3 C++

PROGRAMMING APPROACHES

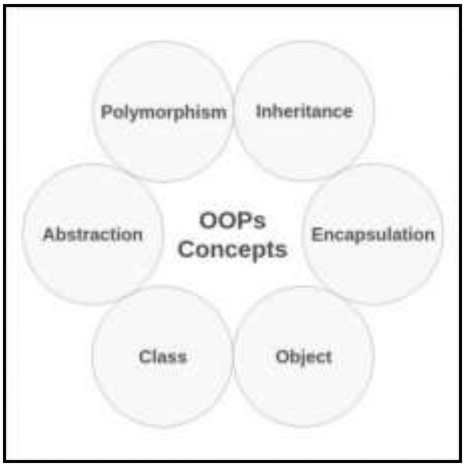
- **Unstructured programming.**
  The main program directly operates on global data. Here “main program” stands for a sequence of commands or statements, which modify data, which is global throughout the whole program. These programming techniques generate tremendous problems once the program gets sufficiently large.
- **Procedural programming**
  The main program coordinates calls to procedures and hand over appropriate data as parameters. A single program, which is divided into small pieces, called procedures. Modular programming allows grouping of procedures into modules.
- Modular programming
  The main program coordinates calls to procedures in separates modulus and hand over appropriate data as parameters. Each module can have its own data. This allows each module to manage an internal state, which is modified by call to procedures of this module.
- **Object oriented programming**
  Object–oriented programming solves some of the problems just mentioned. In contrast to the other techniques, we now have a web (group) of interacting objects, each house–keeping its own state. Objects of the program interact by sending messages to each other. It is most advanced approach and offers many advantages over other approaches just mentioned above.

OBJECT ORIENTED PROGRAMMING
What is OOP?
OOP stands for Object-Oriented Programming.

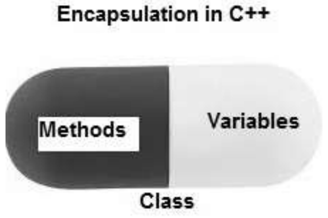
Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-oriented programming – As the name suggests uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Characteristics of an Object Oriented Programming language



Encapsulation: In normal terms, Encapsulation is defined as wrapping up of data and information under a single unit. In Object-Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them.



Encapsulation also leads to *data abstraction or hiding*. As using encapsulation also hidesthe data. In the above example, the data of any of the section like sales, finance or accounts are hidden from any other section.

Abstraction: Data abstraction is one of the most essential and important features of object- oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.
Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

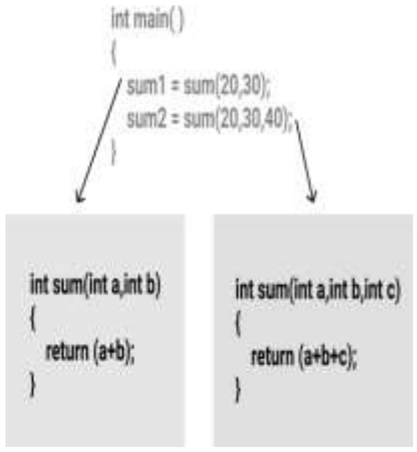
- *Abstraction using Classes*: We can implement Abstraction in C++ using classes. The class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to the outside world and which is not.
- *Abstraction in Header files*: One more type of abstraction in C++ can be header

files. For example, consider the pow() method present in math.h header file. Whenever we need to calculate the power of a number, we simply call the function pow() present in the math.h header file and pass the numbers as arguments without knowing the underlying algorithm according to which the function is actually calculating the power of numbers.

- **<u>Polymorphism:</u>** The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behaviour in different situations. This is called polymorphism.
  An operation may exhibit different behaviours in different instances. The behaviour depends upon the types of data used in the operation.
  C++ supports operator overloading and function overloading.
- *Operator Overloading*: The process of making an operator to exhibit different behaviours in different instances is known as operator overloading.
- *Function Overloading*: Function overloading is using a single function name to performdifferent types of tasks.

Polymorphism is extensively used in implementing inheritance.

Example: Suppose we have to write a function to add some integers, some times there are2 integers, some times there are 3 integers. We can write the Addition Method with the same name having different parameters, the concerned method will be called according to parameters.



```
int main( )
{
  sum1 = sum(20,30);
  sum2 = sum(20,30,40);
}
```

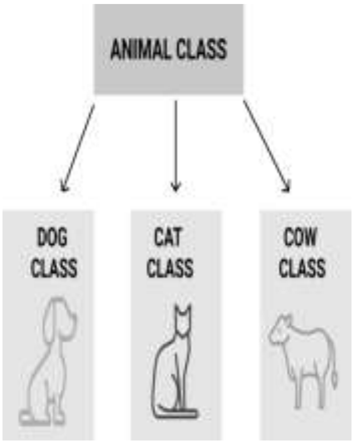
```
int sum(int a,int b)
{
  return (a+b);
}
```

```
int sum(int a,int b,int c)
{
  return (a+b+c);
}
```

<u>Inheritance:</u> The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object- Oriented Programming.

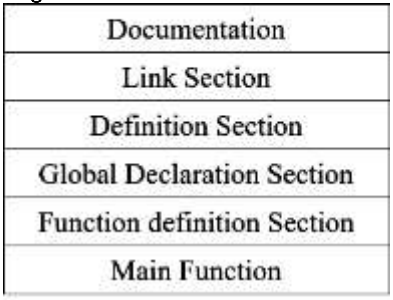
- **Sub Class**: The class that inherits properties from another class is called Sub class orDerived Class.
- **Super Class**:The class whose properties are inherited by sub class is called Base Classor Super class.
- **Reusability**: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusingthe fields and methods of the existing class.

Example: Dog, Cat, Cow can be Derived Class of Animal Base Class.



<u>Structure of C++ Program</u>

The <u>C++ program</u> is written using a specific <u>template structure</u>. The structure of the program written in C++ language is as follows:



Documentation
Link Section
Definition Section
Global Declaration Section
Function definition Section
Main Function

Skeleton of C Program

<u>Documentation Section</u>:

- This section comes first and is used to document the logic of the program that theprogrammer going to code.
- It can be also used to write for purpose of the program.
- Whatever written in the documentation section is the comment and is not compiled bythe compiler.
- Documentation Section is optional since the program can execute without them. Below isthe snippet of the same:

/*
- This is a C++ program to find the factorial of a number.
- The basic requirement for writing this program is to have knowledge of loops.
- To find the factorial of number iterate over range from number to one.

*/

Linking Section:

The linking section contains two parts:

Header Files:

- Generally, a program includes various programming elements like built-in functions, classes, keywords, constants, operators, etc. that are already defined in the standard C++ library.
- In order to use such pre-defined elements in a program, an appropriate header must beincluded in the program.
- Standard headers are specified in a program through the preprocessor directive #include. In Figure, the iostream header is used. When the compiler processes the instruction #include<iostream>, it includes the contents of the stream in the program. This enables the programmer to use standard input, output, and error facilities that are provided only through the standard streams defined in <iostream>. These standard streams process data as a stream of characters, that is, data is read and displayed in a continuous flow. The standard streams defined in <iostream> are listed here.
  #include<iostream>

Namespaces:

- A namespace permits grouping of various entities like classes, objects, functions, andvarious C++ tokens, etc. under a single name.
- Any user can create separate namespaces of its own and can use them in any otherprogram.
- In the below snippets, **namespace std** contains declarations for cout, cin, endl, etc.statements.

  using namespace std;
- Namespaces can be accessed in multiple ways:
- using namespace std;
- using std :: cout;

Definition Section:

- It is used to declare some constants and assign them some value.
- In this section, anyone can define your own datatype using primitive data types.
- In **#define** is a compiler directive which tells the compiler whenever the message isfound to replace it with “Factorial\n”.
- **typedef int K;** this statement telling the compiler that whenever you will encounter Kreplace it by int and as you have declared k as datatype you cannot use it as an identifier.

Global Declaration Section:

- Here, the variables and the class definitions which are going to be used in the programare declared to make them global.
- The scope of the variable declared in this section lasts until the entire program terminates.
- These variables are accessible within the user-defined functions also.

Function Declaration Section:

- It contains all the functions which our main functions need.
- Usually, this section contains the User-defined functions.
- This part of the program can be written after the main function but for this, write the function prototype in this section for the function which for you are going to write code after the main function.

```
// Function to implement the
// factorial of number num
intfactorial(k& num)
{
// Iterate over the loop from
// num to one
for(k i = 1; i <= num; i++) {fact *= i;
}
// Return the factorial calculated
returnfact;
}
```

Main Function:

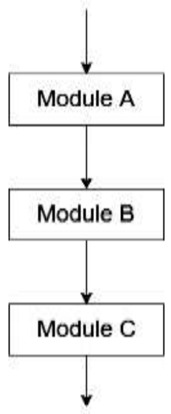
- The main function tells the compiler where to start the execution of the program. Theexecution of the program starts with the main function.
- All the statements that are to be executed are written in the main function.
- The compiler executes all the instructions which are written in the curly braces **{ }** whichencloses the body of the main function.
- Once all instructions from the main function are executed, control comes out of the mainfunction and the program terminates and no further execution occur.

Control Structures are just a way to specify flow of control in programs. Any algorithm or program can be more clear and understood if they use self-contained modules called as logic or control structures. It basically analyzes and chooses in which direction a program flows based on certain parameters or conditions. There are three basic types of logic, or flow of control, known as:

1. Sequence logic, or sequential flow
2. Selection logic, or conditional flow
3. Iteration logic, or repetitive flow

Let us see them in detail:

1. **Sequential Logic (Sequential Flow)**
   Sequential logic as the name suggests follows a serial or sequential flow in which the flow depends on the series of instructions given to the computer. Unless new instructions are given, the modules are executed in the obvious sequence. The sequences may be given, by means of numbered steps explicitly. Also, implicitly follows the order in which modules are written. Most of the processing, even some complex problems, will generally follow this elementary flow pattern.



Sequential Control flow

2. Selection Logic (Conditional Flow)

Selection Logic simply involves a number of conditions or parameters which decides one out of several written modules. The structures which use these type of logic are known as **Conditional Structures**. These structures can be of three types:

Single Alternative This structure has the form:

```
If (condition) then:
    [Module A]
[End of If structure]
```

Double Alternative This structure has the form:

```
If (Condition), then:
   [Module A]
            Else:
  [Module B]
[End if structure]
```

Multiple Alternatives This structure has the form:

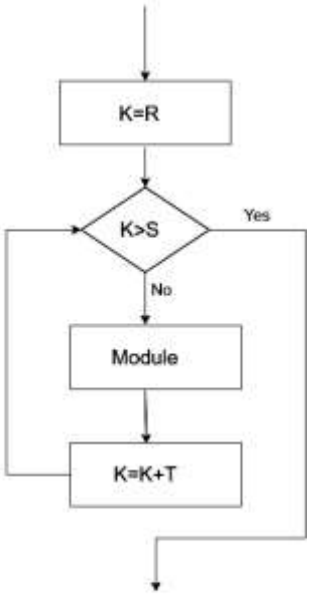
```
If (condition A), then:
  [Module A]
Else if (condition B), then:
  [Module B]
  ..
  ..
Else if (condition N), then:
  [Module N]
[End If structure]
```

3. Iteration Logic (Repetitive Flow)

The Iteration logic employs a loop which involves a repeat statement followed by a module known as the body of a loop.
The two types of these structures are:

- **Repeat-For Structure**
  This structure has the form:
  Repeat for i = A to N by I:
   [Module]
  [End of loop]

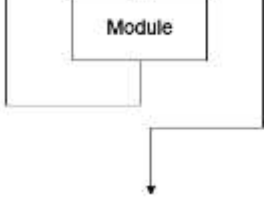
Here, A is the initial value, N is the end value and I is the increment. The loop ends when A>B. K increases or decreases according to the positive and negative value of I respectively.



Repeat-For Flow

Repeat-While Structure

It also uses a condition to control the loop. This structure has the form:
Repeat while condition:
    [Module]
[End of Loop]



Repeat While Flow

C++ What are Classes and Objects?

The building block of C++ that leads to Object-Oriented programming is a Class. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.
For Example: Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits,mileage are their properties.

- A Class is a user-defined data-type which has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behaviour of the objects in a Class.
- In the above example of class Car, the data member will be speed limit, mileage etc andmember functions can apply brakes, increase speed etc.

We can say that a **Class in C++** is a blue-print representing a group of objects which shares some common properties and behaviours.

Create a Class

To create a class, use the class keyword:

Example

Create a class called "MyClass":

```
class MyClass {       // The class
  public:             // Access specifier
    int myNum;        // Attribute (int variable)
    string myString;  // Attribute (string variable)
};
```

Example explained

- The class keyword is used to create a class called MyClass.
- The public keyword is an **access specifier**, which specifies that members (attributes and methods) of the class are accessible from outside the class. You will learn more about access specifiers later.
- Inside the class, there is an integer variable myNum and a string variable myString. When variables are declared within a class, they are called **attributes**.
- At last, end the class definition with a semicolon ;.

Scope rules for different members of the class

A class in C++ has a group of data and associated functions called data members and member functions, which are given one of the access modifiers, namely public, private and protected.

1) Public access modifier

All data members and member functions declared using public access modifier (i.e. those are defined in the public section of the class) are accessible by any function in a program.

2) Private access modifier

If you don't specify, any access modifier to the data members or member functions of class, they are by default private. These are hidden from outside world and they are by default private. These are hidden from outside world and they are normally used to implement data hiding concept of object oriented programming.

3) Protected access modifier

These can only be used by member functions and friend of a class as well as derived class member functions. They are similar to private members since they can't be accessed directly by non-member functions but can be used by derived ones. Therefore this access modifier is more restrictive than public but less restrictive than private.

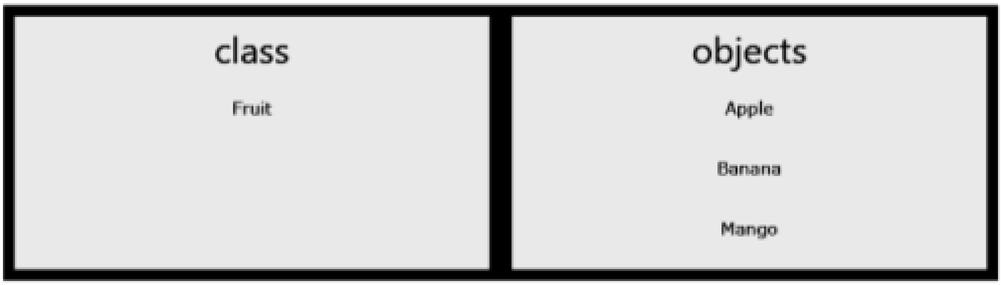
Object

An Object is an identifiable entity with some characteristics and behaviour. An Object is an instance of a Class. When a class is defined, no memory is allocated but whenit is instantiated (i.e. an object is created) memory is allocated.

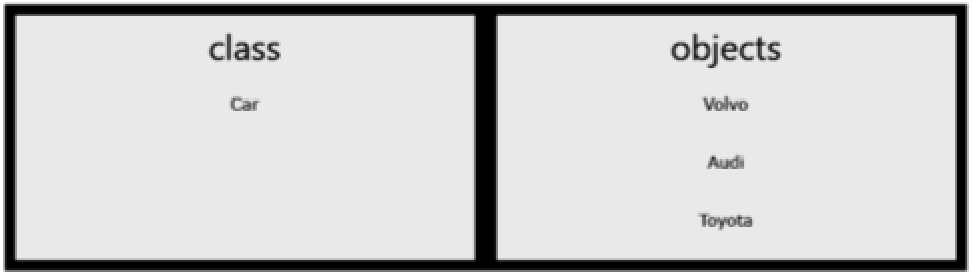
Object take up space in memory and have an associated address like a record in C.

When a program is executed the objects interact by sending messages to one another.

Each object contains data and code to manipulate the data. Objects can interact withouthaving to know details of each other's data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.
Look at the following illustration to see the difference between class and objects:



Another example:



Create an Object

In C++, an object is created from a class. We have already created the class named MyClass, so now we can use this to create objects.
To create an object of MyClass, specify the class name, followed by the object name.
To access the class attributes (myNum and myString), use the dot syntax (.) on the object:

Example

Create an object called "myObj" and access the attributes:

```
class MyClass {       // The class
  public:             // Access specifier
    int myNum;        // Attribute (int variable)
    string myString;  // Attribute (string variable)
};
int main() {
  MyClass myObj;  // Create an object of MyClass
  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";
  // Print attribute values
  cout << myObj.myNum << "\n";
  cout << myObj.myString;
  return 0;
}
```

Function Declaration:

- It contains all the functions which our main functions need.
- Usually, this section contains the User-defined functions.
- This part of the program can be written after the main function but for this, write the function prototype in this section for the function which for you are going to write codeafter the main function.

1. **Methods can be defined in class definitions.**
   e.g. in getdata () { return number 1; }
      C++ compilers treat these as inline functions: they try to expand the bodies of the functions where they are called.
2. Methods can be defined outside the class definitions.
   It is also possible to merely declare a method in a class definition,
   ```
   int get_data();
   ```
   and give the full definition later, outside the class: using scope resolution operator.
   ```
   int ABC:: getdata() { return number 1; }
   ```
   i.e. getdata is a function from class ABC. Because this is outside the class, we must qualify the function name with the class (ABC::).

The scope resolution operator is used to reference the global variable or member function that is out of scope. Therefore, we use the scope resolution operator to access the hidden variable or function of a program. The operator is represented as the double colon (::) symbol.

Program to access the hidden value using the scope resolution (::) operator

Program

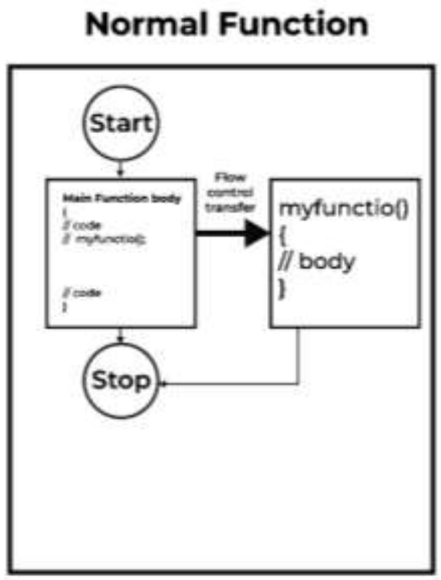
```
#include <iostream>
using namespace std;
// declare global variable
int num = 50;
int main ()
{
// declare local variable
int num = 100;
// print the value of the variables
cout << " The value of the local variable num: " << num;
// use scope resolution operator (::) to access the global variable
cout << "\n The value of the global variable num: " << ::num;
return 0;

}
```

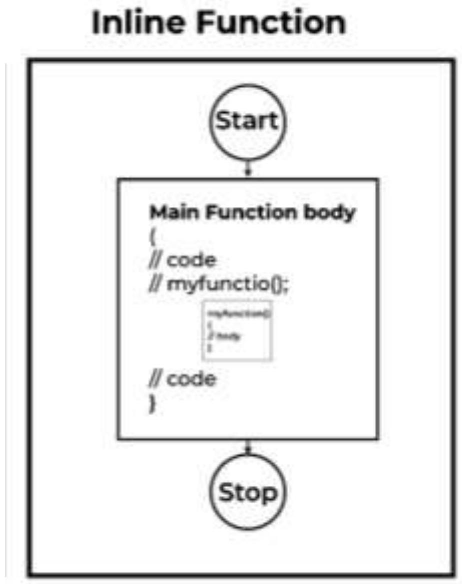
Inline Functions in C++

C++ provides inline functions to reduce the function call overhead. An inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of the inline function call. This substitution is performed by the C++ compiler at compile time. An inline function may increase efficiency if it is small.

Syntax:

```
inline return-type function-name(parameters)

{
   // function code
}
```

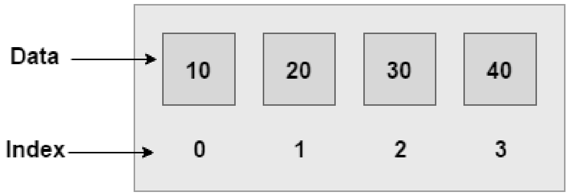




C++ Arrays

Like other programming languages, array in C++ is a group of similar types of elements that have contiguous memory location.

In C++ **std::array** is a container that encapsulates fixed size arrays. In C++, array index starts from 0. We can store only fixed set of elements in C++ array.



Advantages of C++ Array

- Code Optimization (less code)
- Random Access
- Easy to traverse data
- Easy to manipulate data
- Easy to sort data etc.

Disadvantages of C++ Array

- Fixed size

C++ Array Types

There are 2 types of arrays in C++ programming:

1. **Single Dimensional Array**
2. **Multidimensional Array**

A One-Dimensional Array in C++ programming is a special type of variable that can store multiple values of only a single data type such as int, float, double, char, structure, pointer, etc. at a contagious location in computer memory.
Here contagious location means at a fixed gap in computer memory.

A One-Dimensional Array is also known as 1D Array.

Suppose we want to store the age of 10 students. In that case, we have to declare 10 variables in our C++ program to store the age of 10 students.

Now here comes the use of a one-dimensional array. With the help of 1D Array, we will declare a single variable in our C++ program that can store the age of 10 students at a time.

Declaration Syntax of a One Dimensional Array in C++

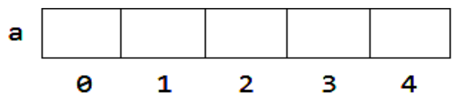
datatype variable_name[size];

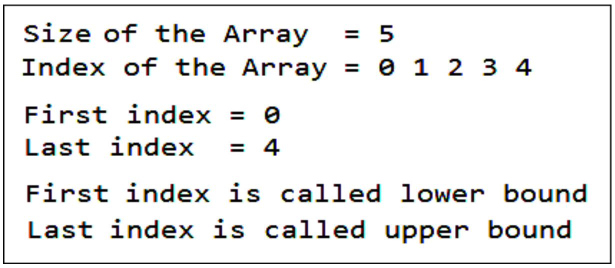
Here, size is the number of elements we want to store in the array.

Example:

int a[5];

Once we declare the 1D Array, it will look like as shown in the picture below:





```
Size of the Array  = 5
Index of the Array = 0 1 2 3 4

First index = 0
Last index  = 4

First index is called lower bound
Last index is called upper bound
```

In above image we can see that the name of the one dimensional array is a and it can store5 integer numbers. Size of the array is 5. Index of the array is 0, 1, 2, 3 and 4.

The first index is called Lower Bound, and the last index is called an Upper Bound. Upper Bound of a one dimensional is always Size – 1.

```
#include <iostream>
using namespace std;
int main()4. {
int arr[5]={10, 0, 20, 0, 30}; //creating and initializing array
//traversing array

for (int i = 0; i < 5; i++)
{
cout<<arr[i]<<"\n";
}
}
```

Output:

```
10
0
20
0
```

C++ Multidimensional Array Example

Declaration Syntax of a One Dimensional Array in C++

datatype variable_name[size][size];

Here, size is the number of elements we want to store in the array.

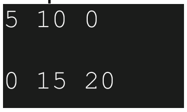
Example

int a[5][5];

Let's see a simple example of multidimensional array in C++ which declares, initializes and traverse two dimensional arrays.

```
#include <iostream>

using namespace std;
int main()

{
int test[3][3]; //declaration of 2D array
test[0][0]=5;
//initialization
test[0][1]=10;

test[1][1]=15;
test[1][2]=20;
test[2][0]=30;
test[2][2]=10;
//traversal
for(int i = 0; i < 3; ++i)
{
for(int j = 0; j < 3; ++j)
{
cout<< test[i][j]<<" ";
}
cout<<"\n"; //new line at each row
}
return 0;
}
```

Output:



```
5 10 0

0 15 20
```

<u>C++ Pointers</u>

<u>Creating Pointers</u>

You learned from the previous chapter, that we can get the **memory address** of a variable by using the & operator:

Example

```
string food = "Pizza"; // A food variable of type string
cout << food;                    // Outputs the value of food (Pizza)
cout << &food; // Outputs the memory address of food (0x6dfed4)
```

A **pointer** however, is a variable that **stores the memory address as its value**.

A pointer variable points to a data type (like int or string) of the same type and is created with the * operator. The address of the variable you're
working with is assigned to the pointer:

Example

string food = "Pizza"; // A food variable of type string
string* ptr = &food; // A pointer variable, with the name ptr,
thatstores the address of food
// Output the value of food
(Pizza)cout << food << "\n";
// Output the memory address of food (0x6dfed4)
cout << &food << "\n";
// Output the memory address of food with the pointer (0x6dfed4)cout <<
ptr << "\n";

OUTPUT

Pizza

0x6dfed4
0x6dfed4

Example explained

Create a pointer variable with the name ptr, that **points to** a string variable by using the asterisk sign * (string* ptr). Note that the type of the pointer has to match the type of the variable you're working with.
Use the & operator to store the memory address of the variable called food and assign it to the pointer.
Now, ptr holds the value of food's memory address.
There are three ways to declare pointer variables, but the first way is preferred:

1. string* mystring; //
Preferredstring

2. *mystring;

3. string * mystring;

C++ References

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 maybe used to refer to the variable.

Creating References in C++

Think of a variable name as a label attached to the variable's location in memory. You can then think of a reference as a second label attached to that memory location. Therefore, you can access the contents of the variable through either the original variable name or the reference. For example, suppose we have the following example –

```
int i = 17;
```

We can declare reference variables for i as follows.

```
int& r = i;
```

Read the & in these declarations as **reference**. Thus, read the first declaration as "r is an integer reference initialized to i" and read the second declaration as "s is a double reference initialized to d.". Following example makes use of references on int and double.

Example:

```
 #include <iostream>
 using namespace std;int main () {
    // declare
    simple variables
    int     i;
    double d;
    // declare reference
    variablesint& r = i;
    double& s = d;
    i = 5;
    cout << "Value of i : " << i << endl;
    cout << "Value of i reference : " << r << endl;
    d = 11.7;
    cout << "Value of d : " << d << endl;
    cout << "Value of d reference : " << s << endl;
    return 0;
 }
```

When the above code is compil*ed together and executed, it produces the following result –*

```
Value of i : 5
Value of i reference : 5
Value of d : 11.7
Value of d reference : 11.7
```

STRING REPRESENTATIONS

C++ provides following two types of string representations

- **The C-style character string.**
- **The string class type introduced with Standard C++.**

The C-Style Character String

The C-style character string originated within the C language and continues to be supported within C++. This string is actually a one-dimensional array of characters which is terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

The following declaration and initialization create a string consisting of the word "Hello".

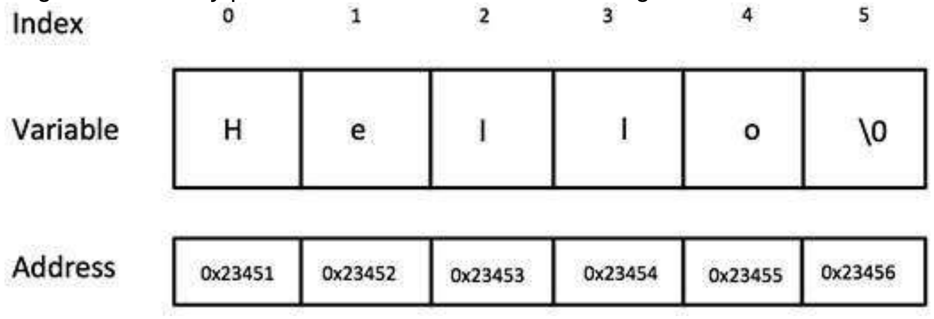
To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

If you follow the rule of array initialization, then you can write the above statement as follows –

char greeting[] = "Hello";

Following is the memory presentation of above defined string in C/C++ –



Actually, you do not place the null character at the end of a string constant. The C++ compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print above-mentioned string –

```
#include
<iostream>
using  namespace std
 int main ()
 {
 char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
 cout << "Greeting message: ";cout << greeting << endl;
return 0;
 }
```

When the above code is compiled and executed, it produces the following result –

```
Greeting message: Hello
```

C++ supports a wide range of functions that manipulate null-terminated strings

Sr.No	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.

4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 ifs1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

Following example makes use of few of the above-mentioned functions –

```
 #include <iostream>
 #include <cstring>
 using namespace std;
int main ()
{
char str1[10] = "Hello";
char str2[10] = "World";
char str3[10]; int len ;
strcpy( str3, str1); // copy str1 into str3
cout << "strcpy( str3, str1) : " << str3 << endl;
strcat( str1, str2); // concatenates str1 and str2
cout << "strcat( str1, str2): " << str1 << endl;
length = strlen(str1); // total lenghth of str1 after concatenation
cout << "strlen(str1) : " << len << endl;
return 0;
 }
```

When the above code is compiled and executed, it produces result something asfollows –

```
strcpy( str3, str1) : Hello
strcat( str1, str2): HelloWorld
strlen(str1) : 10
```

Constructor in C++

A constructor is a particular member function having the same name as the class name. It calls automatically whenever the object of the class is created.

Syntax:

The syntax of the constructor in C++ are given below.

```
1. class class_name
2. {
3. ……….
4.     public
5.     class_name ([parameter list])
6. {
7. ……………….
8. }
9. };
```

In the above-given syntax, class_name is the constructor's name, the public is an access specifier, and the parameter list is optional.

Example of Constructor:

```
#include <iostream.h>
#include <conio.h>
using namespace std;
class hello {                  // The class
public:                        // Access specifier
hello () {                  // Constructor
cout << "Hello World! Program in C++ by using
Constructor";
}
void display()
{
cout <<"Hello World!" <<endl;
}
};
int main()
{
hello myObj;
return 0;
}
```

There are four types of constructors used in C++.

- **Default constructor**
- **Parameterized constructor**
- **Copy constructor**
- **Dynamic constructor**

Default Constructor: A constructor is a class which accepts no parameter and is called a default constructor. If there is no constructor for a class, the compiler implicitly creates a default constructor.

Parameterised Constructor: A constructor is a class that can take parameters and is called a parameterized constructor. It is used to initialize objects with a different set of values.

Copy Constructor: A particular constructor used for the creation of an existing object. Thecopy constructor is used to initialize the thing from another of the same type.

Dynamic Constructor: This type of constructor can be used to allocate the memory while creating the objects. The data members of an object after creation can be initialized, called dynamic initialization.

Destructor in C++?

Destructors have the same class name preceded by (~) tilde symbol. It removes and destroys the memory of the object, which the constructor allocated during the creation of an object.

Syntax:

The syntax of destructor in C++ are given below.

```
class class_name
{
……………..;
……………..;
public:
xyz();                    //constructor
~xyz();                   //destructor};
```

Here, we use the tilde symbol for defining the destructor in C++ programming.
The Destructor has no argument and does not return any value, so it cannot be overloaded.

Example of Destructor:

```
#include <iostream.h>
#include <conio.h>
using namespace std;
class Hello {
public:
//Constructor
Hello () {
cout<< "Constructor function is
called" <<endl;9.     }
//Destructor
~Hello () {
cout << "Destructor function is called" <<endl;
}
//Member function
void display() {
cout <<"Hello World!" <<endl;
}
};
int main(){
//Object created
Hello obj;
//Member function called
obj.display();
return 0;
}
```

Difference between Constructor and Destructor inC++ programming

Following table shows the various differences between constructor and destructor in the C++ programming language:

Basis	**Constructor**	**Destructor**
Purpose of use	To allocate memory to the object,we used a constructor in C++.	To deallocate the memory that the constructor allocated to an object for this purpose, we use the concept of destructor in C++.
Arguments	It may not contain arguments.	It cannot contain the arguments.
Calling	It is called automatically whenever the object of the class is created.	It is called automatically whenever the program terminates.
Memory	Constructor occupies memory.	The Destructor releases memory.
Return type	It has return types.	It doesn't have any return type.
Special symbol	While declaring constructor in the C++ programming language, there is no requirement of the special symbol.	While declaring a destructor in C++ programming language, a particular symbol is required, i.e., tilde symbol.
In numbers	We can use more than one constructor in our program.	We cannot use more than one destructor in the program.
Inheritance	It can be inherited.	It cannot be inherited.
Overloading	It can be overloaded.	It cannot be overloaded.
Execution Order	They are executed in successive order.	They are executed in the constructor's reverse order; basically, they are the inverse of the constructors.
Types	Constructor has four types: ○ Default constructor ○ Copy constructor ○ Parameterized constructor ○ Dynamic constructor	Destructors have noclasses.

Declaration	The following declaration is used for creating a constructor:
	class class_name
	{
		The following declaration is used for creating a destructor:
		class class_name
		{

	……….
	public
	class_name ([parameter list])
	{
	……………….
	}
	};
		……………;
		………
		…….;
		public:
		~xyz();
		{
		…………
		};

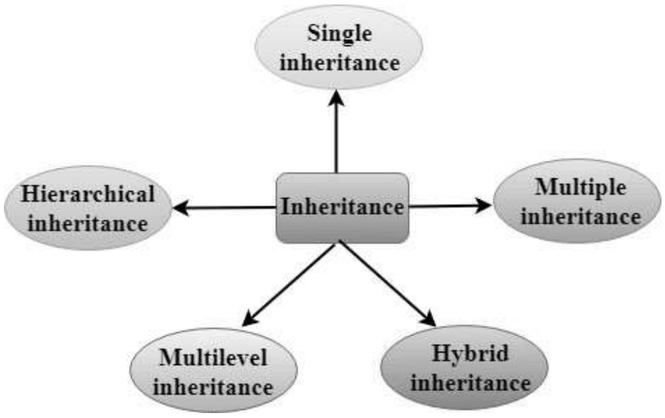
C++ Inheritance

In C++, inheritance is a process in which one object acquires all the properties and behaviours of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviours which are defined in other class.
In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

Types of Inheritance

C++ supports five types of inheritance:

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance



Derived Classes

A Derived class is defined as the class derived from the base class.The Syntax of Derived class:
class derived_class_name :: visibility-mode base_class_name
{
// body of the derived class.
}

Where,
derived_class_name: It is the name of the derived class.
visibility mode: The visibility mode specifies whether the features of the base classare publicly inherited or privately inherited. It can be public or private.

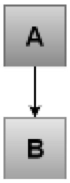
base_class_name: It is the name of the base class.

- When the base class is privately inherited by the derived class, public members of the base class becomes the private members of the derived class. Therefore, the public members of the base class are not accessible by the objects of the derived class only by the member functions of the derived class.
- When the base class is publicly inherited by the derived class, public members of the base class also become the public members of the derived class. Therefore, the public members of the base class are accessible by the objects of the derived class as well as by the member functions of the base class.

Note:

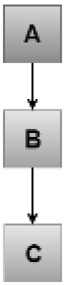
- In C++, the default mode of visibility is private.
- The private members of the base class are never inherited.

C++ Single Inheritance

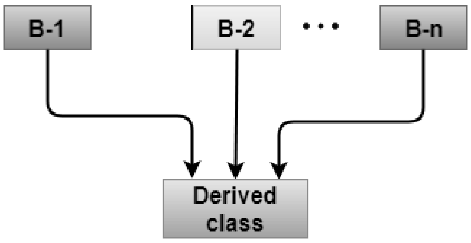


Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.

Where 'A' is the base class, and 'B' is the derived class.
C++ Multilevel Inheritance
Multilevel inheritance is a process of deriving a class from another derived class.



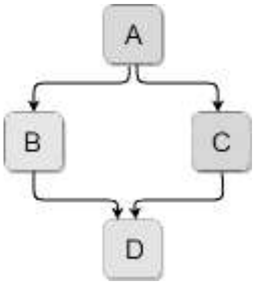
C++ Multiple Inheritance



Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.

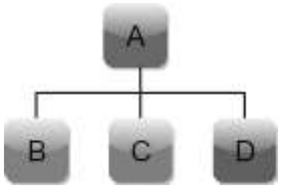
C++ Hybrid Inheritance

Hybrid inheritance is a combination of more than one type of inheritance.



C++ Hierarchical Inheritance

Hierarchical inheritance is defined as the process of deriving more than one class from a base class.



C++ Syntax for Operator Overloading

```
class ClassName{
private:
...............
...............
...............
public:
...............
...............
<ReturnType> operator <Sign> (arguments){
...............
...............
}
...............
...............
};
```

Operator Overloading in Unary operators

First, let's understand what the unary operators are. As the name suggests, unary operators work on a single operand like Increment (++), Decrement (\-\-), logical not (!).

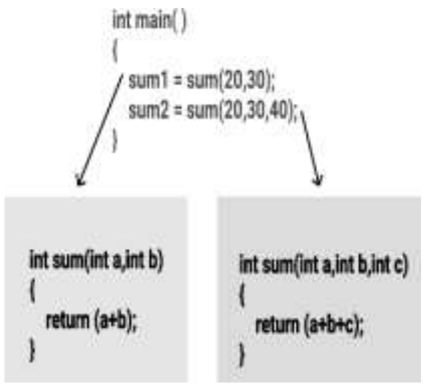
Example

```
#include <iostream>using namespace std;
class TestClass {
private:

public:
int count;
TestClass() :
count(5)
{
void operator --()
{
count = count - 3;

}
void Display()
{
cout << "Count: " << count;
}
};
int main() {
TestClass tc;
--tc;
tc.Display();
return 0;
}
```

Function Overloading

Function overloading is using a single function name to performdifferent types of tasks. Polymorphism is extensively used in implementing inheritance.

Example: Suppose we have to write a function to add some integers, some times there are2 integers, some times there are 3 integers. We can write the Addition Method with the same name having different parameters, the concerned method will be called according to parameters.



```
int main( )
{
   sum1 = sum(20,30);
   sum2 = sum(20,30,40);
}
```

```
int sum(int a,int b)
{
   return (a+b);
}
```

```
int sum(int a,int b,int c)
{
   return (a+b+c);
}
```

Program

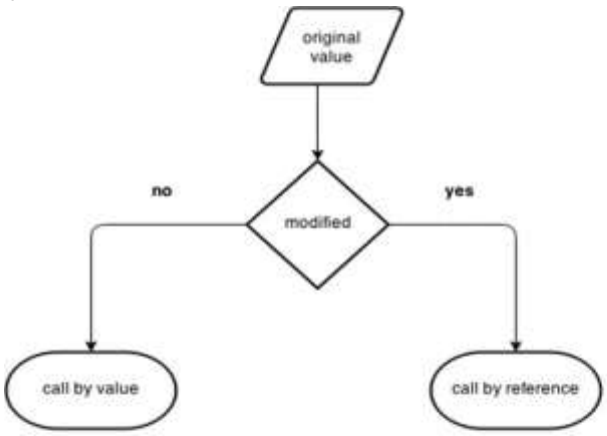
```
#include <iostream>
using namespace std;
void add(int a, int b)
{
  cout << "sum = " << (a + b);
}
void add(double a, double b)
{
    cout << endl << "sum = " << (a + b);
}
 // Driver code
int main()
{
    add(10, 2);
    add(5.3, 6.2);
     return 0;
}
```

The following table demonstrates the difference between run time polymorphism and compile-time polymorphism:

Compile Time Polymorphism	Run time Polymorphism
In Compile time Polymorphism, the call is resolved by the compiler.	In Run time Polymorphism, the call is not resolved by the compiler.
It is also known as Static binding, Early binding and overloading as well.	It is also known as Dynamic binding, Late binding and overriding as well.
Method overloading is the compile-time polymorphism where more than one methods share the same name with different parameters or signature and different return type.	Method overriding is the runtime polymorphism having the same method with same parameters or signature but associated with compared, different classes.
It is achieved by function overloading and operator overloading.	It is achieved by virtual functions and pointers.
It provides fast execution because the method that needs to be executed is known early at the compile time.	It provides slow execution as compare to early binding because the method that needs to be executed is known at the runtime.
Compile time polymorphism is less flexible as all things execute at compile time.	Run time polymorphism is more flexible as all things execute at run time.
Inheritance is not involved.	Inheritance is involved.

Call by value and call by reference in C++

There are two ways to pass value or data to function in C language: call by value and call by reference. Original value is not modified in call by value but it is modified in call by reference.



Let's understand call by value and call by reference in C++ language one by one.

Call by value in C++

In call by value, **original value is not modified.**

In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

Let's try to understand the concept of call by value in C++ language by the example given below:

```
#include <iostream>
using namespace std;
void change(int data);
int main()
{
int data =3;
change(data);
cout << "Value of the data is: " << data<< endl;
return 0;
}
void change(int data)
{
data = 5;
}
```

Output: Value of the data is: 3

Call by reference in C++

In call by reference, original value is modified because we pass reference (address).

Here, address of the value is passed in the function, so actual and formal arguments share the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.

Note: To understand the call by reference, you must have the basic knowledge of pointers.
Let's try to understand the concept of call by reference in C++ language by the example given below:40

```
#include<iostream>
using namespace std;
void swap(int *x, int *y)
{
int swap;
 swap=*x;
*x=*y;
*y=swap;
}
int main()

{
int x=500, y=100;

swap(&x, &y); // passing value to function
cout<<"Value of x is: "<<x<<endl;
cout<<"Value of y is: "<<y<<endl;
return 0;

}
```

Output:

```
Value of x is: 100
Value of y is: 500
```

Difference between call by value and call by reference in C++

No.	Call by value	Call by reference
1	A copy of value is passed to thefunction	An address of value is passed tothe function
2	Changes made inside the functionis not reflected on other functions	Changes made inside the function is reflected outside thefunction also
3	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memorylocation

Virtual Function in C++

A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve Runtime polymorphism
- Functions are declared with a **virtual** keyword in base class.
- The resolving of function call is done at runtime.

Rules for Virtual Functions

- Virtual functions cannot be static.
- A virtual function can be a friend function of another class.
- Virtual functions should be accessed using pointer or reference of base class type to achieve runtime polymorphism.
- The prototype of virtual functions should be the same in the base as well as derived class.
- They are always defined in the base class and overridden in a derived class
- A class may have virtual destructor but it cannot have a virtual constructor.
- Virtual function in a base class must be defined, even though it is not used.
- The base pointer can point to any type of derived object.

Friend function

A **friend class** can access private and protected members of other classes in which it is declared as a friend. It is sometimes useful to allow a particular class to access private and protected members of other classes. For example, a Linked List class may be allowed to access private members of Node.
We can declare a friend class in C++ by using the **friend** keyword.

Syntax:
friend class class_name;   // declared in the base class

Example:

Characteristics of a Friend function:

- The function is not in the scope of the class to which it has been declared as a friend.
- It cannot be called using the object as it is not in the scope of that class.
- It can be invoked like a normal function without using the object.
- It cannot access the member names directly and has to use an object name and dot membership operator with the member name.
- It can be declared either in the private or the public part.

Program

```
// Add members of two different classes using friend functions
#include <iostream>
using namespace std;
// forward declaration
class ClassB;
class ClassA {
   public:
      // constructor to initialize numA to 12
      ClassA() : numA(12) {}
    private:
      int numA;
       // friend function declaration
       friend int add(ClassA, ClassB);
};
class ClassB {
   public:
      // constructor to initialize numB to 1
      ClassB() : numB(1) {}
      private:
      int numB;
       // friend function declaration
      friend int add(ClassA, ClassB);
};
// access members of both classes
int add(ClassA objectA, ClassB objectB) {
   return (objectA.numA + objectB.numB);
}
int main() {
   ClassA objectA;
   ClassB objectB;
   cout << "Sum: " << add(objectA, objectB);
   return 0;
}
```

Output

```
Sum: 13
```

Type Conversion in C++

A type cast is basically a conversion from one type to another. There are two types of type conversion:

1. **Implicit Type Conversion** Also known as 'automatic type conversion'.
   - Done by the compiler on its own, without any external trigger from the user.
   - Generally takes place when in an expression more than one data type is present.

In such condition type conversion (type promotion) takes place to avoid lose of data.

- All the data types of the variables are upgraded to the data type of the variable with largest data type.

bool → char → short int → int → unsigned int → long → unsigned → long long → float → double → long double

- It is possible for implicit conversions to lose information, signs can be lost (whensigned is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

Example of Type Implicit Conversion:

```
// An example of implicit conversion
#include <iostream>
using namespace std;
int main()
{
int x = 10; // integer x
char y = 'a';
// character y
// y implicitly converted to int. ASCII
// value of 'a' is 97
x = x + y;// x is implicitly converted to float
float z = x + 1.0;
cout << "x = " << x << endl
<< "y = " << y << endl
<< "z = " << z << endl;
return 0;
}
```

Output:

```
x = 107
y = a
z = 108
```

Explicit Type Conversion: This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type. In C++, it can be done by two ways:

- **Converting by assignment:** This is done by explicitly defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.

Syntax:

(type) expression

where *type* indicates the data type to which the final result is converted.

Example:

```
// C++ program to demonstrate
// explicit type casting
#include <iostream>
```

```
using namespace std;
int main()
{
double x = 1.2;
// Explicit conversion from double to int
int sum = (int)x+ 1;
cout << "Sum = " << sum;
return 0;
}
```

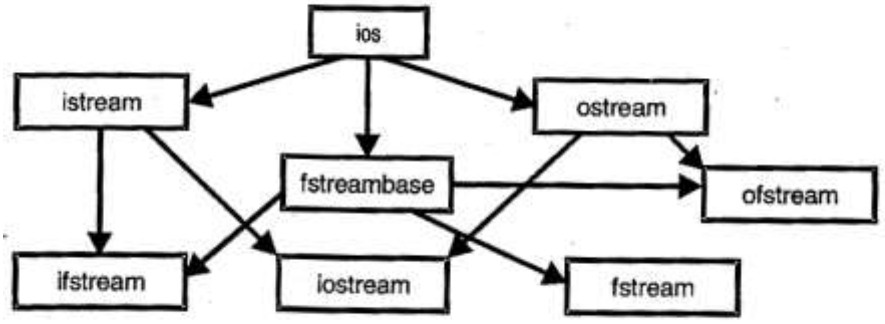
Output:

```
Sum = 2
```

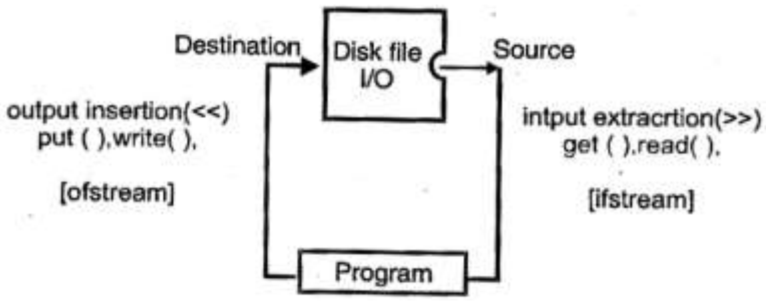
__File and Streams__

- A steam is a general name given to the flow of data.
- Different streams are used to represent different kinds of data flow.
- Each steam is associated with a particular class, containing member functions and definitions for dealing with that particular flow of data.
  e.g. Extraction operator >> is a member of istream class while insertion operator << is a member ostream class and both of these classes are derived from ios class. ifstream represents i/p disk files.

Stream class hierarchy:



Classes used for i/p and o/p to the video display and keyboard are declared in header file, iostream.h while classes used for file I/O are declared in the file fstream.h



<u>WORKING WITH FILE TYPE</u>

Many programming scenarios require handling a large amount of data, and some secondary storage has to be used to store it. The data is stored in the secondary device using the concept of files. Files are a collection of related data stored in a particular storage device.

C++ programs can be written to perform read and write operations on these files.

Working with files generally requires the following kinds of data communication methodologies:

- Data transfer between console units.
- Data transfer between the program and the disk file.

Here are the lists of standard file handling classes:

<u>Header files</u>

- ofstream: This file handling class in C++ signifies the output file stream and is applied to create files for writing information to files.
- ifstream: This file handling class in C++ signifies the input file stream and is applied for reading information from files.
- fstream: This file handling class in C++ generally signifies the file stream and can represent both ofstream and ifstream.

All three above classes are derived from fstream base and the associated iostream class and are explicitly designed to handle disk files.

<u>Detecting EOF (End Of File)</u>

An ifstream object (f1 here) has a value that can be tested for various error conditions. If a conditions is true, object returns a 0 value, otherwise non−zero. One of these conditions is end of file (EOF). The EOF is a signal sent to the program from h/w, when a read or write operation has reached to the end of the file.

The program checks for the EOF in the while loop so that it can stop reading after the last string. The value returned by a stream pointer is actually a pointer, but the address returned has no significance except to be tested for a 0/non−zero value.

<u>Character I/O</u>

Get ( ) and put ( ) functions, which are members of istream and ostream respectively are used to i/p and o/p single character off a time.

<u>Opening and Closing a File in C++</u>

If programmers want to use a disk file for storing data, they need to decide the following things about the file and its intended use. These points that are to be noted are:

- A name for the file.
- Data type and structure of the file.
- Purpose (reading, writing data).
- Opening method.
- Closing the file (after use).

Files can be opened in two ways. They are:

1. Using the constructor function of the class
2. Using member function open of the class

Opening a File in C++

The first operation generally performed on an object of one of these classes to use a file is the procedure known as opening a file. An open file is represented within a program by a stream, and any input or output task performed on this stream will be applied to the physical file associated with it. The syntax of opening a file in C++ is:

Syntax:

open (filename, mode);

There are some mode flags used for file opening. These are:

- ios::app: append to end of file.
- ios::ate: open a file in this mode for output and read/write control to the end of the file.
- ios::in: open a file in this mode for reading.
- ios::out: open a file in this mode for writing.
- ios::trunc: when any file already exists, its contents will be truncated before the file opening(delete contents of file if it exists).
- ios::nocreate: open fails if the file does not exist.
- ios::noreplace: open fails if the file already exist.

Closing a File in C++

When any C++ program terminates, it automatically flushes out all the streams, releases all the allocated memory, and closes all the opened files. But it is good to use the close() functionto close the file-related streams, which are a member of ifsream, ofstream, and fstream objects.

The structure of using this function is:

Syntax:

void close();

General functions used for File handling

1. open(): To create a file.
2. close(): To close an existing file.
3. get(): to read a single character from the file.
4. put(): to write a single character in the file.
5. read(): to read data from a file.
6. write(): to write data into a file.

Reading from and writing to a File

While coding in C++, programmers write information to a file from the program using the stream insertion operator (<<) and reads the data using the stream extraction operator (>>).The only difference is that for files, programmers need to use an ofstream or fstream objectinstead of the cout object and ifstream or fstream object instead of the cin object.

Example:

```
#include <iostream>
#include <fstream.h>
void main ()
{
ofstream file;
file.open ("egone.txt");
file << "Writing to a file in C++…..............";
file.close();
getch();
}
```

Another program for file handling in C++:

Example:

```
#include <iostream>
#include<fstream.h>
void main()
{
char c,fn[10];
cout<<"Enter the file name.";
cin>>fn;
ifstream in(fn);
if(!in)
{
cout<<"Error! File Does not Exist";getch();
return;
}
cout<<endl<<endl;while(in.eof()==0)
{
in.get(c);
cout<<c;
}
getch();

}
```

1. Write a program in C++ to display a Fibonacci series of 15 terms.

OR

Write a program in C++ to display a Fibonacci series of 20 terms (use n < = 18 in this case)

Ans.

```
#include<iostream.h>
Void main ( )
{
int f0, f1, f, n;
f0 = 0;
f1 = 1;
```

```
        clrscr ( );
        cout<< “Fibbonacci series\n”;
        cout<< “n” <<f0<<”\n”<<f1;
        for (n = 1; n<=13; n++)
        {
            f = f0 + f1;
            cout<< “\n” <<f;
            f0 = f1;
            f1 = f;
        }
    }
```

2. Write a program in C++ to calculate and print factorial of first 10 numbers.

Ans.

```
//C++ program to calculate and print factorial of first 10 numbers
#include<iostream.h>
#include<conio.h>
Void main()
{
    Int fact, n, I;
    clrscr( );
    count<< “Number” <<”\t” << “Factorial”;
    for (n = 1; n<=10; n++)
    {
        fact = 1;
        for (I = 1; i<=n; i++)
        {
        Fact = fact*I;
        }
    cout<<endl<<n<< “\t”<<fact;
    }
}
```

3. Write a C++ program to find factorial of a natural number input during program execution.

Ans.

```
    //Program to find factorial of a number
#include<iostream.h>
#include<conio.h>
void main ( )
{
    int fact, number;
    clrscr ( );
    fact = 1;
    cout<< “Enter the number” <<endl;
    cin >> number;
    for (I = 1; I < = number; i++)
    {
    fact = fact * I;
    }
    cout << “The factorial of a inputted number is:” << fact;
}
```

4. What is an Armstrong number? Write a program in C++ to check whether the given number is armstrong or not.

Ans. Armstrong number:

"If sum of the cubes of digits of a number is equal to the original number, then the number is said to be an Armstrong number".

e.g. 153 is an Armstrong number.

```
//C++ Program to find whether the number is Armstrong or not.
#include<iostream.h>
#include<conio.h>
Void main()
{
int n, dn, temp, d;
cout<< "Enter a number";
cin>>n;
dn = n;
temp = 0;
while (dn! = 0)
{
    d = dn%10;
    temp=temp+(d*d*d);
    dn = dn/10;
}
If(n ==temp)
{
count<<n<<"is Armstrong no.";
}
else
    {
    cout<<n<<" is not an armstrong number";
    }
return 0;
}
```

5. Write a program to perform arithmetic calculations such as addition, subtraction, multiplication or division, depending on choice using switch statement.

Ans.
```
//C++ program to generate simple calculator
#include<iostream.h>
#include<conio.h>
void main ()
{
float a, b, result;
int ch;
clrscr();
cout<< "Enter two numbers";
cin>>a>>b;
cout<<"\n1–addition\n 2–subtraction\n 3–multiplication\ n 4–division";
cout<< "Enter Your Choice:";
cin>>ch;
switch (ch)
{
     case 1:
          result=a+b;
          cout<<”Sum is” << result;
          break;
     case 2:
          result = a–b;
          cout<< “Difference is”<<result;
          break;
     case 3:
          result=a*b;
          out<< “Product is”<<result;
          break;
     case 4:
          result=a/b;
          cout<< “Division is” << result;
          break;
     default:
          cout<< “invalid choice”;
          break;
     }
     }
```

6. What is a recursive function? Write a program in C++ to calculate addition of first numbers using recursive function.

Ans. Recursive function:

“A function which is called within the body of the same function itself is called as recursive function,”

//Program to calculate addition of first n numbers

```
#include<iostream.h>
#include<conio.h>
int add (int);
void main ()
{
Int n, sum;
cout<< “Enter a number \n”;
cin>>n;
sum=add(n);
cout<< “Addition of first” <<n<< “numbers is”<<sum;
}
//function to calculate addition
Int add (int x)
{
Int S = 0;
if (x!=0)
{
S=x+add(x–1);
}
Return(S);
}
```

7. Write a C++ program to accept a set of 10 numbers and print the numbers using pointers.

Ans: //Program to print numbers using pointers.

```
#include<iostream.h>
#include<conio.h>
Void main()
{
Int a[10], i, *ptr;
Clrscr();
count<< “Enter 10 number” <<endl;
for (i = 0; i<=9; i ++)
{
Cin>>a[i];
}
ptr=&a[0]; //or use ptr = &a;
cout<< “\n The numbers are \n”;
for (i=0; i<=9; i++)
{
Cout<<*ptr<<”\n”;
ptr++;
}
}
```

8. Write a program in C++ to read a line of text and to count number of words in a text.

Ans. //Count words in a line of text

```
#include<isostream.h>
#include<string.h>
void main()
{
    char line [80];
    int count = 1, len, I;
    out<<”\n Entera line of text\n”;
    cin.getline (line, 80);
    len = strlen (line);
    for (i = 0; I < = len; i++)
{
    if (line [i] = = ‘ ‘)
    Count++;
}
    cout<<”No. of words are”;
    cout<<count;
    return 0;
}
```

9. Write a C++ program to find the Greatest Common Divisor of two numbers. Define a method find to accept the values and calculate GCD of two numbers and print the GCD value.

Ans. //Program to find GCD value

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

```
class gcd
{
    int a, b;
    public:
        void find ( );
};
Void gcd :: find (void)
{
    cout<<”Enter the value of a and b\n”;
    cin>>a>>b;
    while (a ! = b)
    {
    if (a > b)
        a = a – b;
    if (b > a)
        b = b – a;
    }
        cout << “The gcd is:” <<a;
}
Void main ( )
{
    gcd obj1;
    obj1.find ( );
    return 0;
}
```

10. Write a program in C++ to calculate Fibonacci series of ‘n’ numbers using constructor.

Ans.

```
//Program to generate Fibonacci series
#include<isostream.h>
#include<conio.h>
class fibonacci
{
    private:
    long int f0,f1, fib;
    public:
    fibonacci (void);
    void process (void);
    void display (void);
};
fibonacci::Fibonacci::process (void)
{
    fib=f0+f1;
    f0=f1;
    f1=fib;
}
void fibonacci::display (void)
{
    cout<<fib<<”\t”;
}
Void main ( )
{
     int I, n;
     fibonacci F;
     cout<<"/n Enter number of elements"<<endl;
     cin>>n;
     for (i = 1; i<=n; i++)
{
     F.process( );
     F.display( );
}
}
```

11. Implement a circle class. Each object of this class will represent a circle, accepting its radius value as float. Include an area () function which will calculate the area of circle.

Ans.
```
//C++program to implement a circle class
#include<isosteam.h>
class circle
{
          float a;
          float r;
     public:
          void area (void);
};
void circle: : area (void)
{
     cout<< "Enter radius of circle";
     cin>>r;
     a = 3.142 *r*r;
     cout<<"The area of a circle is";
     out<<a;
}
     void main ()
{
     circle C;
     C.area ();
}
```

12. Write a C++ program to accept a number and test whether it is prime or not.

Ans.
```
//C++ program to test whether the inputted number is prime or not
#include<iostream.h>
void main ()
{
          int prime, C = 0;
          cout << "Enter the number";
          cin >> prime;
          for (int i = 2; i < prime; i++)
     {
          if (prime% i ==0)
```

```
                C = 1;
        }
        if (C ==0)
            cout << " The number" <<prime
                << "is prime number";
        else
            cout << "The number" <<prime
                << is not a prime number";
    }
```

13. Write an object oriented program in C++ to read an integer number and find the sum of digits of integer [Hint: input 125 output 8 i.e. 1 + 2 + 5 = 81]

Ans.

```
#include<iostream.h>
#include<conio.h>
void main()
{
int val, num, sum = 0;
cout<< "Enter the number:";
cin>> val;
num = val;
while (num ! = 0)
{
    sum = sum + num % 10;
    num = num /10;
}
    cout<< "The sum of the digits of "<<val<< "is"<<sum;
}
```

14. Write a C++ Program to exchange the contents of two variables using call by reference.

Ans.

```
//Program to exchange the contents of two variables using call by//reference
#include<iostream.h>
void swap (int* , int*);// function prototype
void main ()
{
int a, b;
cout << "Enter the values";
cin >> a >> b;
cout << "Before Swapping";
cout << "a =" << a;
cout << "b=" <<b;
swap (& a, & b);      // call by reference
cout << "After Swapping";
cout << "a =" << a;
cout << "b = " << b;
}
void swap (int *a, int *b) // function definition
{
int temp;
temp = *a;   // assign the value at address a to temp
*a = *b;        // put the value at b into a
*b = temp;   // put the value at temp into b
}
```

15. Write a program in C++ to read a set of numbers from keyboard and find out the largest number in the given array.

Ans.
```
//Program to find out largest number from the given array
# include <iostream.h>
void main ()
{
     int num [10 ], maximum;
     cout << “Enter the number”;
     for (int i = 0; I < 10; i++)
     cin >> num [ i ];
     maximum = num [0];
     for (int j = 1; j < 10; j++)
{
     if (maximum < num [ j ])
     max = num [ j ];
}
     cout << “The largest number in the array is” << maximum;
}
```

16. Write a program in C++ to find Greatest Common Divisor (GCD) of two natural numbers.

Ans.
```
// To find Greatest Common Divisor of two natural numbers.
#include <iostream.h>
void main ()
{
int n1, n2;
cout <<”Enter the two natural numbers”;
cin >> n1 >> n2;
while (n1 ! = n2)
{
if (n1 > n2)
n1 = n1 – n2;
if (n2 > n1)
n2 = n2 – n1;
}
Cout << “The GCD is :” << n1;
}
```

17. Write a program in C++ that inputs and stores 10 numbers in an array and prints the sum and average of the array elements.

Ans.
```
//Program to print the sum and average of the array elements.
# include <iostream.h>
Void main ()
{
int num [ 10 ], sum;
float avg = 0.0;
cout <<"Enter the 10 elements";
for (int i = 1; i < = 10; i++)
cin >> num [ i ];
sum = 0;
for (i = 1; i < = 10; i ++)
{
sum = sum + num [ i ];
}
avg = sum /10;
cout <<"The sum of numbers is:"<<sum << endl;
cout <<"The average of the array element is: " << avg;
}
```

18. Write a C++ program to find the smallest of four given integers using min ( ) function to return the smallest of four given integers. int min (int, int, int, int)

Ans. //Program to find the smallest of four given integers using function.

```
# include <iostream.h>
void main ()
{
     int a, b, c, d, small;
     int main (int, int, int, int); //Prototype)
     cout << "Enter the four numbers:" <<endl;
     cin >> a >> b >> c >> d;
     small = min (a, b, c, d); //function call
     cout <<"The smallest number is:" <<small;
}
//function definition
int min (int n1, int n2, int n3, int n4)
{
     int low;
     if (n1 < n2)
     low = n1;
     else
     low = n2;
     if (n3 < low)
     low = n3;
     if (n4 < low)
     low = n4;
     return (low);
}
```

19. Write C++ program to print the input string in a reverse order using function, which first locates the end of string. Then it swaps the first character with the last character with the second last character and so on.

Ans. //C++ program to reverse the string

```
# include <iostream.h>
# include <stdio.h>
# include <string.h>
void reverse (char str[ ], int);
void main ( )
{
     char str [80];
     cout << “Enter the string”;
     gets (str);
     int len = strlen (str);
          reverse (str,len);
}
Void reverse (char str1[ ], int 1)
{
     int mid = 1/2;
     for (int I = 1; I < = mid; i ++)
     {
          char temp = str1 [i];
          str1 [i] = str1 [1];
          str1 [1] = temp;
          1– –;
     }
     cout << “Reverse of string is:”;
     puts (str1);
}
```

20. Write a C++ Program by using swap function to interchange given two numbers void swap (int & x, int & y);

Ans.

```
// C++ program for interchange values.
# include <iostream.h>
void swap (int & x, int & y);
void main ( )
{
     int a, b;
     cout << “Enter values for a and b”;
     cin >> a >> b;
     swap (a, b);
     cout << “After swapping” << end 1;
     cout << “a = “ << a;
     cout << “b = “ << b;
}
void swap (int & x, int & y)
{
     int temp = x;
     x = y;
     y = temp;
}
```

Exercise

Select the correct alternative and rewrite the following.

1. float *ptr:
In above declaration, data type of ptr is _____ and data type of variable pointed by ptr is _____
(i) float, float (ii) float, int (iii) int, float (iv) pointer, float

1. (iv) pointer, float

2. When a function is called by reference, it can work an _______ variables in the calling program.
(i) Original (ii) Virtual (iii) Copies of (iv) None of these

2. (i) Original

3. *ptr ++ means______
(i) Increment the content of ptr by size of data type to which ptr is pointer.
(ii) Increment the content of ptr by 1.
(iii) Increment the content of memory location pointed by ptr by 1.
(iv) None of these.

3. (iii) Increment the content of memory location pointed by ptr by 1.

4. Last character of a string is_______
(i) 0 (ii) \0 (iii) \n (iv) end

4. (ii) \0

5. When we use string functions such as strlen(), strew() etc. then it must include the ______
(i) #include<string.h> (ii) #include<iostrem.h>
(iii) #include<fstream.h> (iv) #include<iostring.h>

5. (i) #include<string.h>

6. Objects are basic ______ in object oriented programming.
(i) Run time entities (ii) Compile time entities
(iii) Data types (iv) None of these

6. (i) Run time entities

7. Object is a variable, whose data type is _______
(i) integer (ii) class (iii) structure (iv) float

7. (ii) class

8. _______ is not a visibility label.
(i) Public (ii) Private (iii) Separate (iv) Protected

8. (iii) Separate

9. The members declared under _______ visibility label are hidden from external use.
(i) Public (ii) Private (iii) Both (i) and (ii) (iv) None of (i) and (ii)

9. (ii) Private

10. If all visibility labels are missing, then by default members of class are ________

(i) Public (ii) Protected (iii) Private (iv) Any of these

10. (iii) Private

11. When a member function is defined inside the class, then it is treated as a ______ function.

(i) inline (ii) outline (iii) external (iv) virtual

11. (i) inline

12. A class is defined as follows,

```
class abc
{
private:
        int a;
public:
        friend void getdata (void);
};
```

The correct header for defining getdata () function outside the class is

(i) friend void abc::getdata (void)
(ii) void abc::getdata (void)
(iii) friend void getdata (void)
(iv) void getdata (void)

12. (iv) void getdata (void)

13. A constructor can never return a value. Hence it has ______

(i) no return type (ii) void return type (iii) int return type (iv) any of these

13. (i) no return type

14. A destructor is invoked implicitly by the complier upon _____ the program.

(i) entry in (ii) exit from (iii) Mid point of (iv) none of these

14. (ii) exit from

15. A special function, which is used to define an additional task to an operator is called as __________

(i) operator overloading (ii) operator function
(iii) friend function (iv) constructor

15. (ii) operator function

16. Operator function as a member function will have only one argument for _____ operators.

(i) unary (ii) binary (iii) sizeof (iv) none of these

16. (ii) binary

17. _________is the operator which cannot be overloaded.

(i) scope resolution (ii) binary
(iii) unary (iv) none of these

17. (i) scope resolution

18. The derivation of one class from another derived class is called as_______

(i) multiple inheritance (ii) single inheritance
(iii) multilevel inheritance (iv) hybrid inheritance

18. (iii) multilevel inheritance

19. When a class is made _____ the complier takes necessary care to see that only one copy of that class is inherited in derived classes.

(i) virtual base class (ii) base class
(iii) derived class (iv) single class

19. (i) virtual base class

20. In early binding, the correct function to be invoked is selected at ________

(i) runtime (ii) polymorphism time
(iii) compile time (iv) none of these

20. (iii) compile time

21. To achieve run–time polymorphism, C++ supports mechanism of _______

(i) function overloading (ii) operator overloading
(iii) virtual functions (iv) both (i) and (ii)

21. (iii) virtual functions

22. _________ stream extracts data from the file.

(i) input (ii) output (iii) input/output (iv) none of these

22. (i) input

23. The class _______ is not derived from ifstream base class.

(i) filebuf (ii) fstream (iii) ifstream (iv) ofstream

23. (i) filebuf

24. ios::in means_________

(i) open a file for writing only (ii) open a file for reading only
(iii) open fails if file already exists (iv) open a file in binary mode

24. (ii) open a file for reading only

25. C++ was developed by Bjarne stroustrup at________

(i) Xerox corporation (ii) At and T Bell laboratories
(iii) Polo Alu Research Centre (PARC) (iv) None of these

25. (ii) At and T Bell laboratories

26. ________ is not a built–in datatype in C++.

(i) void (ii) float (iii) char (iv) class

26. (iv) class

27. The do–while statement is _________

(i) an entry control loop (ii) conditional statement

(iii) an exit control loop (iv) none of these

27. (iii) an exit control loop

28. Function overloading is an example of _________

(i) Inheritance (ii) Run–time polymorphism

(iii) Compile time polymorphism (iv) All of these

28. (iii) Compile time polymorphism

29. The range of signed short int is ________

(i) –128 to 127 (ii) –32768 to 32767

(iii) 0 to 255 (iv) 0 to 65535

29. (ii) –32768 to 32767

30. If the value of a = 4 and b = 7, then the value of p after execution of the statement p = –– b + a ++ is _______

(i) 10 (ii) 11 (iii) 9 (iv) 12

30. (i) 10

31. To read data from a file, the file should be opened in ______ mode.

(i) input (ii) output (iii) append (iv) None of these

31. (i) input

32. The ability to take more than one form is called _______ in object–oriented programming.

(i) inheritance (ii) encapsulation (iii) polymorphism (iv) data abstraction

32. (iii) polymorphism

33. Which of the following is not a feature of object–oriented programming?

(i) Follows bottom–up approach in program design.

(ii) Objects may communicate with each other through functions.

(iii) Follows top–down approach in program design.

(iv) Programs are divided into what are known as objects.

33. (iii) Follows top–down approach in program design.

34. Programming in C++ using classes is called ______ programming.

(i) procedure oriented (ii) event drivern

(iii) object oriented (iv) database

34. (iii) object oriented

35. ________ is not a derived data type in C++.

(i) Class (ii) Array (iii) Function (iv) Pointer

35. (i) Class

36. _________ is not the feature of OOPs.

(i) Polymorphism (ii) Inheritance

(iii) Data Abstraction (iv) Top–down Approach

36. (iv) Top–down Approach

37. A derived class with several base classes is _______ inheritance.

(i) single (ii) multiple (iii) multilevel (iv) hierarchical

37. (ii) multiple

38. Which of the following allows to access the private data of other class is ______

(i) In–line function (ii) Friend function

(iii) Main function (iv) All of the above

38. (ii) Friend function

39. While accessing the number in a float array using pointer, the pointers value every time increases by _________.

(i) 2 (ii) 4 (iii) 8 (iv) 16

39. (ii) 4

40. Following operator cannot be overloaded.

(i) ++ (ii) :: (iii) – (iv) *

40. (ii) ::

41. ________ type of member function of class never takes my argument.

(i) Constructor (ii) Destructor (iii) Operator (iv) None of these

41. (ii) Destructor

42. For a 23 and b = 3, the value of c after execution of the statement c = (a/b) * (a%b) will be______

(i) 14 (ii) 49 (iii) 21 (iv) 69

42. (i) 14

43. Out of the following C++ operators, ________ operators can be overloaded.

(i) Sizeof (ii) :: (iii) ?: (iv) *

43. (iv) *

44. A pointer is a variable that holds ______ of another variable.

(i) Value (ii) Memory Address

(iii) Data–type (iv) None of these

44. (ii) Memory Address

45. Object Oriented Programming follows __________ approach in program design.

(i) top–down (ii) Non–hieranchical

(iii) random (iv) Bottom–up

45. (iv) Bottom–up

46. When a base class is privately inherited by a derived class, public member of the class becomes ________ of the derived class.

(i) Public Member (ii) Protected Member

(iii) Private Member (iv) Non–Member

46. (iii) Private Member

47. In C^{++} ________ is an extraction operator.

(i) << (ii) >> (iii) && (iv) !

47. (ii) >>

48. Object oriented programming uses __________ approach of programming.

(i) Linear (ii) Non–linear (iii) Top down (iv) Bottom up

48. (iv) Bottom up

49. We can not define more then one ________ in a class.

(i) Constructor (ii) Destructor

(iii) Member Function (iv) Data Member

49. (ii) Destructor

50. In C++, double type data consumes ______ bytes in memory.

(i) 2 (ii) 4 (iii) 6 (iv) 8

50. (iv) 8

51. _____ symbol is used to declare destructor function in C++.

(i) @ (ii) # (iii) ~ (iv) !

51. (iii) ~

52. What will be the value of x after execution of following expression in C++?

X = + + m + n + + ; where m = 10 and n = 15.

(i) 25 (ii) 27 (iii) 26 (iv) 28

52. (iii) 26

53. ________ operator can be overloaded.

(i) +(Plus) (ii) | | (Logical OR)

(iii) %(Modulus) (iv) All i, ii and iii

53. (iv) All i, ii and iii

54. ________ operator cannot be overloaded.

(i) ++ (ii) + (iii) : : (iv) >>

54. (iii) : :

55. If any access specifier is not specified for member function or data number in class then by default it is ________.

(i) public (ii) private (iii) protected (v) void

55. (ii) private

❑ ❑ ❑ ❑ ❑ ❑