;
43     }
44 };
45
46 class officer:public education
47 {
48     char grade[100];
49     public:
50     void set_details(char *g)
51     {
52         strcpy(grade,g);
53     }
54
55     void show()
56     {
57         cout<<"name"<<setw(15)<<"code"<<setw(15)<<"Catagory"
58         <<setw(22)<<"Qualification"<<endl<<name<<setw(10)
59         <<code<<setw(15)<<grade<<setw(25)<<quali<<endl<<endl;
60     }
61 };
62
63 class typist: public staff
64 {
65     protected:
66     float speed;
67     public:
68     void set_speed(float s)
69     {
70         speed=s;
71     }
72 };
73
74 class regular:public typist
75 {
76     protected:
77     float wage;

```

```

77     public:
78     void set_wage(float w){ wage=w; }
79     void show()
80     {
81         cout<<"name"<<setw(16)<<"code"<<setw(15)<<"speed"<<setw(15)
82 <<"wage"<<endl<<name<<setw(10)<<code<<setw(15)<<speed
83 <<setw(15)<<wage<<endl<<endl;
84     }
85 };
86 class causal:public typist
87 {
88     float wage;
89     public:
90     void set_wage(float w){ wage=w; }
91
92     void show()
93     {
94         cout<<"name"<<setw(16)<<"code"<<setw(15)<<"speed"<<setw(15)
95 <<"wage"<<endl<<name<<setw(10)<<code<<setw(15)<<speed
96 <<setw(15)<<wage<<endl<<endl;
97     }
98
99 };
100
101 int main()
102 {
103
104     teacher t;
105     t.set_info("Ataur",420);
106     t.set_details("programming with c++"," Tata McGraw Hill");
107     t.set_qualification("PHD from programming ");
108     officer o;
109     o.set_info("Md. Rashed",222);
110     o.set_details("First class");
111     o.set_qualification("2 years experienced");
112     regular rt;
113     rt.set_info("Robiul Awal",333);
114     rt.set_speed(85.5);
115     rt.set_wage(15000);
116     causal ct;
117     ct.set_info("Kawser Ahmed",333);
118     ct.set_speed(78.9);
119     ct.set_wage(10000);
120     cout<<" About teacher: "<<endl;
121     t.show();
122     cout<<" About officer:"<<endl;
123     o.show();
124     cout<<" About regular typist :"<<endl;
125     rt.show();
126     cout<<" About causal typist :"<<endl;
127     ct.show();

```

```
128
129 return 0;
130}
```

output

name	code	Experience	payment
Hasibul	111	3 years	1500tk

8.6: In exercise 8.3 the classes teacher, officer, and typist are derived from the class staff. As we know we can use container classes in place of inheritance in some situations. Redesign the program of exercise 8.3 such that the classes teacher, officer and typist contain the objects of staff.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 #include<string.h>
4
5 class staff
6 {
7     public:
8         int code;
9         char name[100];
10        public:
11        void set_info(char *n,int c)
12        {
13            strcpy(name,n);
14            code=c;
15        }
16};
17
18class teacher : public staff
19{
20    protected:
21    char sub[100],publication[100];
22    public:
23    void set_details(char *s,char *p)
24    {
25        strcpy(sub,s);strcpy(publication,p);
26    }
27    void show()
28    {
29        cout<<"name"<<setw(8)<<"code"<<setw(15)<<"subject"<<setw(25)
30<<"publication"<<endl<<name<<setw(8)<<code<<setw(25)<<sub
```

```

31<<setw(22)<<publication<<endl;
32    }
33);
34
35    class officer:public staff
36{
37    char grade[100];
38    public:
39    void set_details(char *g)
40    {
41        strcpy(grade,g);
42    }
43    void show()
44    {
45        cout<<" name "<<setw(15)<<"code"<<setw(15)<<"Catagory "<<endl
46        <<name<<setw(10)<<code<<setw(15)<<grade<<endl;
47    }
48};
49
50class typist: public staff
51{
52    protected:
53    float speed;
54    public:
55    void set_speed(float s)
56    {
57        speed=s;
58    }
59    void show()
60    {
61        cout<<" name "<<setw(15)<<"code"<<setw(15)<<"speed"<<endl
62        <<name<<setw(10)<<code<<setw(15)<<speed<<endl<<endl;
63    }
64};
65
66    int main()
67{
68
69    teacher t;
70    t.set_info("Ataur",420);
71    t.set_details("programming with c++"," Tata McGraw Hill");
72
73    officer o;
74    o.set_info("Md. Rashed",222);
75    o.set_details("First class");
76
77    typist tp;
78    tp.set_info("Robiul Awal",333);
79    tp.set_speed(85.5);
80
81    cout<<" About teacher: "<<endl;

```

```

82  t.show();
83  cout<<" About officer:"<<endl;
84  o.show();
85  cout<<" About typist :"<<endl;
86  tp.show();
87  return 0;
88}

```

output

About teacher:

name	code	subject	publication
Ataur	420	programming with c++	Tata McGraw Hill

About officer:

name	code	Catagory
Md. Rashed	222	First class

About typist :

name	code	speed
Robiul Awal	333	85.5

8.7: We have learned that OOP is well suited for designing simulation programs. Using the techniques and tricks learned so far, design a program that would simulate a simple real-world system familiar to you

Solution:

```

1  #include<iostream.h>
2  #include<stdio.h>
3  #include<string.h>
4  #include<iomanip.h>
5  #include<conio.h>
6
7  char *sub[10]={"Bangla 1st paper","Bangla 2nd paper","English 1st paper",
8              "English 2nd paper","Mathematics","Religion",
9              "Physics","Chemistry","Sociology","Higher Mathematics"};
10
11 class student_info
12 {

```

```

13
14     public:
15         char name[40];
16         char roll[20];
17     public:
18         void set_info();
19 };
20
21 void student_info::set_info()
22 {
23     cout<<"Enter student name : ";
24     gets(name);
25     cout<<"Enter roll: ";
26     gets(roll);
27 }
28
29 class subject :public student_info
30 {
31
32     public:
33         float mark[10];
34
35     public:
36         void set_mark();
37 };
38
39 void subject::set_mark()
40 {
41     cout<<" marks of : \n";
42     for(int i=0;i<10;i++)
43     {
44         cout<<sub[i]<<" = ? ";
45         cin>>mark[i];
46     }
47
48 }
49 class conversion :public subject
50 {
51     float gpa[10];
52     char grade[20][20];
53     public:
54     void convert_to_gpa();
55     void show();
56 };
57 void conversion::convert_to_gpa()
58 {
59     for(int i=0;i<10;i++)
60     {
61         if(mark[i]>=80)
62         {
63             gpa[i]=5.00;

```

```

64         strcpy(grade[i],"A+");
65     }
66     else if(mark[i]>=70 && mark[i]<80)
67     {
68         gpa[i]=4.00;
69         strcpy(grade[i],"A");
70     }
71     else if(mark[i]>=60 && mark[i]<70)
72     {
73         gpa[i]=3.50;
74         strcpy(grade[i],"A-");
75     }
76     else if(mark[i]>=50 && mark[i]<60)
77     {
78         gpa[i]=3.00;
79         strcpy(grade[i],"B");
80     }
81     else if(mark[i]>=40 && mark[i]<50)
82     {
83         gpa[i]=2.00;
84     strcpy(grade[i],"C");
85     }
86     else if(mark[i]>=33 && mark[i]<40)
87     {
88         gpa[i]=1.00;
89         strcpy(grade[i],"D");
90     }
91     else
92     {
93         gpa[i]=0.00;
94         strcpy(grade[i],"Fail");
95     }
96 }
97 }
98
99 void conversion::show()
100{
101     cout<<" result of \n";
102     cout<<"name : "<<name<<"\n";
103     cout<<"Roll : " <<roll<<"\n";
104     cout<<setw(25)<<"Subject" <<setw(17)<<"Marks"
105         <<setw(14)<<"GPA" <<setw(12)<<"Grade \n";
106     for(int i=0;i<10;i++)
107     {
108         cout<<setw(25)<<sub[i]<<setw(15)<<mark[i]
109         <<setw(15)<<gpa[i]<<setw(10)<<grade[i]<<"\n";
110     }
111}
112int main()
113{
114     clrscr();

```

```

115     conversion A;
116     A.set_info();
117     A.set_mark();
118     A.convert_to_gpa();
119     A.show();
120     getch();
121     return 0;
122}

```

output

Enter student name : santo

Enter roll: 156271

marks of :

Bangla 1st paper = ? 74

Bangla 2nd paper = ? 87

English 1st paper = ? 45

English 2nd paper = ? 56

Mathematics = ? 87

Religion = ? 59

Physics = ? 75

Chemistry = ? 65

Sociology = ? 39

Higher Mathematics = ? 86

result of

name :santo

Roll : 156271

Subject	Marks	GPA	Grade
Bangla 1st paper	74	4	A
Bangla 2nd paper	87	5	A+

English 1st paper	45	2	C
English 2nd paper	56	3	B
Mathematics	87	5	A+
Religion	59	3	B
Physics	75	4	A
Chemistry	65	3.5	A-
Sociology	39	1	D
Higher Mathematics	86	5	A+

Chapter 9

Review Questions

9.1: What does polymorphism mean in C++ language?

Ans: In short, polymorphism means one thing with several distinct forms.

– In details, using operators or functions in different ways, depending on what they are operating on, is called polymorphism.

9.2: How is polymorphism achieved at (a) compile time, and (b) run time?

Ans: Polymorphism can be achieved at compile time by early binding. Early binding means an object is bound to its function call at compile time.

And we can achieve run time polymorphism by a mechanism known as virtual function.

9.3: Discuss the different ways by which we can access public member functions of an object.

Ans: We can access public member functions of an object by

- (i) Object name and dot membership operator.
- (ii) Pointer to object and function name.

9.4: Explain, with an example, how you would create space for an array of objects using pointers.

Ans: We can also create an array of objects using pointers. For example, the statement `item *ptr = new item [10]; // array of 10 objects.` creates memory space for an array of 10 objects of item.

9.5: What does this pointer point to?

Ans: 'this' pointer points to invoking object.

9.6: What are the applications of this pointer?

Ans: One important application of the pointer 'this' is to return the object it points to. For example, the statement `return * this;` inside a member function will return the object that invoked the function.

9.7: What is a virtual function?

Ans: When we use the same function name in both the base and derived classes the function in the base class is declared as virtual using the keyword `virtual` preceding its normal declaration.

9.8: Why do we need virtual functions?

Ans: If we need the same function name at base class and derived class then, we need virtual function.

9.9: When do we make a virtual function "pure"? What are the implications of making a function a pure virtual function?

Ans: When a function is defined as empty, then this function is called a do nothing function. The implications of making a function a pure virtual function is to achieve run time polymorphism.

9.10: State which of the following statements are TRUE or FALSE.

- (a) Virtual functions are used to create pointers to base classes.
- (b) Virtual functions allow us to use the same function call to invoke member functions of objects of different classes.

- (c) A pointer to a base class cannot be made to point to objects of derived class.
- (d) this pointer points to the object that is currently used to invoke a function.
- (e) this pointer can be used like any other pointer to access the members of the object it points to.
- (f) this pointer can be made to point to any object by assigning the address of the object.
- (g) Pure virtual functions force the programmer to redefine the virtual function inside the derived classes.

Ans:

- (a) TRUE
- (b) TRUE
- (c) FALSE
- (d) TRUE
- (e) TRUE
- (f) TRUE
- (g) TRUE

Debugging Exercises

9.1: Identify the error in the following program.

```
#include <iostream.h>;
class Info
{
    char* name;
    int Number;
public:
    void getInfo()
    {
        cout << "Info::getInfo";
        getName();
    }

    void getName()
    {
        cout << "Info::getName";
    }
};

class Name: public Info
{
    char *name;
public:
    void getName()
    {
        cout << "Name::getName";
    }
};
```

```

void main()
{
    Info *P;
    Name n;
    P = n;
    p->getInfo();
}
/*

```

Solution: Here P=n will replace with P=&n in the main() function. Because P is a pointer.

9.2: Identify the error in the following program.

```

#include <iostream.h>;
class Person
{
    int age;
public:
    Person()
    {
    }

    Person(int age)
    {
        this.age = age;
    }
    Person& operator < (Person &p)
    {
        return age < p.age? p: *this;
    }
    int getAge()
    {
        return age;
    }
};

void main()
{
    Person P1 (15);
    Person P2 (11);
    Person P3;
    //if P1 is less than P2
    P3 = P1 < P2; P1.lessthan(P2)
    cout << P3.getAge();
}
/*

```

Solution: The function

```
person (int age)
{
    this.age = age;
}
```

should write like as...

```
person (int age)
{
    -this > age = age;
}
```

9.3: Identify the error in the following program.

```
#include <iostream.h>;
class Human
{
public:
    Human()
    {
    }

    virtual ~Human()
    {
        cout << "Human::~Human";
    }
};
class Student: public Human
{
public:
    Student()
    {
    }
    ~Student()
    {
        cout << "Student::~Student()";
    }
};

void main()
{
    Human *H = new Student();
    delete H;
}
```

```
}
```

Solution: Here we cannot write `Human *H = new student();` in the `main()` function because base class's member includes in derived class's object so we should write this as follow
`student *H = new Student();`

9.4: Correct the errors in the following program.

```
class Human
{
private:
    int m;
public:
    void getdata()
    {
        cout << " Enter number:";
        cin >> m;
    }
};
main()
{
    test T;
    T->getdata();
    T->display();

    test *p;
    p = new test;
    p.getdata();
    (*p).display();
}
```

Solution: Here `T->getdata` replace with `T.getdata` and `T->display` replace with `T.display` in the `main()` function. Because in this program T is object to pointer.

9.5: Debug and run the following program. What will be the output?

```
#include<iostream.h>
class A
{
protected:
    int a,b;
public:
    A(int x = 0, int y)
    {
```

```

        a = x;
        b = y;
    }
    virtual void print ();
};
class B: public A
{
    private:
        float p,q;
    public:
        B(intm, int n, float u, float v)
        {
            p = u;
            q = v;
        }
        B() {p = p = 0;}
        void input(float u, float v);
        virtual void print(float);
};
void A::print(void)
{
    cout << A values: << a <<"<< b << "\n";
}
void B::print(float)
{
    cout << B values: << u <<"<< v << "\n";
}
void B::input(float x, float y)
{
    p = x;
    q = y;
}
main()
{
    A a1(10,20), *ptr;
    B b1;
    b1.input(7.5,3.142);

    ptr = &a1;
    ptr->print();

    ptr = &b1;
    ptr->print();
}

```

Programming Exercises

9.1: Create a base class called shape. Use this class to store two double type values that could be used to compute the area of figures. Derive two specific classes called triangle and rectangle from the base shape. Add to the base class, a member function get_data() to initialize base class data members and another member function display_area() to compute and display the area of figures. Make display_area() as a virtual function and redefine this function in the derived classes to suit their requirements.

Using these three classes, design a program that will accept dimensions of a triangle or a rectangle interactively, and display the area.

Remember the two values given as input will be treated as lengths of two sides in the case of rectangles and as base and height in the case of triangles, and used as follows:

Area of rectangle = $x * y$

Area of triangle = $\frac{1}{2} * x * y$

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 class shape
4 {
5     public:
6     double x,y;
7     public:
8     void get_data()
9     {
10         cin>>x>>y;
11     }
12     }
13     double get_x(){return x;}
14         double get_y(){return y;}
15     virtual void display_area(){ }
16};
17
18class triangle:public shape
19{
20     public:
21     void display_area()
22     {
23         double a;
24         a=(x*y)/2;
25         cout<<" Area of triangle = "<<a<<endl;
26     }
27 }
28};
29class rectangle:public shape
30{
```



```

31     public:
32     void display_area()
33     {
34         double a;
35         a=x*y;
36         cout<<" Area of rectangle = "<<a<<endl;
37     }
38};
39     int main()
40{
41
42     shape *s[2];
43     triangle t;
44     s[0]=&t;
45     rectangle r;
46     s[1]=&r;
47     cout<<" Enter the value of x & y for triangle: ";
48     s[0]->get_data();
49     cout<<" Enter the value of x & y for rectangle: ";
50     s[1]->get_data();
51     s[0]->display_area();
52     s[1]->display_area();
53     return 0;
54 }

```

output

Enter the value of x & y for triangle: 12 26

Enter the value of x & y for rectangle: 24 14

Area of triangle = 156

Area of rectangle = 336

9.2: Extend the above program to display the area of circles. This requires addition of a new derived class 'circle' that computes the area of a circle. Remember, for a circle we need only one value, its radius, but the get_data() function in base class requires two values to be passed.(Hint: Make the second argument of get_data() function as a default one with zero value.)

Solution:

```

1 #include<iostream.h>
2 #include<iomanip.h>
3 #define pi 3.1416
4 class shape

```

```

5 {
6     public:
7     double x,y;
8     public:
9     void get_data(double a,double b)
10    {
11        x=a;
12        y=b;
13
14    }
15    double get_x(){return x;}
16    double get_y(){return y;}
17    virtual void display_area(){}
18};
19
20class triangle:public shape
21{
22    public:
23    void display_area()
24    {
25        double a;
26        a=(x*y)/2;
27        cout<<" Area of triangle = "<<a<<endl;
28
29    }
30};
31
32    class rectangle:public shape
33{
34    public:
35    void display_area()
36    {
37        double a;
38        a=x*y;
39        cout<<" Area of rectangle = "<<a<<endl;
40    }
41};
42class circle:public shape
43{
44    public:
45    void display_area()
46    {
47        double a;
48        a=pi*x*x;
49        cout<<" Area of circle = "<<a<<endl;
50    }
51};
52
53int main()
54{
55

```

```

56         shape *s[3];
57
58     triangle t;
59     s[0]=&t;
60
61     rectangle r;
62     s[1]=&r;
63
64     circle c;
65     s[2]=&c;
66     double x,y;
67     cout<<" Enter the value of x & y for triangle: ";
68     cin>>x>>y;
69     s[0]->get_data(x,y);
70     cout<<" Enter the value of x & y for rectangle: ";
71     cin>>x>>y;
72     s[1]->get_data(x,y);
73     cout<<" Enter the radius of circle : ";
74     double rd;
75     cin>>rd;
76     s[2]->get_data(rd,0);
77     cout<<endl<<endl;
78     s[0]->display_area();
79     s[1]->display_area();
80     s[2]->display_area();
81
82     return 0;
83}

```

output

Enter the value of x & y for triangle: 10 24

Enter the value of x & y for rectangle: 14 23

Enter the radius of circle : 12

Area of triangle = 120

Area of rectangle = 322

Area of circle = 452.3904

9.3: Run the program above with the following modifications:

(a) Remove the definition of display_area() from one of the derived classes.

(b) In addition to the above change, declare the `display_area()` as virtual in the base class `shape`.
Comment on the output in each case.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 #define pi 3.1416
4 class shape
5 {
6     public:
7     double x,y;
8     public:
9     void get_data(double a,double b)
10    {
11        x=a;
12        y=b;
13    }
14 }
15 double get_x(){return x;}
16 double get_y(){return y;}
17 virtual void display_area(){}
18 };
19 class triangle:public shape
20 {
21     public:
22     void display_area()
23     {
24         double a;
25         a=(x*y)/2;
26         cout<<" Area of triangle = "<<a<<endl;
27     }
28 };
29 class rectangle:public shape
30 {
31     public:
32     void display_area()
33     {
34         double a;
35         a=x*y;
36         cout<<" Area of rectangle = "<<a<<endl;
37     }
38 };
39 class circle:public shape
40 {
41     public:
42     void display_area()
43     {
44         double a;
```

```

45     a=pi*x*x;
46     cout<<" Area of circle = "<<a<<endl;
47 }
48);
49
50int main()
51{
52
53     shape *s[3];
54     triangle t;
55     s[0]=&t;
56
57     rectangle r;
58     s[1]=&r;
59     circle c;
60     s[2]=&c;
61     double x,y;
62     cout<<" Enter the value of x & y for triangle: ";
63     cin>>x>>y;
64     s[0]->get_data(x,y);
65     cout<<" Enter the value of x & y for rectangle: ";
66     cin>>x>>y;
67     s[1]->get_data(x,y);
68     cout<<" Enter the radius of circle : ";
69     double rd;
70     cin>>rd;
71     s[2]->get_data(rd,0);
72     cout<<endl<<endl;
73     s[0]->display_area();
74     s[1]->display_area();
75     s[2]->display_area();
76
77     return 0;
78}

```

output

Enter the value of x & y for triangle: 28 32

Enter the value of x & y for rectangle: 25 36

Enter the radius of circle : 20

Area of triangle = 448

Area of rectangle = 900

Area of circle = 1256.64

Chapter 10

Review Questions

10.1: **What is a stream?**

Ans: The I/O system in C++ is designed to work with a wide variety of devices including terminals, disks and tape drives. Although each device is very different, the I/O system supplies an interface to the programmer that is independent of the actual device being accessed. This interface is known as stream.

10.2: **Describe briefly the features of I/O system supported by C++.**

Ans: See table 10.1, page 293 of Balagurusamy.

10.3: **How do the I/O facilities in C++ differ from that in C?**

Ans: We can design our own manipulators for certain special purposes in C++ but not in C.

Example :

```
1 ostream & unit (ostream & output)
2 {
3     output << "TK";
4     return output;
5 }
6 The statement
7     cout << 100 << unit;
8 will print
9     100 Tk.
```

10.4: **Why are the words such as cin and cout not considered as keywords?**

Ans: cin and cout are related to iostream object, that's why we must include iostream header file in our program to use cin and cout.

Hence cin and cout are considered as iostream objects, not keywords.

10.5: **How is cout able to display various types of data without any special instructions?**

Ans: The biggest single advantage to the stream methods is: they are type-safe. If you change a variable's type, the subsequent stream operations using that variable will either automatically accommodate the change, or will indicate an incompatibility at compile time. Thus cout is able to display various types of data without any special instructions.

10.6: Why is it necessary to include the file iostream in all our programs?

Ans: Because any program has input or output statements and without iostream header file we can not use input or output statements, that's why it is necessary to include the file iostream in all our programs.

10.7: Discuss the various forms of get() function supported by the input stream. How are they used?

Ans: There are two types of get () functions. We can use both get(char*) and get(void) prototype to fetch a character including the blank space.

10.8: How do the following two statements differ in operation?

```
cin >> c;  
cin.get(c);
```

Ans: cin>>c will read a character but it will skip the white space and new line character. cin.get (c); will read a character including white space and newline character.

10.9: Both cin and getline() function can be used for reading a string. Comment.

Ans: cin reads a string and stops when a space or newline is encountered. getline () function reads a whole line of text that ends with a newline character.

**10.10: Discuss the implications of size parameter in the following statement:
cout.write(line, size);**

Ans: cout.write (line, size)

The first argument line represents the name of the string to be displayed and the second argument size indicates the number of characters to display.

10.11: What does the following statement do?

```
cout.write(s1,m).write(s2,n);
```

Ans: The statement
`cout.write(s1,m).write (s2,n);`
is equivalent to the following two statements;
`cout.write(s1,m);`
`cout.write(s2,n);`
So, `cout.write (s1,m).write (s2,n);` will print two string s1 and s2

10.12: What role does the iomanip file play?

Ans: The header file `iomanip` provides a set of functions called manipulators which can be used to manipulate the output formats.

10.13: What is the role of `fill()` function? When do we use this function?

Ans: `fill ()` is known as filling and padding function. The unused positions of the field are filled with white spaces, by default. We can use the `fill ()` function to fill the unused positions by any desired character.

General form :

`cout.fill (ch);`

Example :

`cout.fill ('*');`

10.14: Discuss the syntax of `set()` function.

Ans: syntax of set function:

`cout.setf (arg1, arg2);`

here arg1 is one of the formatting flags
and arg2 is bit field.

Example:

`cout.setf (ios::left, ios :: adjust field);`

10.15: What is the basic difference between manipulators and ioa member functions in implementation? Give examples.

Ans: We can design our manipulator using manipulator function, which can not be done by ios member functions

Example :

```
1 ostream & show (ostream & output)
2 {
3     output.setf (ios :: showpoint);
4     output.setf(ios :: showpos);
5     output << setw(10);
6     return output;
7 }
```

This function defines a manipulator called show that turns on the flags showpoint and showpos declared in the class ios and sets the field width to 10.

10.16: State whether the following statements are TRUE or FALSE.

- (a) A C++ stream is a file.
- (b) C++ never truncates data.
- (c) The main advantage of width() junction is that we can use one width specification for more than one items.
- (d) The get(void) function provides a single-character input that does not skip over the white spaces.
- (e) The header file iomanip can be used in place of iostream.
- (f) We cannot use both the C 110 functions and C++ 110 functions in the same program.
- (g) A programmer can define a manipulator that could represent a set of format functions.

Ans:

- (a) TRUE
- (b) FALSE
- (c) FALSE
- (d) TRUE
- (e) TRUE
- (f) FALSE
- (g) TRUE

Debugging Exercises

10.1: Sorry this question is missed. We will try to give this question with solution very soon. Stay with us

10.2: Will the statement cout.setf(ios::right) work or not?

```
1 #include <iostream.h>
2 void main()
3 {
4     cout.width(5);
5     cout << "99" << endl;
```

```

6
7 cout.setf(ios::left);
8 cout.width(5);
9 cout << "99" << endl;
10
11 cout.setf(ios::right);
12 cout << "99" << endl;
13}

```

Solution: cout.setf(ios :: right) will not work because width() function was not invoked before cout.setf(ios :: right)

10.3: State errors, if any, in the following statements.

(a) cout << (void*) amount; (b) cout << put ("John") ; (c) cout << width(); (d) int p = cout.width(10); (e) cout.width(10).precision(3); (f) cout.setf(ios::scientific,ios::left); (g) ch = cin.get(); (h) cin.get().get(); (i) cin.get(c).get(); (j) cout << setw(5) << setprecision(2); (k) cout << resetiosflags(ios::left ios::showpos);

Solution:

(a) missing

(b) put () a member function of ostream class. It is used to output a character. Hence it does not return any value; so cout<<("John"); is not valid. **Correction :**
cout.put ('John');

(c) width () function does not return any value. So cout <**Correction : cout.width ();**

**(e) cout.width (10) precision (3); must be written separately
cout.width (10);
cout.precision (3);**

**(f) If you want to see output as scientific format you can achieve this as follow:
cout.setf (ios :: scientific, ios :: floatfield);**

**If you want to see output at left field you can achieve this as follows:
cout.setf (ios :: left, ios :: adjustfield);**

(g) No error

(h) cannot be concatenated;

Correction :

cin.get ()

cin.get ()

(i) No error

(j) No error

(k) No error

Programming Exercises

10.1: Write a program to read a list containing item name, item code, and cost interactively and produce a three column output as shown below.

Name	Code	Cost
Turbo C++	1001	250.95
C primer	905	95.70

.....
.....
.....
Note that the name and code are left-justified and the cost is right justified with a precision of two digits. Trailing zeros are shown.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 #include<string.h>
4 class item
5 {
6     char name[40];
7     int code;
8     float cost;
9     public:
10    void get_data(char *n,int c,float co)
11    {
12        strcpy(name,n);
13        code=c;
14        cost=co;
15    }
16    void display();
17
18 };
19
20 void item:: display()
21 {
22     cout.precision(2);
23     cout.setf(ios::fixed,ios::floatfield);
24     cout.setf(ios::showpoint);
25     cout.setf(ios::left,ios::adjustfield);
26     cout<<setw(40)<<name<<code;
27     cout.setf(ios::right,ios::adjustfield);
28     cout<<setw(15)<<cost<<endl;
29 }
30
```

```

31 int main()
32 {
33     item a[5];
34     a[0].get_data("Tarbo C++",1001,250.95);
35     a[1].get_data("C primer",905,95.7);
36     a[2].get_data("algorithm",1111,120.5);
37     a[3].get_data("principle of electronics",2220,150.85);
38     a[4].get_data("solution of balagurusamy",6666,145.00);
39     cout<<setw(10)<<"name"<<setw(34)<<"code"<<setw(15)<<"cost"<<endl;
40     for(int i=0;i<60;i++)
41         cout<<"-";
42     cout<<endl;
43     for(i=0;i<5;i++)
44         a[i].display();
45     return 0;
46}

```

output

name	code	cost
Tarbo C++	1001	250.95
C Primer	905	95.70
algorithm	1111	120.50
Principle of electronics	2220	150.85
Solution of balaguruswamy	6666	145.00

10.2: Modify the above program to fill the unused spaces with hyphens.

Solution:

```

1 #include<iostream.h>
2 #include<iomanip.h>
3 #include<string.h>
4 class item
5 {
6     char name[40];
7     int code;
8     float cost;
9     public:
10    void get_data(char *n,int c,float co)

```

```

11     {
12         strcpy(name,n);
13         code=c;
14         cost=co;
15     }
16     void display();
17
18 };
19 void item:: display()
20 {
21
22     cout.precision(2);
23     cout.fill('-');
24     cout.setf(ios::fixed,ios::floatfield);
25     cout.setf(ios::showpoint);
26     cout.setf(ios::left,ios::adjustfield);
27     cout<<setw(40)<<name<<code;
28     cout.setf(ios::right,ios::adjustfield);
29     cout<<setw(15)<<cost<<endl;
30 }
31
32 int main()
33 {
34     item a[5];
35     a[0].get_data("Tarbo C++",1001,250.95);
36     a[1].get_data("C primer",905,95.7);
37     a[2].get_data("algorithm",1111,120.5);
38     a[3].get_data("principle of electronics",2220,150.85);
39     a[4].get_data("solution of balagurusamy",6666,145.00);
40     cout<<setw(10)<<"name"<<setw(34)<<"code"<<setw(15)<<"cost"<<endl;
41     for(int i=0;i<60;i++)
42         cout<<"-";
43     cout<<endl;
44     for(i=0;i<5;i++)
45         a[i].display();
46     return 0;
47 }

```

output

name	code	cost
Tarbo C++	1001	250.95
C Primer	905	95.70
algorithm	1111	120.50

10.3: Write a program which reads a text from the keyboard and displays the following information on the screen in two columns:

- (a) Number of lines
- (b) Number of words
- (c) Number of characters

Strings should be left-justified and numbers should be right-justified in a suitable field width.

Solution:

```
1 #include<iostream.h>
2 #include<iomanip.h>
3 #include<string.h>
4 #include<stdio.h>
5
6 int main()
7 {
8     char line[1000];
9     char ch;
10    int c;
11    int word,lines,chr;
12    word=0;
13    lines=0;
14    chr=0;
15    int end=0;
16    cout<<" Enter text : \n";
17    while(end==0)
18    {
19        c=0;
20        while((ch=getchar())!='\n')
21            line l=ch;
22        line l='\0';
23        if(line[0]=='\0')
24            break;
25        else
26        {
27            word++;
28            for(int i=0;line[i]!='\0';i++)
29                if(line[i]==' ' || line[i]=='\t' || line[i]=='\n')
30                    word++;
31        }
32        lines++;
33        chr+=strlen(line);
34    }
```

```

35
36     cout.setf(ios::left,ios::adjustfield);
37     cout<<setw(25)<<"Number of lines"<<setw(25)
38     <<"Number of words " <<"Number of characters " <<endl;
39     cout.setf(ios::right,ios::adjustfield);
40     cout<<setw(10)<<lines<<setw(24)<<word<<setw(25)<<chr<<endl<<endl;
41     return 0;
42}

```

output

Enter text :

santo reads in class five.

He always speak the truth.

He respects his teachers.

He feels shy when I admire him.

I like his morality.

Number of lines	Number of words	Number of characters
-----------------	-----------------	----------------------

5	25	128
---	----	-----

Note: If you press the Enter button two times, the program will terminate.

Chapter 11

Review Questions

11.1: What are input and output streams?

Ans:The stream that supplies data to the program is known as input stream and one that receives data from the program is known as output stream.

11.2: What are the steps involved in using a file in a C++ program?

Ans:The I/O system of C++ handles file operations which are very much similar to the console

input and output operations. The input stream extracts data from the file and output stream inserts data to the file.

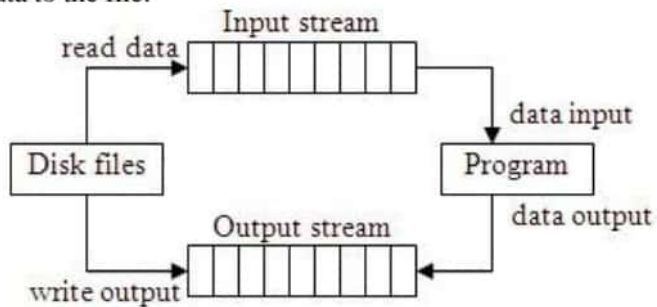


Fig : File input and output streams

11.3: Describe the various classes available for file operations.

Ans:See fig 11.3 of Balagurusamy, Page-325

11.4: What is the difference between opening a file with a constructor function and opening a file with open() function? When is one method preferred over the other?

Ans:When we open a file with a constructor a file name must be supplied during creation of object.

Example : ofstream outfile ("result"); //output only.

But when we use open () function we can provide file name later.

Example :

```
ofstream outfile;
outfile.open ("Result");
```

Another difference is we can provide file modes using open () function.

```
stream-object open ("file name", mode);
```

constructor method is proffered for only one file in the stream.

member function open () of the class is preferred when we want to manage multiple files using one stream.

11.5: Explain how while(fin) statement detects the end of a file that is connected to fin stream

Ans:The given statement is
while (fin)

An ifstream object, such as fin, returns a value of 0 if any error occurs in the file operation including the end-of-file condition. We know while loop will terminate when fin returns zero. Thus end of file is detected.

11.6: What is a file mode? Describe the various file mode options available.

Ans: File-mode specifies the purpose for which the file is opened.

Various file mode options :

Parameter	Meaning
ios::app	Append to end of file
ios::ate	Go to end of file on opening
ios::binary	Binary file
ios::in	open file for reading only
ios::nocreate	open fails if the file does not exist
ios::noreplace	open fails if file already exist
ios::out	open file for writing only.
ios::trunc	Delete the contents of the file if it exist.

11.7: Write a statement that will create an object called fob for writing, and associate it with a file name DATA

Ans: ofstream fob;
fob.open ("DATA");

11.8: How many file objects would you need to create to manage the following situations?

- (a) To process four files sequentially.
- (b) To merge two sorted files into a third file. Explain.

Ans: (a) only a single stream object.

(b) In such case we need to create two separate input stream for handling the two input files and one output stream for handling the output file. That means require three stream object.

11.9: Both ios::ate and ios::app place the file pointer at the end of the file (when it is opened). What then, is the difference between them?

Ans: Both ios::ate and ios::app take us to the end of the file when it is opened. The difference between the two parameters is that the ios::app allows us to add data to the end of the file only. While ios::ate mode permits us to add data or to modify the existing data anywhere in the file.

11.10: What does the “current position” mean when applied to files?

Ans: The ‘current position’ applied to file means the byte location at which the next read or write operation will take place.

11.11: Write statements using seekg() to achieve the following:

- (a) To move the pointer by 15 positions backward from current position.
- (b) To go to the beginning after an operation is over.
- (c) To go backward by 20 bytes from the end.
- (d) To go to byte number 50 in the file.

Ans: Let, fout be an ofstream object.

- (a) fout.seekg (-15,ios::cur);
- (b) fout.seekg(0,ios::beg);
- (c) fout.seekg (-20,ios::end);
- (d) fout.seekg (49,ios::beg);

11.12: What are the advantages of saving data in binary form?

Ans: The binary format is more accurate for storing the numbers as they are stored in the exact internal representation. There are no conversions while saving the data and therefore saving is much faster read () and write () function do this job.

11.13: Describe how would you determine number of objects in a file. When do you need such information?

Ans: We can find out the total number of objects in a file using object_length as follows:

$n = \text{file_size} / \text{object_length}$;

This information is required to

1. Display the contents of a file.
2. Modify an existing item
3. Adding new item
4. Deleting an existing item

11.14: Describe the various approaches by which we can detect the end-of-file condition successfully.

Ans: Let, `fin` be an object of `ifstream`. If end of file occurs `fin` returns zero. So we can use `while (fin)` to detect end of file.

11.15: State whether the following statements are TRUE or FALSE.

- (a) A stream may be connected to more than one file at a time.
- (b) A file pointer always contains the address of the file.
- (c) The statement `outfile.write(char * & obj,sizeof(obj));` writes only data in `obj` to `outfile`.
- (d) The `ios::ate` mode allows us to write data anywhere in the file.
- (e) We can add data to an existing file by opening in write mode.
- (f) The parameter `ios::app` can be used only with the files capable of output.
- (g) The data written to a file with `write()` function can be read with the `get()` function.
- (h) We can use the functions `tellp()` and `tellg()` interchangeably for any file.
- (i) Binary files store floating point values more accurately and compactly than the text files.
- (j) The `fin.fail()` call returns non-zero when an operation on the file has failed.

Ans:

- (a) FALSE
- (b) FALSE
- * file pointer can be a synonym for current position.
- (c) TRUE
- (d) TRUE
- (e) TRUE
- (f) TRUE
- (g) FALSE
- (h) FALSE
- * `tellp()` is for output file and `tellg()` for input file
- (i) TRUE
- (j) FALSE
- * `fin.fail()` returns non-zero when an input or output operation on the file has failed.

Debugging Exercises

NOTE: We are sorry to say that Debugging Exercise question no 11.1 , 11.2 and 11.3 is unfortunately missed. We will come back with this question and answers very soon. Stay tune with us. Happy coding

11.4: Find errors in the following statements.

- (a) ifstream infile("DATA");
- (b) finl.getline(); //finl is input stream
- (c) if(finl.eof() == 0) exit(1);
- (d) close(f1);
- (e) infile.open(argc);
- (g) sfinout.open(file,ios::in |ios::out| ios::ate);

Solution:

(a) Error : Improper use of ifstream.

Correction : ifstream infile ("DATA");

(b) Here you must give two argument for getline () function one is string type variable name and another is string size. Such as

```
char s[20];
getline(s, 20);
```

(c) If we write

```
if (finl, eof ( ) == 0)
exit (1);
```

it input only one character from file and then it will exit.

(d) close () is void type argument function so it will be fl.close ()

(e) argc is called argument counter and agrv is called argument vector so to open data file we can write the statement like this:

```
infile.open (argv[1]) :
```

(f) The argument file must write as " file" because this is string type.

Correction :

```
fstream sfinout;
ios::out, ios::ate); | sfinout.open ("file", ios::in
```

Programming Exercises

11.1 Write a program that reads a text file and creates another file that is identical except that every sequence of consecutive blank spaces is replaced by a single space.

Solution:

```
#include<iostream.h>
#include<fstream.h>
#include<stdio.h>
#include<string.h>
```

```
int main()
```

```

{
    ofstream of; // of is a object of ofstream file.
    of.open("hasib.txt"); // open a text document file named "hasib" to write.
    char c;
    ifstream inf("santo.txt"); // open a text document file named "santo" to read.
    inf.get(c); // retrieve a single character from "santo" file.

    while(inf) // is character is '/0' ? if not go inside loop
    {
        if(c==' ') // is character is a space? If yes execute "if" statement.
        {
            while(inf) // is character is '/0' ? if not go inside loop
            {
                inf.get(c); // retrieve a single character from "santo" file.
                if(c!=' ') // is character is not a space?
                {
                    // If yes go outside the while loop.
                    break; // this is just for skip the space.
                }
            }
            of<<" "; // write single space to "hasib" text document file.
            of<<c; // write next charcter to "hasib" text document file.
        }
        else
        of<<c; // write single character where no space exits.
        inf.get(c); // retrieve a single character from "santo" text document again.
    }
    return 0;
}

```

Note: Before you run this program you just create a '.txt' document to a particular drive where your compiler was installed.

11.2 A file contains a list of telephone numbers in the following form

John 23456

Ahmed 9876

.....

the names contain only one word and the names and telephone numbers are separated by white spaces. Write program to read this file and output the list in two columns. The names should be left justified and the numbers should be right justified.

Solution:

```
#include<iostream.h>
#include<fstream.h>
#include<string.h>
#include<iomanip.h>

int main()
{
    char name[100],n[100],number[100];

    ifstream  santo("robin.txt");
    cout<<" Enter your desired name to find mobile number : " ;
    cin>>n;

    again:
    santo>>name;
    if(!strcmp(name,n))
    {
        santo.getline(number,100);
        cout<<setw(-20)<<name<<setw(25)<<number<<"\n";

    }
    else
    {
        if(santo.eof()!=0)
        cout<<" Sorry your input name is not found in list \n";
        else
        goto again;
    }
    return 0;
}
```

Note: You just create a '.txt' document like as question no 11.1 which contains peoples name and phone number.

output

```
Enter your desired name to find mobile number : john
john 23456
```

11.3 Write a program that will create a data file containing the list of telephone numbers given in exercise in 11.2. Use a class object to store each set of data.

Solution:

```
#include<iostream.h>
#include<fstream.h>
#define size 5

class phone
{
    public:
    void set_data();
};

void phone:: set_data()
{
    ofstream santo("phone.txt");
    char *name[size]={ "sattar","santo","kamruzzaman","robin","kawser"};
    char *number[size]
    ={"01673050495","01723783117","01818953250","+214324513","+455652132"};
    for(int i=0;i<size;i++)
    {
        santo.setf(ios::left,ios::adjustfield);
        santo.width(20);
        santo<<name[i];
        santo.setf(ios::right,ios::adjustfield);
        santo.width(15);
        santo<<number[i]<<"\n";
    }
}

int main()
{
    phone book;
    book.set_data();

    return 0;
}
```

Note: When you run this program, It will create a txt document named 'phone'.

11.4 Write an interactive, menu-driven program that will access the file created in exercise 11.3 and implement the tasks:

(a) Determine the telephone number of the specified person.

- (b) Determine the name if a telephone number is known.
(c) Update, the telephone number, whenever there is a change.

Solution:

```
#include<iostream.h>
#include<fstream.h>
#include<string.h>
#include<iomanip.h>

int main()
{
    char name[100],n[100],m[100],number[100];

    ifstream santo("robin.txt");
    int test;
    cout<<" Press 1 to find mobile number of specified person\n "
    <<" Press 2 to find name of specified number \n"
    <<" Press 3 to update number \n"
    <<" What is your option ? : ";
    cin>>test;

    if(test==1)
    {
        cout<<" Enter the desired name : ";
        cin>>n;
        cout<<"\n";
        again1:
        santo>>name;
        if(!strcmp(name,n))
        {
            santo.getline(number,100);
            cout<<setw(-20)<<name<<setw(25)<<number<<"\n";

        }
        else
        {
            if(santo.eof()!=0)
            cout<<" Sorry your input name is not found in list \n";
            else
            goto again1;
        }
    }

    else if(test==2)
    {
```



```

cout<<" Enter the desired number : ";
cin>>n;
cout<<"\n";
again2:
santo>>name;
santo>>number;

if(!strcmp(number,n))
{
    cout<<setw(-20)<<number<<setw(25)<<name<<"\n";
}
else
{
    if(santo.eof()!=0)
        cout<<" Sorry your input number is not found in list \n";

    else
        goto again2;
}
}
else if(test == 3)
{
    ofstream hasib("modified.txt");
    cout<<"Enter the name whose number have to change : ";
    cin>>n;
    again3:
    santo>>name>>number;

    if(!strcmp(n,name))
    {
        cout<<" Enter changed mobile number of"<<name<<": ";
        cin>>m;
        hasib.setf(ios::left,ios::adjustfield);
        hasib.width(20);
        hasib<<name;
        hasib.setf(ios::right,ios::adjustfield);
        hasib.width(15);
        hasib<<m<<"\n";
        while(santo)
        {
            santo>>name>>number;
            hasib.setf(ios::left,ios::adjustfield);
            hasib.width(20);
            hasib<<name;
            hasib.setf(ios::right,ios::adjustfield);
            hasib.width(15);
            hasib<<number<<"\n";
        }
    }
}
}

```

```

else
{
    if(santo.eof()!=0)
    cout<<" Sorry your input name is not available \n";
    else
    {
        hasib.setf(ios::left,ios::adjustfield);
        hasib.width(20);
        hasib<<name;
        hasib.setf(ios::right,ios::adjustfield);
        hasib.width(15);
        hasib<<number<<"\n";
        goto again3;
    }
}

}

return 0;
}

```

During First Run :

output

Press 1 to find mobile number of specified person
 Press 2 to find name of specified number
 Press 3 to update number
 What is your option ?: 1
 Enter the desired name : john
 john 23456

During Second Run :

output

Press 1 to find mobile number of specified person
 Press 2 to find name of specified number
 Press 3 to update number
 What is your option ?: 3
 Enter the name whose number have to change : Ahmed
 Enter changed mobile number of Ahmed : 9876

Chapter 12

Review Questions

12.1: What is generic programming? How is it implemented in C++?

Ans: Generic programming is an approach where generic types are used as parameters in algorithms so that they work for variety of suitable data types and data structures.

It is implemented in C++ using class templates.

12.2: A template can be considered as a kind of macro. Then, what is the difference between them?

Ans: Templates are a feature of the C++ programming language that allows functions and classes to be operated with generic types. This allows a function or class to work on many different data types without being written for each one. Templates are of great utility to programmers in C++, especially when combined with multiple inheritance and operator overloading. The C++ STL provides many useful functions within a framework of connected templates.

On the other hand the # define directive is typically used to associate meaningful identifiers with constants, keywords and commonly used statements or expressions. Identifiers that represent constants are sometimes called “symbolic constant” or “manifest constant”.

Identifiers that represent statements or expressions are called “macros”. In this preprocessor documentation only the term ‘macro’ is used.

12.3: Distinguish between overloaded functions and function templates.

Ans: Basic difference is function body will not differ in function overloading.

12.4: Distinguish between the terms class template and template class.

Ans: Class template is a generic class. The class template definition is very similar to an ordinary class definition except the prefix template and the use of type T.

On the other hand a class created from a class template is called a template class. The process of creating a specific class from a class template is called instantiation.

12.5: A class (or function) template is known as a parameterized class (or function).

Comment.

Ans: When an object of specific type is defined for actual use, the template definition for that class is substituted with the required data type. Since a template is defined with a parameter that would be replaced by a specified data type at the time of actual use of the class or function, the templates are called parameterized classes or functions.

12.6: State which of the following definitions are illegal.

(a) template

```
class city  
{..... };
```

(b) template

```
class city  
{..... }
```

(c) template

```
class city  
{..... };
```

(d) template

```
class city  
{..... };
```

(e) class

```
class list  
{..... };
```

(f) class

```
class list  
{..... };
```

Ans:

(a) Ok

(b) Illegal (type name expected)

```
template < class P, class R, class s >
```

```
class city
```

```
{..... }
```

(c) Ok

(d) Ok

(e) Ok

(f) Ok

12.7: Identify which of the following junction template definitions are illegal.

(a) template
void fun(A, B)
{....};

(b) template
void fun(A, A)
{....};

(c) template
void fun(A, A)
{....};

(d) template
T fun(T, R)
{....};

(c) template
A fun(int A)
{....};

Ans:

- (a) Illegal
- (b) Ok
- (c) Ok
- (d) Ok

Debugging Exercises

12.1: Identify the error in the following program.

```
1 #include<iostream.h>
2 class Test
3 {
4     int intNumber;
5     float floatNumber;
6 public:
7     Test()
8     {
```

```

9     intNumber = 0;
10    floatNumber = 0.0;
11    }
12    int getNumber()
13    {
14        return intNumber;
15    }
16    float getNumber()
17    {
18        return floatNumber;
19    }
20};
21void main ()
22 {
23     Test objTest1;
24     objTest1.getNumber();
25 }

```

Solution:It show ambiguity error, because the compiler consider int getNumber() and float getNumber() as same function. It happened because you write objtest1.getNumber(); in the main() function

12.2: Identify the error in the following program.

```

1 #include<iostream.h>
2 template <class R1, class T2>
3 class Person
4 {
5     T1 m_t1;
6     T2 m_t2;
7 public:
8     Person(T1 t1, T2 t2)
9     {
10        m_t1 = t1;
11        m_t2 = t2;
12        cout << m_t1 << " " << m_t2 <<endl;
13    }
14    Person(T2 t2, T1 t1)
15    {
16        m_t2 = t2;
17        m_t1 = t1;
18        cout << m_t1 << " " << m_t2 <<endl;
19    }
20};
21void main ()
22 {
23     Person< int, float> objPerson1(1, 2.345);
24     Person<float, char> objPerson2(2.132, 'r');

```

```
25 }
```

Solution: Here the two functions ['person (T1 t1, T2, t2)' and 'person (T2 t2, T1, t1)'] are same. So you can write one of them.

12.3: Identify the error in the following program.

```
1 #include<iostream.h>
2 template<class T1, class T2>
3 T1& MinMax(T1 t1, T1 t2)
4 {
5     return t1 > t2 ? ta : t2;
6     cout << " ";
7 }
8 void main()
9 {
10    cout << ++MinMax(2, 3);
11 }
```

Solution: There is no error in this program. It will run successfully.

12.4: Find errors, if any, in the following code segment.

```
1 template<class T>
2 T max(T, T)
3 {..... };
4 unsigned int m;
5 int main()
6 {
7     max(m, 100);
8 }
```

/

Solution: First you declared T as int type data and then declared as unsigned int type. So it will show error. If you write this as,

unsigned int n = 100; [N.B must be in main() function]
and max (m, n);

then it will not show any error.

Note: Write 'return 0;' in the main() function.

Programming Exercises

12.1: Write a function template for finding the minimum value contained in an array.

Solution:

```
#include<iostream.h>
const int size=5;
template <class T>
class vector
{
    T v[size];
public:
    vector(){ }
    vector(T *b);
    void show();
    minimum(vector<T> &m);
};
template <class T>
vector<T>::vector(T *b)
{
    for(int i=0;i<size;i++)
        v[i]=b[i];
}
template<class T>
T vector<T>::minimum(vector<T> &m)
{
    int j=0;
    for(int k=1;k<size;k++)
    {
        if(m.v[j]>m.v[k])
            j=k;
    }
    return m.v[j];
}
template<class T>
void vector<T>::show()
{
    cout<<"("<<v[0];
    for(int i=1;i<size;i++)
        cout<<" "<<v[i];
    cout<<")";
    cout<<"\n";
}
int main()
{
    int x[size]={5,7,3,1,8};//size is 5;
    float y[size]={1.2,1.5,2.3,1.0,0.501};
    vector<int> v1;
    vector<float> v2;
```



```

        v1=x;
        v2=y;
        cout<<" minimum value = "<<v1.minimum(v1)<<" of array";
        v1.show();
        cout<<" minimum value = "<<v2.minimum(v2)<<" of array";
        v2.show();

    return 0;
}

```

output

```

minimum value = 1 of array(5,7,3,1,8)
minimum value = 0.501 of array(1.2,1.5,2.3,1,0.501)

```

12.2: Write a class template to represent a generic vector. Include member functions to perform the following tasks:

- To create the vector
- To modify the value of a given element
- To multiply by a scalar value
- To display the vector in the following form (10, 20, 30 ...)

Solution:

```

#include<iostream.h>
#include<iomanip.h>
template <class santo>
class vector
{
    float *p;
    int size;
    public:
    void creat_vector(santo a);
    void set_element(int i,santo value);
    void modify(void);
    void multiply(santo b);
    void display(void);
};
template <class santo>
void vector<santo>::creat_vector(santo a)
{
    size=a;
    p=new float[size];
}
template <class santo>
void vector<santo>::set_element(int i,santo value)
{

```

```

        p[i]=value;
    }

template <class santo>
void vector<santo> :: multiply(santo b)
{
    for(int i=0;i<size;i++)
        p[i]=b*p[i];
}
template <class santo>
void vector<santo>:: display(void)
{
    cout<<"p["<<size<<"] = ( ";
    for(int i=0;i<size;i++)
    {
        if(i==size-1)
            cout<<p[i];
        else
            cout<<p[i]<<" , ";
    }
    cout<<")"<<endl;
}

template <class santo>
void vector<santo>::modify(void)
{
    int i;
    cout<<" to edit a given element enter position of the element : ";
    cin>>i;
    i--;
    cout<<" Now enter new value of "<<i+1<<"th element : ";
    santo v;
    cin>>v;
    p[i]=v;
    cout<<" Now new contents : "<<endl;
    display();

    cout<<" to delete an element enter position of the element : ";
    cin>>i;
    i--;

    for(int j=i;j<size;j++)
    {
        p[j]=p[j+1];
    }
    size--;
    cout<<" New contents : "<<endl;
    display();
}

```

```

int main()
{
    vector<float> hasib;
    int s;
    cout<<" enter size of vector : ";
    cin>>s;
    hasib.creat_vector(s);
    cout<<" enter "<<s<<" elements one by one : "<<endl;
    for(int i=0;i<s;i++)
    {
        float v;
        cin>>v;
        hasib.set_element(i,v);
    }
    cout<<" Now contents : "<<endl;
    hasib.display();
    cout<<" to multiply this vector by a scalar quantity enter this scalar quantity : ";
    float m;
    cin>>m;
    hasib.multiply(m);
    cout<<" Now contents : "<<endl;
    hasib.display();
    hasib.modify();
    return 0;
}

```

output

enter size of vector : 4

enter 4 elements one by one :

4 3 2 5

Now contents :

p[4] = (4 , 3 , 2 , 5)

to multiply this vector by a scalar quantity enter this scalar quantity : 3

Now contents :

p[4] = (12 , 9 , 6 , 15)

to edit a given element enter position of the element : 2

Now enter new value of 2th element : 222

Now new contents :

p[4] = (12 , 222 , 6 , 15)

to delete an element enter position of the element :3

New contents :

p[3] = (12 , 222 , 15)

Mujib'S_World

<https://mujibsworld.wordpress.com>

***** MUJIB *****

USEFUL LINKS

1. <https://www.slideshare.net/mobile/fellowbuddy/computer-networks-lecturenotes>

Links of Different Subjects

1. C Language:

<https://www.youtube.com/playlist?list=PLoVVmGDgrrnRJaU5uVw1ZK4V-JDwXKm6R>

2. C++ Language:

https://www.youtube.com/playlist?list=PLoVVmGDgrrnS5_TiSg193ezTPd-Ukb25k

3. Java Programming Language:

<https://www.youtube.com/playlist?list=PLoVVmGDgrrnRoTyxlwhhBpqVP1hNdW9R1>

4. Python Programming Language:

<https://www.youtube.com/playlist?list=PLoVVmGDgrrnRWX96zfQZfOYvhxZbnjXBX>

5. Oracle 11g:

<https://www.youtube.com/playlist?list=PLoVVmGDgrrnSKUaV7Lddfa5Cp4CyGYsEA>

6. Operating System:

<https://www.youtube.com/playlist?list=PLoVVmGDgrrnT1upH8ajPJ91obEjC3XmaQ>

7. MySql Tutorial:

<https://www.youtube.com/playlist?list=PLoVVmGDgrrnT2cE6MQRf4UsqAAq-nAWFZ>

8. Unix/Linux Tutorial:

<https://www.youtube.com/playlist?list=PLoVVmGDgrrnTTbMIHd6on55ENnyhuyGAv>

9. Web Technology:

<https://www.youtube.com/playlist?list=PLoVVmGDgrrnRs1KhtqovspEfnUYnUjFkb>

10. DBMS Tutorial

<https://www.youtube.com/playlist?list=PLoVVmGDgrrnR1nwaOM-X9ykVbLSVpRQ-B>

11. HTML Tutorial

<https://www.youtube.com/playlist?list=PLoVVmGDgrrnRxTIXhwV1CDWHKUSNrWz4a>

12. Computer Organization:

<https://www.youtube.com/playlist?list=PLoVVmGDgrrnQx7HizoZ68mb1myhNY70EV>

13. Data Structure and Algorithm:

<https://www.youtube.com/playlist?list=PLoVVmGDgrrnQmbTYV62R6bjYxziny4Rqi>

14. Computer Science Concept:

<https://www.youtube.com/playlist?list=PLoVVmGDgrrnQuMX0pqGQ12-GDkQchalNI>

2. <https://mcqsets.com/mcq-questions-2/computer-fundamentals/computer-networks-mcq-questions/>
3. <https://engineeringinterviewquestions.com/computer-networks-multiple-choice-questions-and-answers/>
4. <http://examradar.com/short-question-answer-computer-network/>
5. <http://nrcmcn.blogspot.com/2016/08/long-answer-questions.html?m=1>
6. <https://www.geeksforgeeks.org/layers-of-osi-model/#referrer=https://www.google.com>
7. <https://www.guru99.com/basic-computer-network.html>
8. <https://www.slideshare.net/mobile/Shawon800/osi-model-72960964>
9. <https://www.sanfoundry.com/computer-networks-mcqs-basics/>
10. <https://affaircloud.com/computer-quiz-data-communication-networking-set-5/?amp#referrer=https://www.google.com>
11. https://www.proprofs.com/quiz-school/topic/amp/data-communication#aoh=15881450114015&csi=1&referrer=https%3A%2F%2Fwww.google.com&_tf=From%20%251%24s