Enable | Enlighten | Enrich
(Deemed to be University)
(Under Section 3 of UGC Act 1956)

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT : **OBJECT ORIENTED PROGRAMMING WITH C++**

SEMESTER : I

SUBJECT CODE: **17CCP104** **CLASS** : **I M.COM CA**

FIRST INTERNAL TEST ANSWER KEY

PART – A (20 * 1 = 20)
Multiple choice Questions

1. OOP language
2. C language
3. Tokens
4. do-while
5. for
6. console input and output
7. object oriented programming
8. objects
9. ;
10. encapsulation
11. extraction operator
12. Visibility
13. array
14. Operator Overloading
15. Two
16. Bjarne stroustrup
17. Inheritance
18. constants
19. Abstraction
20. ::

PART – B (3 * 2 = 6)
Answer All the Questions

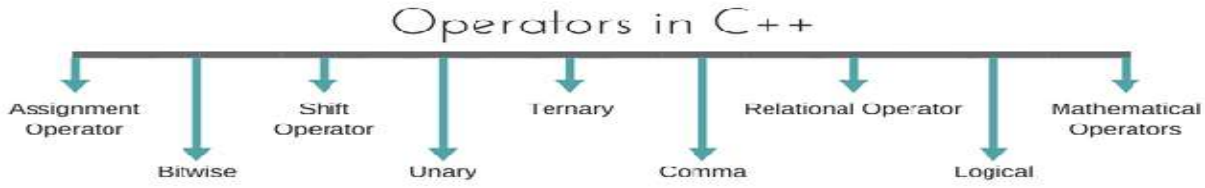
21. Note down the Structure of C++ Program

Structure of C++ program

- Include section
- Class declaration
- Member function definition
- Main function program

22. Define Operator

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. **C++** is rich in built-in operators and provide the following types of operators.



22. How will you specify a Class.?

- Class is composed of three things: a name, attributes, and operations.
- Class is a way to bind the data and its associated functions together

Class specification has 2 parts:

- Class Declaration.
- Class function definitions

Class Declaration:

Syntax:

```
class class_name
{
    private:
        variable declaration;
        function declaration;
    public:
        variable declaration;
        function declaration;
};
```

Example:

```
class book
{
   int pgno;
   public:
      void getpage();
 };:
```

PART – C (3 * 8 = 24)

Answer All the Questions

24.a) Describe the Basic Concepts of OOPs

Basic Concepts of OOPs

- Objects
- Classes
- Data abstraction and Encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

Objects:

- An object can be considered a "thing" that can perform a set of related activities.
- The set of activities that the object performs defines the object's behavior.
- For example, the hand can grip something or a Student (object) can give the name or address.
- Objects are run time entity or real world entity.

Classes:

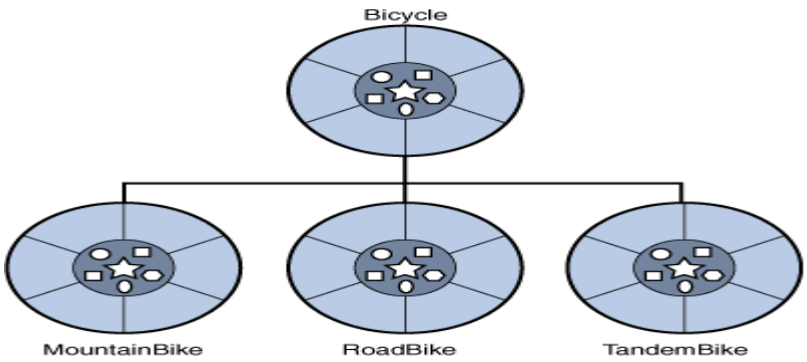
- A class is simply a representation of a type of object.
- It is the blueprint/ plan/ template that describe the details of an object.
- A class is the blueprint from which the individual objects are created.
- Class is composed of three things: a name, attributes, and operations.
- For example Student is an object has name, age, course, etc as attributes. Read, write, etc as operations

Data abstraction and Encapsulation

- The encapsulation is the inclusion within a program object of all the resources need for the object to function - basically, the methods and the data.
- In OOP the encapsulation is mainly achieved by creating classes, the classes expose public methods and properties.
- The class is kind of a container or capsule or a cell, which encapsulate the set of methods, attribute and properties to provide its indented functionalities to other classes.
- In that sense, encapsulation also allows a class to change its internal implementation without hurting the overall functioning of the system.
- That idea of encapsulation is to hide how a class does it but to allow requesting what to do.
- Abstraction is an emphasis on the idea, qualities and properties rather than the particulars.
- The importance of abstraction is derived from its ability to hide irrelevant details and from the use of names to reference objects.
- Abstraction is essential in the construction of programs.
- It places the emphasis on what an object is or does rather than how it is represented or how it works.

Inheritance

- Ability of a new class to be created, from an existing class by extending it, is called inheritance.
- Different kinds of objects often have a certain amount in common with each other.
- Object-oriented programming allows classes to inherit commonly used state and behavior from other classes.
- In this example, Bicycle now becomes the super class of MountainBike, RoadBike, and TandemBike. In the Java programming language, each class is allowed to have one direct super class, and each super class has the potential for an unlimited number of subclasses:



- The new class created is called as derived class
- The existing class is called as base class.
- The base class provides the property the derived class receives the property.
- It reduces the complexity of the programming.
- This is the most common and most natural and widely accepted way of implement this relationship.

Polymorphism

- Polymorphism is the process taking more then one form.
- More precisely Polymorphisms mean the ability to request that the same operations be performed by a wide range of different types of things.
- In OOP the polymorphisms is achieved by using many different techniques named method overloading, operator overloading and method overriding,
- The method overloading is the ability to define several methods all with the same name.
- The operator overloading (less commonly known as ad-hoc polymorphisms) is a specific case of polymorphisms in which some or all of operators like +, - or == are treated as polymorphic functions and as such have different behaviors depending on the types of its arguments.

Dynamic binding

- Dynamic binding is the process of resolving the function to be associated with the respective functions calls during their runtime rather than compile time.

Message passing

- Every data in an object in oops that is capable of processing request known as message.
- All objects can communicate with each other by sending message to each other
- Message passing, also known as interfacing, describes the communication between objects using their public interfaces.

24.b) **List out**

i) **Application of Oops**

- For Develop Graphical related application like computer and mobile games.
- To evaluate any kind of mathematical equation use C++ language.
- C++ Language are also used for design OS. Like window xp.
- Google also use C++ for Indexing
- Few parts of apple OS X are written in C++ programming language.

- Internet browser Firefox are written in C++ programming language
- All major applications of adobe systems are developed in C++ programming language. Like Photoshop, ImageReady, Illustrator and Adobe Premier.
- Some of the Google applications are also written in C++, including Google file system and Google Chromium.
- C++ are used for design database like MySQL.

ii) **C++ Statements**

- Preprocessor directives

  A preprocessor directive begins with the character #. This must either be the first character of the line or the first character of the line after some leading whitespace

- Comments

  Comments may be of the form:

  / / *Comment* \n

  (Or)

  /* comment */

  - The first form allows a trailing comment on a single line, while the second form allows comments that span multiple lines.
  - Comments may appear anywhere.

- Declarations

  Declarations give the compiler information about the types, storage requirements and initial values of identifiers.

  - **General form:** void type identifier intializer;

- Function Declarations

```
void type identifier (formal_argument_list  )
{
function_body
}
Executable statements
while ( expression ) statement
do statement while ( expression )
```

```
for ( expression_1; expression_2; expression_3 ) statement
equivalent to:
expression_1; while ( expression_2 ) { statement expression_3 }
switch ( expression )
{
declarations
case constant_expression: statement
...
default: statement
}
break;
continue;
return;
return expression;
goto statement_label;
 statement_label: executable_statement
```

25.a) Illustrate: Control Structures.

Control Structure:

- Sequence structure
- Selection structure
- Loop structure

Selection Structure:

- if statement
- if…else
- Nested if
- if… else ladder
- switch statement

if Statement:

if statement takes condition in parenthesis and a block of statements within braces. If condition is true, it will return non-zero value, and statements given in if block will get execute.

Syntax:

```
if(condition)
{
        True Block;
}
```

Next Statement;

Example:

```
if(i>10)
   cout<<"i greater than 10";
```

if ...else Statement

if statement takes condition in parenthesis and a block of statements within braces. If condition is true, it will return non-zero value, and statements given in if block will get execute. If condition is false, it will returns zero, and statements given in else block will get execute.

Syntax:

```
if(condition)
{
        True Block;
}
else
{
        False Block;
}
Next Statement;
```

Example:

```
if(i>10)
   cout<<"i greater than 10";
else
   cout<<"i less than 10";
```

Nested if Statement:

In nested if-else, one if-else statement contains another if-else statement.

Syntax:

```
if(condition 1)
{
        if(condition 2)
        {
                True block
        }
        else
        {
                False block condition 2;
        }
}
else
  {
        False block condition 1;
```

```
        }
    Next Statement;
```

Example:

```
if(m1>40)
{
        if(m2>40)
        {
                cout<<"pass";
        }
        else
        {
                cout<<"Fail";
        }
}
else
        {
                cout<<"Fail";
        }
```

if... else ladder:

if-else ladder is used for checking multiple conditions, if the first condition will not satisfy, compiler will jump to else block and check the next condition, whether it is true or not and so on.

Syntax:

```
    if(condition 1)
    {
             True block-1
    }
    else if(condition 2)
   {
        True block -2;
   }
   else
   {
        False block;
    }
     Next Statement;
```

Example:

```
if(a>b)
{
        if(a>c)
        {
                cout<<"A is Greatest";
        }
        else
        {
                cout<<"C is Greatest";
```

```
                }
}
else
if(b>c)
{
                cout<<"B is Greatest";
}
else
{
                cout<<"C is Greatest";
}
```

Switch Case:

Syntax:

```
                switch(expression)
                {
                        case exp1:
                                Statements
                        case exp2:
                                Statements
                        ……..
                        ……..
                        default:
                                Statements
                }
```

Example

```
i=4;
switch(i)
{
                case 1:
                        cout<<"one";
                case 2:
                        cout<<"two";
                case 3:
                        cout<<"three";
                default:
                        cout<<"Wrong Choice";
}
```

do-while – while statement, for statement

Loop Structure

Entry control:

Entry control Structure checks the condition First and the Statement is executed

- while loop

- for loop

Exit control:

Exit control Structure First the Statement is Executed and then checks the condition

- do… while

While Loop:

While loop is also called entry control loop because, in while loop, compiler will 1st check the condition, whether it is true or false, if condition is true then execute the statements.

Syntax:

```
while(Condition)
{
        Statement Block
}
```

Example:

```
while(i<5)
{
        cout<<"Welcome";
        i++;
}
```

For Loop:

In for loop has initialization, condition and increment/decrement all together. Initialization will be done once at the beginning of loop. Then, the condition is checked by the compiler. If the condition is false, for loop is terminated. But, if condition is true then, the statements are executed until condition is false.

Syntax:

```
for(initialization ; condition checking ; Increment/Decrement)
{
        Statement Block
}
```

Example:

```
for(i=1;i<5;i++)
{
        cout<<"Welcome";
```

```
}
```

Do..While

The do-while loop is also called exit control loop because, in do-while loop, compiler will 1st execute the statements, then check the condition, whether it is true or false.

Syntax:

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

Example:

```
do
{
        cout<<"Welcome";
        i++;
}while(i<5);
```

Difference b/w while loop and do-while loop

while loop	**do-while loop**
It is entry control loop.	It is exit control loop.
In this loop condition is checked before loop execution.	In this loop condition is checked at the end of loop.
It will never execute loop if condition is false.	It will executes loop at least once when the initial condition is false.
There is no semicolon at the end of while statement	There is semicolon at the end of while statement.

Jump Statements in C++

Jump statements are used to interrupt the normal flow of program.

Types of Jump Statements

- Break
- Continue
- GoTo

Break Statement

The break statement is used inside loop or switch statement. When compiler finds the break statement inside a loop, compiler will abort the loop and continue to execute statements followed by loop.

Example of break statement

```
#include<iostream.h>

void main()
{
    int a=1;

    while(a<=10)
    {
        if(a==5)
          break;

        cout << "\nStatement " << a;
        a++;
    }
        cout << "\nEnd of Program.";
}
```

Output :

```
Statement 1
Statement 2
Statement 3
Statement 4
End of Program.
```

Continue Statement

The continue statement is also used inside loop. When compiler finds the break statement inside a loop, compiler will skip all the followling statements in the loop and resume the loop.

Example of continue statement

```
#include<iostream.h>

void main()
```

```
{
        int a=0;

        while(a<10)
         {

            a++;

            if(a==5)
              continue;

             cout << "\nStatement " << a;

         }
             cout << "\nEnd of Program.";
}
```

Output :

```
Statement 1
Statement 2
Statemnet 3
Statement 4
Statement 6
Statement 7
Statement 8
Statement 9
Statement 10
End of Program.
```

Goto Statement

The goto statement is a jump statement which jumps from one point to another point within a function.

Syntax of goto statement

```
goto label;
    - - - - - - - - - -
    - - - - - - - - - -
label:
    - - - - - - - - - -
    - - - - - - - - - -
```

In the above syntax, label is an identifier. When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code after it.

Example of goto statement

```
#include<iostream.h>

void main()
{
        cout << "\nStatement 1.";
        cout << "\nStatement 2.";
        cout << "\nStatement 3.";

        goto last;

        cout << "\nStatement 4.";
        cout << "\nStatement 5.";

        last:

        cout << "\nEnd of Program.";
}
```

Output :

```
Statement 1.
Statement 2.
Statement 3.
End of Program.
```

25. b)Explain : Functions in C++

A function is a block of codes that performs a specific task and may return value. The main() function is the first user defined function invoked by the compiler. While it is possible to write any code within main function, it leads number of problems. The program may become too large and complex and it is difficult to test, debugg and maintain the complex code. For that reason, We use function to place independent code in separate modules called function or subprogram. In order to make a program using function, we need to perform the followling three steps.

- Function declaration
- Function definition
- Function call

Function declaration

Like variables, all the functions must be declared. Function declaration statement includes function name, what function will take and what function will return.

Syntax : return-type function-name(argument list);

Where, return-type : type of value function will return.
function-name : any valid C++ identifier.
argument list : represents the type and number of value function will take, values are sent by the calling statement.

Example: int Add(int, int);

Function definition

Function definition includes the actual working or implementation.

Syntax:

```
return-type function-name(argument list)
        {
                - - - - - - - - - -
                body of function
                - - - - - - - - - -
        }
```

Example

```
int Add(int x,int y)
        {
            int sum;

             sum = x + y;

            return sum;
        }
```

There are two approaches to passing argument to a function:

- Call by Value
- Call by Reference/Address

Call by Value

In this approach, the values are passed as function argument to the definition of function.

```
#include<iostream.h>

    void main()
    {
        int A=10,B=20;

        cout << "\nValues before calling";
        cout << "\nA : " << A;
        cout << "\nB : " << B;

        fun(A,B);                    //Statement    1

        cout << "\nValues after calling";
        cout << "\nA : " << A;
        cout << "\nB : " << B;


    }

    void fun(int X,int Y)            //Statement    2
    {
        X=11;
        Y=22;
    }
```

Output :

```
        Values before calling
        A : 10
        B : 20
        Values after  calling
        A : 10
        B : 20
```

Call by Reference

In this approach, the references/addresses are passed as function argument to the definition of function.

```
#include<iostream.h>

    void main()
    {
        int A=10,B=20;

        cout << "\nValues before calling";
        cout << "\nA : " << A;
```

```
        cout << "\nB : " << B;

        fun(&A,&B);                    //Statement    1

        cout << "\nValues after calling";
        cout << "\nA : " << A;
        cout << "\nB : " << B;


    }

    void fun(int *X,int *Y)            //Statement    2
    {
        *X=11;
        *Y=22;
    }
```

Output :

```
        Values before calling
        A : 10
        B : 20
        Values after  calling
        A : 11
        B : 22
```

Inline functions

This function is to save memory space by making common block for the code we need to execute many times.

Syntax:

```
inline function
{
    function body
}
```

Example of inline function

```
#include<iostream.h>
#include<conio.h>
inline  int add(int a,int b)
{
  return(a+b);
}
void main()
{
   int m1,m2;
   clrscr();
   cout<<"Enter the first number:";
```

```
    cin>>m1;
    cout<<"Enter the Second number:";
    cin>>m2;
    cout<<"Addition Result:"<<add(m1,m2)<<endl;
}
```

26. a)How will you define member functions

Defining a Member functions

- Definition in 2 places
- Outside the class definition.
- Inside the class definition.

Outside the Class Definition

Syntax:

```
return_type class_name :: function_name(argument list)
{
        - - - - - - - - - -
        body of function
        - - - - - - - - - -
}
```

return_type Type of value function will return.

class_name:: A program may contain more than one class and these classes may have similar member functions. Class_name:: tells the compiler which class the function belongs to and the scope of the member function is restricted to the class_name.

function_name Can be any valid C++ identifier.

argument list Represents the type and number of value function will take, values are sent by the calling statement.

Example of defining member function outside class

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

class Employee
{
```

```
        int Id;
        char Name[25];
        int Age;
        long Salary;

        public:
        void GetData();
        void PutData();

};

void Employee :: GetData()          //Statement 1 : Defining GetData()
{
        cout<<"\n\tEnter Employee Id : ";
        cin>>Id;

        cout<<"\n\tEnter Employee Name : ";
        cin>>Name;

        cout<<"\n\tEnter Employee Age : ";
        cin>>Age;

        cout<<"\n\tEnter Employee Salary : ";
        cin>>Salary;
}

void Employee :: PutData()          //Statement 2 : Defining PutData()
{
        cout<<"\n\nEmployee Id : "<<Id;
        cout<<"\nEmployee Name : "<<Name;
        cout<<"\nEmployee Age : "<<Age;
        cout<<"\nEmployee Salary : "<<Salary;
}

void main()
{

        Employee E;          //Statement 3 : Creating Object

        E.GetData();         //Statement 4 : Calling GetData()
        E.PutData();         //Statement 5 : Calling PutData()

}
```

```
Output :

        Enter Employee Id : 1
        Enter Employee Name : Kumar
        Enter Employee Age : 29
        Enter Employee Salary : 45000

        Employee Id : 1
        Employee Name : Kumar
        Employee Age : 29
        Employee Salary : 45000
```

Inside the Class Definition:

Syntax:

```
return_type function_name(argument list)
{
        - - - - - - - - - -
        body of function
        - - - - - - - - - -
}
```

return_type Type of value function will return.

function_name Can be any valid C++ identifier.

argument list Represents the type and number of value function will take, values are sent by the calling statement.

Definition of member function inside class is similar to defining normal function. There is no need to tell compiler about the class the function belongs to because the definition of member function is already in the class.

Example of defining member function inside class

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

class Employee
{
    int Id;
    char Name[25];
    int Age;
```

```
        long Salary;

        public:
        void GetData()          //Statement 1 : Defining GetData()
        {
            cout<<"\n\tEnter Employee Id : ";
            cin>>Id;

            cout<<"\n\tEnter Employee Name : ";
            cin>>Name;

            cout<<"\n\tEnter Employee Age : ";
            cin>>Age;

            cout<<"\n\tEnter Employee Salary : ";
            cin>>Salary;
        }

        void PutData()          //Statement 2 : Defining PutData()
        {
            cout<<"\n\nEmployee Id : "<<Id;
            cout<<"\nEmployee Name : "<<Name;
            cout<<"\nEmployee Age : "<<Age;
            cout<<"\nEmployee Salary : "<<Salary;
        }

    };
    void main()
    {

        Employee E;          //Statement 3 : Creating Object

        E.GetData();         //Statement 4 : Calling GetData()
        E.PutData();         //Statement 5 : Calling PutData()

    }
```

Output :

```
        Enter Employee Id : 1
        Enter Employee Name : Kumar
        Enter Employee Age : 29
        Enter Employee Salary : 45000
```

```
Employee Id : 1
Employee Name : Kumar
Employee Age : 29
```

Employee Salary : 45000

26.b) Write a program to print the book details. (member functions should be defined outside the class)

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

class Book
{
      Int BkId;
      char bkname[25];
      char author;

      public:
      void GetData();
      void PutData();

};

void Book :: GetData()          //Statement 1 : Defining GetData()
{
      cout<<"\n\tEnter Book Id : ";
      cin>>BkId;

      cout<<"\n\tEnter Book Name : ";
      cin>>bkname;

      cout<<"\n\tEnter Author Name : ";
      cin>>author;

}

void Book :: PutData()          //Statement 2 : Defining PutData()
{
      cout<<"\n\nBookId : "<<BkId;
      cout<<"\nBook Name : "<<bkname;
      cout<<"\nAuthor name : "<<author;

}
```

```
void main()
{

      Book B:          //Statement 3 : Creating Object

       B.GetData();          //Statement 4 : Calling GetData()
      B.PutData();          //Statement 5 : Calling PutData()

}
```

Output :

```
       Enter Book Id: 110001
       Enter Book Name: Object Oriented Programming with c++
       Enter Author Name: E.Balagursamy

     Book Id: 110001
     Book Name: Object Oriented Programming with c++
     Author Name: E.Balagursamy
```

Enable | Enlighten | Enrich
(Deemed to be University)
(Under Section 3 of UGC Act 1956)

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT : **OBJECT ORIENTED PROGRAMMING WITH C++**

SEMESTER : I

SUBJECT CODE: **17CCP104** **CLASS** : **I M.COM CA**

SECOND INTERNAL TEST ANSWER KEY

PART – A (20 * 1 = 20)
Multiple choice Questions

1. Single of
2. Operator overloading
3. Inheritance
4. Inheritance
5. Protected
6. Internal
7. Function Overloading
8. Base
9. Two colons
10. Overloading
11. Inheritance
12. Hierarchical
13. Global
14. The null character
15. Abstract class
16. the declarations of the basic standard input-output library
17. the insertion operator is overloaded in ofstream
18. checking for end of file
19. Console.h
20. seek to end of file before each write

PART – B (3 * 2 = 6)
Answer All the Questions

21. Define Constructor

Constructor is a special function used to initialize class data members or we can say constructor is used to initialize the object of class.

- Constructor name class name must be same.
- Constructor doesn't return value.
- Constructor is invoked automatically, when the object of class is created.

22. Define Operator Overloading

The process of giving special meaning to a method or an operator is called Operator Overloading,. Overloading is the process of adding an extra or additional operation to an existing operation. Overloading consist of same name but differ in their argument list, Number of argument or both.

There are two types of overloading

- ❖ Function overloading
- ❖ Operator overloading

23. Define Pointer

A pointer is a variable that holds a memory address. This address is the location of another object (typically, a variable) in memory. That is, if one variable contains the address of another variable, the first variable is said to point to the second.

- A pointer declaration consists of a base type, an *, and the variable name.
- The general form of declaring a pointer variable is :

```
type *name;
```

where ,

-The 'type' is the base type of the pointer and may be any valid type.

- The 'name' is the name of pointer variable.

- The base type of the pointer defines what type of variables the pointer can point to.

PART – C (3 * 8 = 24)

Answer All the Questions

24. a) Elucidate the rules for overloading operator

Rules for Overloading Operators:

1. Only existing operators can be overloaded. New operators cannot be created
2. The overloaded operator must have at least one operand that is of user-defined type

3. Can not be able to change the predefined meaning of the Operator.
4. An overloaded operator follows the syntax rules of the original operators. They can not be overridden
5. Some Operators that can not be overloaded.
6. Certain Operators can not be overloaded using the friend Function.

Operators cannot be overloaded

- Membership operators (.)
- Pointer-to-member operator (.*)
- Scope resolution operator (::)
- Size of operator (sizeof)
- Conditional operator (?:)

Operators Cannot be Overloaded Using friend Function

- Assignment operator (=)
- Function call operator (())
- Subscripting operator ([])
- Class member access operator (->)

Example

```
#include<iostream.h>
class Add
{
  int lat,log;
  public:
    Add(){}
    Add(int l,int lt)
    {
       lat=l;
       log=lt;
    }
    void show()
    {
      cout<<lat<<" ";
      cout<<log<<" ";
    }
    Add operator -(Add o);
};
Add Add::operator -(Add o)
{
  Add t;
```

```
  t.lat=o.lat+lat;
  t.log=o.log+log;
  return t;
}

void main()
{
  Add a(10,20),b(30,50);
  a.show();
  b.show();
  a=a-b;
  a.show();
}
```

24.b) Explain in detail about type conversions.

Type Conversions

- Type Conversion is the process of change the data type of variable.
- When constants and variables of different types are mixed in an expression, C applies automatic type conversion to the operand as per certain rules.
- In the Assignment Operation the type of data to the right of an assignment operator is automatically converted to the type of the variable on the left.

Example:

```
int m;
float x=5.89;
m=x;
```

The value of m is 5 since the fraction part is truncated.

- The Compiler does not support automatic type conversion for user defined data types. Since the type conversion should be performed explicitly.
- Three types of situations arises in the data conversion between incompatible types:

- Conversion from basic type to class type
- Conversion from class type to basic type
- Conversion from one class type to another class type.

Conversion from basic type to class type

- In this left hand operand of = operator is always a class object.

Example:

```
String s1,s2;
char *name1=”C++ Programming”;
char *name2=”C Programming”;
s1=String(name1);
s2=name2;
```

String is the class, name1 is char variable which is converted explicitly in the statement

```
s1=String(name1);
```

name2 is char variable which is converted implicitly by call the constructor in the statement

```
s2=name2;
```

The constructor used for type conversion takes a single argument whose type is to be converted. This conversion is done by overloaded = operator.

Conversion from class type to basic type

- In this left hand operand of = operator is always a variable type or basic type.
- The constructor performs a fine job in conversion from basic to class type.
- The conversion of class to basic model is done by overloaded casting operator.
- The general form of an overloaded casting operator function is

```
operator typename()
{
    ..........
    ..........//function statement
}
```

Example:

```
String :: operator double()
```

```
{
        double d=0;
        d=s[0]+s[1];
        return(d);
}
```

String is a class converted to basic type double, where s is a String class object

```
String s1,s2;

float c1,c2;

c1=float(s1);

c2=s2;
```

c1 is float variable , String is the class which is converted explicitly in the statement

```
c1=float(s1);
```

c2 is float variable , String is the class which is converted implicitly in the statement

```
c2=s2;
```

The casting operator function should satisfy the following conditions:

- It must be a class member
- It must not specify a return type
- It must not have any arguments.

Conversion from one class type to another class type.

Example:

```
S1= S2   // objects of different types
```

- S1 and S2 are the object of two different classes class X and class Y.
- The class Y type is converted into X type.
- Y is known as the source class and X is known as the designation class.
- This type of conversion is performed using either a contractor or a conversion function

- Casting function is of the form

  operator typename()

Type Conversion Table

Conversions Required	Conversion takes place in	
	Source class	Designation class
Basic →class	Not applicable	Constructor
Class→Basic	Casting operator	Not applicable
Class→class	Casting operator	Constructor

25.a) Elaborate manipulation of string using operator

C++ allows us the facility of manipulate strings using the concept of operator overloading.

Example: we can overload + operator to concate two strings. We can overload == operator to compare two strings. The Following Example in which we overload + operator to concate two strings

```
#include<string.h>
#include<iostream.h>
#include<conio.h>
class string
{
public:
char strl[30];
string()
{
strcpy(strl,"");
}
```

```
void getstring();
void putstring();
int stlen();
int operator>(char*mm)
{
if(strcmp(strl,mm)>0)
return(1);
else
return(0);
}
int operator==(char*mm)
{
if(strcmp(strl,mm)==0)
return(1);
else
return(0);
}
char*operator+(char*sc)
{
return strcat(strl,sc);
}
};
void string::getstring()
{
cout<<endl<<"\n enter your string<type end to stop>:";
cin>>strl;
}
void string::putstring()
{
cout<<"your string is:"<<strl<<endl;
}
```

```
int string::stlen()
{
int len;
len=strlen(strl);
return(len);
}
void main()
{
string stl;
char mystr[30];
char*strcnt;
clrscr();
while(1)
{
stl.getstring();
if(stl=="end"||stl=="END"||stl=="END")
break;
cout<<endl<<"enter second string:";
cin>>mystr;
if(stl==mystr)
cout<<endl<<stl.strl<<">"<<mystr<<endl<<endl;
else
if(stl>mystr)
cout<<endl<<stl.strl<<">"<<mystr<<endl<<endl;
else
cout<<endl<<stl.strl<<">"<<mystr<<endl<<endl;
strcnt=new char[30];
strcnt=stl+mystr;
cout<<"concatenated string is:"<<strcnt;
cout<<"\n length of a string is :"<<stl.stlen();
}
```

}

Output

Enter your string:hai

Enter second string:hello

Concatenated string is : haihello

25.b) Write a program for opening and closing a file with example

File Operations:

- Open file
- Read and Write Operations
- Closing a file

Opening and closing a file

- Use a disk file requires
- Suitable name for the file.
- Data type and structure.
- Purpose
- Opening method

Example

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<fstream.h>
void main()
{
ofstream out("student");
char name[25];
int regno;
long int m1,m2,m3,m4,m5,total,m;
float avg;
char result[10];
clrscr();
cout<<"\n\t student file:";
cout<<"\n enter the name of the student:";
cin>>name;
cout<<"\n enter the regno:";
cin>>regno;
cout<<"\n enter the tamil mark:";
```

```
cin>>m1;
cout<<"\n enter the cf mark:";
cin>>m2;
cout<<"\n enter the me mark:";
cin>>m3;
cout<<"\n enter the OR mark:";
cin>>m4;
cout<<"\n enter ther english mark:";
cin>>m5;
total=(m1+m2+m3+m4+m5);
avg=(total/5);
if((m1>40)&&(m2>40)&&(m3>40)&&(m4>40)&&(m5>40))
{
strcpy(result,"pass");
}
else
{
strcpy(result,"fail");
}
cout<<name<<"\n"<<regno<<"\n"<<m1<<"\n"<<m2<<"\n"<<m3<<"\n"<<m4<<"\n"<<m5<<"\
n";
ifstream in("student");
clrscr();
cout<<"\n\t mark statement:";
cout<<"\n\t **** *********:";
if(in.eof()==0)
{
cout<<"\n name:"<<name;
cout<<"\n regno:"<<regno;
cout<<"\n mark1:"<<m1;
cout<<"\n mark2:"<<m2;
cout<<"\n mark3:"<<m3;
cout<<"\n mark4:"<<m4;
cout<<"\n mark5:"<<m5;
cout<<"\n total:"<<total;
cout<<"\n average:"<<avg;
cout<<"\n _____:";
cout<<"\n result:"<<result;
cout<<"\n #######:";
}
in.close();
getch();
}
```

Output

```
name : keerthana
```

regno : 3
mark 1 :90
mark 2 :95
mark 3 : 70
mark 4 : 85
mark 5 :75
total :423
average : 84

result: pass

26.a) Briefly explain about destructor

Destructor

Constructor allocates the memory for an object. Destructor deallocate the memory occupied by an object. Like constructor, destructor name and class name must be same, preceded by a tilde(~) sign. Destructor take no argument and have no return value.

Constructor is invoked automatically when the object created. Destructor is invoked when the object goes out of scope. In other words, Destructor is invoked, when compiler comes out form the function where an object is created.

- Destructor destroys the objects that have been created by a constructor
- It is also a special member function
- Its also same name as the class name preceded by a tilde

Syntax:

~class-name();

- It never takes any arguments and have no return value
- Automatically invoked by compiler at the end of program
- delete is used for free memory

Program:

```
#include<iostream.h>
int count=0;
class copy
{
```

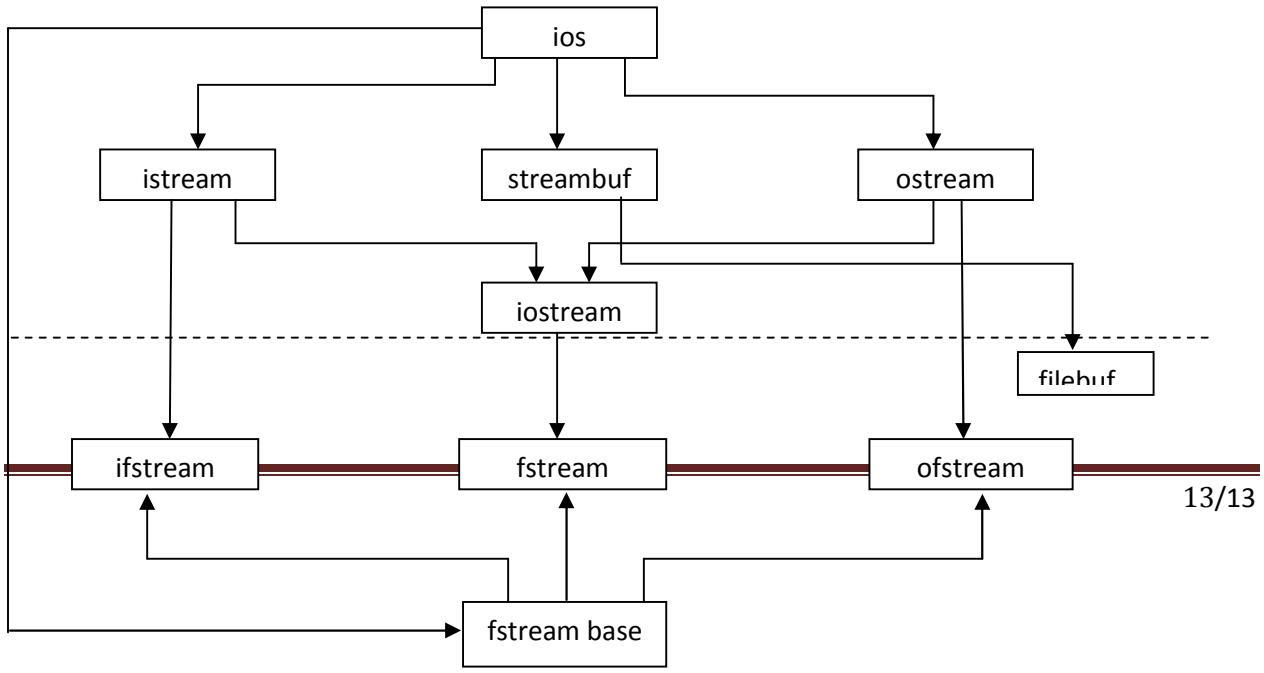
```
  int y;
  public:
     copy()
     {
                    count++;
                    cout<<"\nNo Objects Created:"<<count;
      }
       ~copy()
      {             cout<<"\nNo Objects Deleted:"<<count;
                    count--;
       }
};
void main()
{
 copy a,b,c,d;
}
```

26.b) Explain how the files can be works for stream operations with example

Classes for File Stream Operations:

- The C++ I/O system contains a set of classes that define the file handling methods.
- File handling class includes ifstream, ofstream, and fstream. These classes are derived from the corresponding iostream class.
- These are the class designed to manage the disk files
- All the classes are declared in fstream so all the program should include this header file

- **ofstream:** Stream class to write on files
- **ifstream:** Stream class to read from files
- **fstream:** Stream class to both read and write from/to files.
- **filebuf**:Used to manage the buffered I/O of the file stream. It contains Open() and Close() functions
- **Fstreambase**: The fstreambase class serves as a base class for ifstream.ofstream and fstream classes.



Reg No.....

[17CCP104]

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Established Under Section 3 of UGC Act 1956)
Coimbatore – 641021
(For the candidates admitted from 2017 onwards)
I M.Com (CA)
Second Internal Test, October - 2017
OBJECT ORIENTED PROGRMMING WITH C++

Time: 2 hours Maximum: 50 marks
Date & sess:

PART – A (20 * 1 = 20)
Multiple choice Questions

1. The operator that cannot be overloaded is _____
   a) Single of b) + c) - d) =
2. _____ is called compile time polymorphism.
   a) Operator overloading b) Function overloading
   c) Overloading unary operator d) Overloading binary operator
3. The mechanism of deriving a new class from an old one is called _____
   a) Operator overloading b) Inheritance c) Polymorphism d) Access mechanism
4. C++ can be reused using _____
   a) Inheritance b) Encapsulation c) Polymorphism d) Overloading
5. A member declared as _____ cannot be accessed by the function outside the class.
   a) Private b) Protected c) Public d) Visibility
6. Which of the following language feature is not an access specifier in C++?
   a)Public b)Private c)Protected d)Internal
7. _____ is the process of using the same name for two or more functions
   a) Function Overloading b) Operator Overloading
   c) Default Function d) Constructors
8. A class that is inherited is called as _____ class
   a) Derived b) Child c) Base d) Abstract
9. The scope resolution operator is
   a) comma b) semicolon c) colon d) Two colons
10. The feature that allows you to use the same function name for separate functions that have different argument lists is called _____
    a)Overriding b) Overloading c) Constructing d) destructing
11. Reusability of the code can be achieved in CPP through _____
    a)Encapsulation b) Inheritance c) Polymorphism d) Message Passing
12. Class X, class Y and class Z are derived from class BASE. This is _____ inheritance
    a)Multiple b) Multilevel c) Hierarchical d) Single
13. _____ variables remain in memory until the program ends
    a)Area b) Global c) Local d) Reference
14. The end of string is recognized by
    a)The null character b) The newline character c) The $ sign d) The / sign
15. If a class contains pure virtual function, then it is termed as_____.
    a)Abstract class b) Virtual class c) Pure Local class d) Sealed class
16. The file iostream includes_____
    a)the declarations of the basic standard input-output library
    b)the streams of includes and outputs of program effect
    c) mathematical functions are included d) string functions are included
17. We can output text to an object of class ofstream using the insertion operator Â« because
    a)the ofstream class is a stream b) the insertion operator works with all classes
    c) we are actually outputting to cout
    d) the insertion operator is overloaded in ofstream
18. eof( ) is the function used for _____
    a)asserting no errors in a file b)appending data to a file
    c) counting the amount of data in a file d)checking for end of file
19. Which of the following header files is not used for file handling?
    a) Ofstream.h b)Fstream.h c)Ifstream.h d)Console.h
20. ios::app is a file open mode for _____
    a) opening a file b)open the file for input
    c)open file for output d)seek to end of file before each write

PART - B (3 * 2 = 6)

Answer All the Questions

21. Define Constructor.

22. Define Operator Overloading.

23. Define Pointer.

PART - C (3 * 8 = 24)

Answer All the Questions

24. a) Elucidate the rules for overloading operator.

(OR)

b) Explain in detail about type conversions.

25. a) Elaborate manipulation of string using operators.

(OR)

b) Write a program for opening and closing a file with example

26. a) Briefly explain about destructor.

(OR)

b) Explain how the files can be works for stream operations with example.

No. of copies : 45

Reg No.....

[17CCP104]

KARPAGAM UNIVERSTIY
Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
Coimbatore – 641021
(For the candidates admitted from 2017 onwards)
I M.Com (CA)
First Internal Test, August - 2017
OBJECT ORIENTED PROGRMMING WITH C++

Time: 2 hours Maximum: 50 marks
Date & sess: 31.08.2017 & FN

PART – A (20 * 1 = 20)
Multiple choice Questions

1. C++ is an _____

(a) High level language (b)OOP language (c) Assembly language (d) procedure language

2. C++ is a subset of _____

(a) C language (b) Java (c)Visual Basic (d)ASP

3. The smallest individual units in a program are known as _____

(a)Tokens (b)Control Structure (c) Expressions (d) Keywords

4. The _____is an exit-controlled loop

(a) while (b)do-while (c) for (d) switch

5. The _____is an entry-controlled loop

(a) while (b) do-while (c) for (d) switch

6. <conio.h> stands for _____

(a) console input and output (b)standard input and output

(c)console input (d) console output

7. OOPS stands for

(a)object oriented programming (b)object oriented procedure
(c)object (d)Functions

8. Program is divided in to

(a)objects (b)functions (c)class (d)Methods

9. Class should end with

(a) ; (b) : (c) :: (d).

10. The wrapping up of data and functions in to a single unit is called

(a)encapsulation (b)data abstraction (c)polymorphism (d)inheritance

11. The ">>" in c++ is known as

(a)insertion (b)extraction operator (c)greater (d)lesser

12 The keywords private and public are known as _____ labels

(a) Static (b) Dynamic (c) Visibility (d) Const

13. Find odd man out _____

(a) int (b) float (c) char (d)array

14.Operator which allows to assign the multiple meaning to Operation is known as_____

(a) Operator Overloading (b) Over Loading
(c) Method Overloading (d) Function Overloading

15. Binary Operator requires _____Operand

(a) One (b)Two (c) Three (d) Four

16. c++ was developed by

(a)Bjarne Stroustrup (b)Balagurusamy (c)Denish (d)Ritche

17._____ is the process by which objects of one class acquire the properties of objects of another class.

(a)Objects (b)Data abstraction (c)Encapsulation (d)Inheritance

18._____ refers to fixed value that do not change.

(a)identifiers (b)constants (c)variables (d)dynamic

19. _____ refers to the act of representing essential features without including the background details or explanations

(a)encapsulation (b)inheritance (c)Dynamic binding (d)Abstraction

20. _____ is a scope resolution operator.

(a). (b):; (c):: (d).

PART – B (3 * 2 = 6)

Answer All the Questions

21. Note down the Structure of C++ Program

22. Define Operator

23. How will you specify a Class.?

PART – C (3 * 8 = 24)

Answer All the Questions

24. a) Describe the Basic Concepts of OOPs

(OR)

b) List out i) Application of Oops ii) C++ Statements

25. a) Illustrate: Control Structures.

(OR)

b) Explain : Functions in C++

26. a) How will you define member functions.

(OR)

b) Write a program to print the book details. (member functions should be defined outside the class)

Enable | Enlighten | Enrich
(Deemed to be University)
(Under Section 3 of UGC Act 1956)

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University Established Under Section 3 of UGC Act 1956)

Coimbatore – 641 021.

LESSON PLAN
DEPARTMENT OF COMMERCE WITH COMPUTER APPLICAITON

STAFF NAME: Dr.S.HEMALATHA

SUBJECT NAME: OBJECT ORIENTED PROGRAMMING WITH C++

SUB.CODE:17CCP104

SEMESTER: I CLASS: I M.COM (CA)

UNIT 1

S.No	Lecture Duration Period	Topics to be Covered	Support Material/Page Nos
1	1	**PRINCIPLES OF OBJECT ORIENTED PROGRAMMING** • Introduction and Evolution of OOP's	T:4-7 R2:1-3
2	1	• Procedure oriented and object oriented paradigm	
3	1	**Basic concepts of OOP's**	T:7-15
4	1	**Benefits of OOPs** • Objects • Classes • Data abstraction and Encapsulation • Inheritance	R2:6-11
5	1	**Benefits of OOPs** • Polymorphism • Dynamic binding • Message passing	

6	1	• **Object Oriented Languages**	W1,W2
7	1	**Applications of Oops**	W1,W2
8	1	**Beginning with C++** • **Introduction to C++**	R3:36-55 T:19-30
9	1	• **Applications of C++**	R2:13-15
10	1	• **C++ Statements**	
11	1	• **Structure of C++ Program**	W1,W2
12	1	Recapitulation	
		Important Questions Discussion	
		Total No of Hours Planned For Unit 1=12	

UNIT – II

S.No	Lecture Duration Period	Topics to be Covered	Support Material/Page Nos
1	1	**Tokens ,Expressions and Control Structures** • Tokens, Keywords, Identifiers	T:35-49
2	1	**Basic and User Defined Data Types** • Variable Declaration • Variable Initialization	R2:32-38 W3
3	1	**Operators** • Operators in C++ • Operator Overloading	T:49-64 R2:46-66 W2
4	1	**Operators** • Operator Precedence	
5	1	**Control Structures Decision** • If & If-Else Statements • Jump Statements	T:64-69 R2:112-159 R1:124-138
6	1	**Control Structures Decision** • Goto Statements • Break Statements	
7	1	**Looping** • Switch case Statements • Do-While Statements	R!:185-210 W1
8	1	**For Statements**	W3
9	1	**Functions in C++** • The main Function • Function Prototyping	T:77-90 R2:179-185
10	1	**Functions in C++** • Call by Reference, Return by Reference	R3:273-275
11	1	• Inline Functions	
12	1	**Functions in C++** • Function Overloading	

		• Recapitulation Important Questions Discussion	
		Total No of Hours Planned For Unit II=12	

UNIT – III

S.No	Lecture Duration Period	Topics to be Covered	Support Material/Page Nos
1	1	**CLASSES AND OBJECTS** • Introduction • Specifying a Class	T:96-119
2	1	• Defining Member Functions	
3	1	• Nesting of Member Functions	R2:298-423 W2
4	1	• Arrays within a Class	W3 R3:326-362 W2
5	1	• Static Data Members • Static Member Function	
6	1	• Private Member Functions	
7	1	• Array of Objects • Objects as Function Arguments	T:119-135
8	1	• Friendly Functions • Pointers to Members	R!:185-210 W1
9	1	**Constructors and Destructors** • Constructors	T:144-164
10	1	• Copy Constructor • Dynamic Constructor	R2:455-476 W4
11	1	• Constructor two-dimensional Arrays	R3:499-502
12	1	• Destructor Recapitulation Important Questions Discussion	
		Total No of Hours Planned For Unit II1=12	

UNIT – IV

S.No	Lecture Duration Period	Topics to be Covered	Support Material/Page Nos
1	1	**Operator overloading** • Type Conversion –Introduction	T:171-186 R3:571-669
2	1	• Defining Operator Overloading	W2
3	1	• Overloading unary and binary operator • Overloading binary operator using friends	
4	1	• Manipulation String using Operators	T:186-195 R2:518-524 W3
5	1	• Rules for Operator Overloading • Type Conversions	
6	1	**Inheritance:** • Extending Classes • Defining Derived Classes	T:202-232
7	1	• Single Inheritance	R2:538-548
8	1	• Multilevel Inheritance	
9	1	• Multiple Inheritance • Hierarchical Inheritance	W3 W4
10	1	• Hybrid Inheritance	
11	1	• Virtual Base Classes • Abstract Classes	R3:708 W4
12	1	**Recapitulation** **Important Questions Discussion**	
		Total No of Hours Planned For Unit IV=12	

UNIT – V

S.No	Lecture Duration Period	Topics to be Covered	Support Material/Page Nos
1	1	**Pointers, Virtual Functions and Polymorphism** • **Pointers- Introduction**	T:251-343
2	1	Pointers to Objects	
3	1	• Pointers to Derived Classes	
4	1	• Virtual Functions	
5	1	Working with Files • Introduction • Classes for File Stream Operations	R3:841-856
6	1	Opening and Closing File	R2: 638-645 W2
7	1	• File Pointers	
8	1	• File Pointers Manipulations	
9	1	• Sequential I/O Operations	W3
10	1	**Recapitulation** **Important Questions Discussion** **ESE Question Paper Discussions**	
11	1	**Previous ESE Question Paper Discussions**	
12	1	**Previous ESE Question Paper Discussions**	
		Total No of Hours Planned For Unit V=12	
Total Planned Hours	**60**		

TEXT BOOK

1. Balagurusamy, E. (2013). *Object Oriented Programming With C++*, 6th edition, New Delhi: Tata McGraw Hill Publishing Company Ltd.

REFERENCE BOOKS

1. BjarneStroustroup. (2014). *Programming -- Principles and Practice using C++*, 2nd Edition, Addison-Wesley.
2. BjarneStroustrup,. (2013). *The C++ Programming Language*, 4th Edition, Addison- Wesley.
3. Paul Deitel, Harvey Deitel. (2011*). C++ How to Program*, 8th Edition, Prentice Hall.
4. D.Ravichandran. (2010). *Programming with C++*.3rd Edition.. New Delhi: Tata McGraw Hill Publishing Company Ltd.

WEBSITES

W1: http://www.hscripts.com

W2: http://www3.ntu.edu

W3: http://www.bcanotes.com

W4: http://www.ddegjust.ac.in

Reg. No.....

[11CCP105]

KARPAGAM UNIVERSITY

(Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2011 onwards)

M.Com DEGREE EXAMINATION, NOVEMBER 2011

First Semester

COMMERCE (COMPUTER APPLICATIONS)

OBJECT ORIENTED PROGRAMMING WITH C++

Time: 3 hours Maximum : 100 marks

PART – A (15 x 2 = 30 Marks)
Answer ALL the Questions

1. Define polymorphism.
2. Define Encapsulation.
3. What is Oops?
4. Define constant.
5. What is meant by operator overloading?
6. Define do-while statement with syntax.
7. Define class
8. What is nesting of member function?
9. Define Destructor.
10. Define Inheritance.
11. What is an operator function? Give syntax.
12. Define abstract class.
13. What is this pointer?
14. What is static linking?
15. Write any 2 of the file stream clauses function.

PART B (5 X 14= 70 Marks)
Answer ALL the Questions

16. a. Explain the basic concepts of Oops.

Or

b. Structure of C++. Explains.

17. a. What is token? Explain its types

Or

b. Explain the control structures with an example.

18. a. Explain constructor with example.

Or

b. Briefly explain friendly function with example.

19. a. Explain the manipulation of string using operators with example program.

Or

b. Define inheritance and explain its types.

20. a. Explain pointers to objects with an example.

Or

b. Explain virtual functions and its rules.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

Semester I

		L	T	P	C
17CCP104	**OBJECT ORIENTED PROGRAMMING WTIH C++**	**4**	**-**	**-**	**4**

Course Objective

- Course instructs on Tokens and Control Structures, Classes and Objects, Operator Overloading, Pointers and Polymorphism

Learning Outcomes

- This course aims to enlighten the students on the Application of OOPS adopted in practice.
- To gain a sound knowledge in the basics of Operation Overloading
- To gain a sound knowledge in the application of function and polymorphism

UNIT I

Oriented programming- Principles of Object Oriented Programming – a Look at Procedure and Object Oriented Paradigm – Basic Concepts of Object Oriented Programming- Basic Concepts of Object Oriented Programming- Benefits of OOP – Object Oriented Languages – Application of OOP, Beginning with C++ - What is C++? – Applications of C++ - C++ Statements – Structure of C++ Program.

UNIT II

Tokens and Control Structures- Tokens , Expressions and Control Structures – Token – Keywords – Identifiers – Basic and User – Defined Data Types – Operators in C++ - Operator Overloading- Operator Precedence – Control Structure Functions in C++ - the Main

Function – Call By Reference – Return by Reference – In line Function – Function Overloading.

UNIT III

Classes and objects – Introduction- Specifying a Class – Defining Member Function – Nesting of Member Functions - Private Member Functions- Arrays within a Class – Static Data Members- Static Member Functions- Array of Objects – Objects as Function Arguments- Friendly Functions- Pointers to Members. Constructors and Destructors

UNIT IV

Operator Overloading – Type Conversions – Introduction – Defining Operator Overloading– Manipulation of String using Operators – Rules for Overloading Operators- Types Conversions. Inheritance – Abstract Classes.

UNIT V

Pointers, Virtual functions and Polymorphism – Pointers to Objects – This Pointer – Pointer to Derived Classes – Virtual Functions. Working with Files – Classes for File Stream Operations Opening and Closing a File

Suggested Readings:

Text Book:

1. Balagurusamy, E. (2013). *Object Oriented Programming With C++*, 6th edition, New Delhi: Tata McGraw Hill Publishing Company Ltd.

Reference Books:

1. BjarneStroustroup. (2014). *Programming -- Principles and Practice using C++*, 2nd Edition, Addison-Wesley .
2. BjarneStroustrup,. (2013). *The C++ Programming Language*, 4th Edition, Addison- Wesley.
3. Paul Deitel, Harvey Deitel. (2011*). C++ How to Program*, 8th Edition, Prentice Hall,.
4. D.Ravichandran. (2010). *Programming with C++*.3rd Edition.. New Delhi: Tata McGraw Hill Publishing Company Ltd.

Websites

- W1: http://www.hscripts.com
- W2: http://www3.ntu.edu
- W3: http://www.bcanotes.com
- W4: http://www.ddegjust.ac.in

Enable | Enlighten | Enrich
(Deemed to be University)
(Under Section 3 of UGC Act 1956)

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT : **OBJECT ORIENTED PROGRAMMING WITH C++**
SEMESTER : I
SUBJECT CODE: **17CCP104** **CLASS** : **I M.COM CA**

UNIT I

Oriented programming- Principles of Object Oriented Programming – a Look at Procedure and Object Oriented Paradigm – Basic Concepts of Object Oriented Programming- Basic Concepts of Object Oriented Programming- Benefits of OOP – Object Oriented Languages – Application of OOP, Beginning with C++ - What is C++? – Applications of C++ - C++ Statements – Structure of C++ Program.

Unit 1

Introduction

Object-Oriented Programming (OOP) is the term used to describe a programming approach based on **objects** and **classes**. The object-oriented paradigm allows us to organize software as a collection of objects that consist of both data and behaviour. This is in contrast to conventional functional programming practice that only loosely connects data and behaviour.

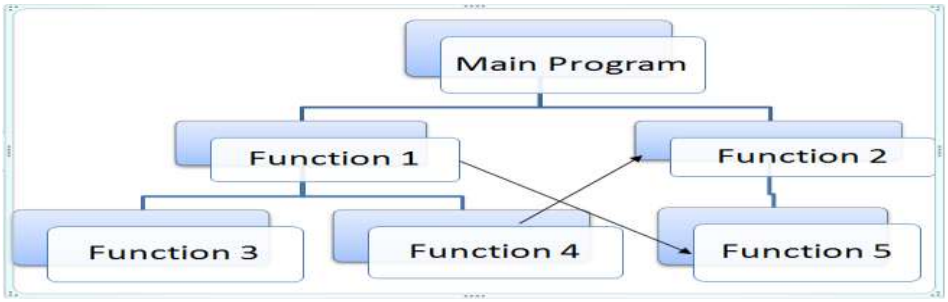
The object-oriented programming approach encourages:

- Modularization: where the application can be decomposed into modules.
- Software re-use: where an application can be composed from existing and new modules.

Procedure-Oriented Programming

High level languages such as COBOL, FORTRAN and C, is commonly known as procedure oriented programming (POP). In the procedure oriented programming, program is divided into sub programs or modules and then assembled to form a complete program. These modules are called functions.

In a multi-function program, many important data items are placed as global so that they may be accessed by all functions. Each function may have its own local data. If a function made any changes to global data, these changes will reflect in other functions. Global data are more unsafe to an accidental change by a function. In a large program it is very difficult to identify what data is used by which function.



Characteristics by procedure-oriented programming are:

Emphasis is on doing things (algorithms).

Large programs are divided into smaller programs known as functions.

Most of the functions share global data.

Data move openly around the system from function to function.

Functions transform data from one form to another.

Employs top-down approach in program design.

Features of POP

1. Mainly focused on writing the algorithms.
2. Most function uses Global data for sharing which are accessed freely from function to function in the system.
3. POP follows the top down approach in program design.
4. It does not have data hiding options.
5. Functions transform data from one form to another.
6. Data can be moved openly from one function to another around the system.
7. Sub-division of large program into smaller programs called functions.
8. Overloading process is not applicable in POP.

Difference Between Procedure Oriented Programming (POP) & Object Oriented Programming (OOP)

	Procedure Oriented Programming	Object Oriented Programming
Divided Into	In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
Importance	In POP,Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
Approach	POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
Expansion	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data cannot move easily from function to function, it can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

Principles of Object Oriented Programming: Basic concepts of Object Oriented Programming

- Objects
- Classes
- Data abstraction and Encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

Objects:

- An object can be considered a "thing" that can perform a set of related activities.
- The set of activities that the object performs defines the object's behavior.
- For example, the hand can grip something or a Student (object) can give the name or address.
- Objects are run time entity or real world entity.

Classes:

- A class is simply a representation of a type of object.
- It is the blueprint/ plan/ template that describe the details of an object.
- A class is the blueprint from which the individual objects are created.
- Class is composed of three things: a name, attributes, and operations.
- For example Student is an object has name, age, course, etc as attributes. Read, write, etc as operations

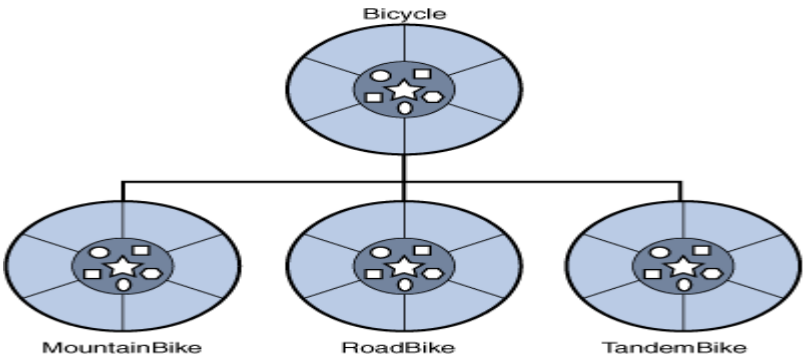
Data abstraction and Encapsulation

- The encapsulation is the inclusion within a program object of all the resources need for the object to function - basically, the methods and the data.
- In OOP the encapsulation is mainly achieved by creating classes, the classes expose public methods and properties.
- The class is kind of a container or capsule or a cell, which encapsulate the set of methods, attribute and properties to provide its indented functionalities to other classes.
- In that sense, encapsulation also allows a class to change its internal implementation without hurting the overall functioning of the system.
- That idea of encapsulation is to hide how a class does it but to allow requesting what to do.
- Abstraction is an emphasis on the idea, qualities and properties rather than the particulars.
- The importance of abstraction is derived from its ability to hide irrelevant details and from the use of names to reference objects.
- Abstraction is essential in the construction of programs.
- It places the emphasis on what an object is or does rather than how it is represented or how it works.

Inheritance

- Ability of a new class to be created, from an existing class by extending it, is called inheritance.
- Different kinds of objects often have a certain amount in common with each other.

- Object-oriented programming allows classes to inherit commonly used state and behavior from other classes.
- In this example, Bicycle now becomes the super class of MountainBike, RoadBike, and TandemBike. In the Java programming language, each class is allowed to have one direct super class, and each super class has the potential for an unlimited number of subclasses:



- The new class created is called as derived class
- The existing class is called as base class.
- The base class provides the property the derived class receives the property.
- It reduces the complexity of the programming.
- This is the most common and most natural and widely accepted way of implement this relationship.

Polymorphism

- Polymorphism is the process taking more then one form.
- More precisely Polymorphisms mean the ability to request that the same operations be performed by a wide range of different types of things.

- In OOP the polymorphisms is achieved by using many different techniques named method overloading, operator overloading and method overriding,
- The method overloading is the ability to define several methods all with the same name.
- The operator overloading (less commonly known as ad-hoc polymorphisms) is a specific case of polymorphisms in which some or all of operators like +, - or == are treated as polymorphic functions and as such have different behaviors depending on the types of its arguments.

Dynamic binding

- Dynamic binding is the process of resolving the function to be associated with the respective functions calls during their runtime rather than compile time.

Message passing

- Every data in an object in oops that is capable of processing request known as message.
- All objects can communicate with each other by sending message to each other
- Message passing, also known as interfacing, describes the communication between objects using their public interfaces.

Benefits of OOP

1. **Reusability:** In OOP's programs functions and modules that are written by a user can be reused by other users without any modification.
2. **Inheritance:** Through this we can eliminate redundant code and extend the use of existing classes.

3. **Data Hiding:** The programmer can hide the data and functions in a class from other classes. It helps the programmer to build the secure programs.
4. **Reduced complexity of a problem:** The given problem can be viewed as a collection of different objects. Each object is responsible for a specific task. The problem is solved by interfacing the objects. This technique reduces the complexity of the program design.
5. **Easy to maintain and Upgrade:** OOP makes it easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
6. **Message Passing:** The technique of message communication between objects makes the interface with external systems easier.
7. **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.

Object Oriented Language

Object-oriented programming is not the right of any particular languages.

Like structured programming, OOP concepts can be implemented using languages such as C and Pascal. It is specially designed to support the OOP concepts makes it easier to implement them.

Object Oriented Languages has two categories:

1. **Object-based programming languages**
2. **Object-oriented programming languages**

Object-based programming is the style of programming that primarily supports encapsulation and object identity.

Major feature that are required for object based programming are:

- Data encapsulation
- Data hiding and access mechanisms
- Automatic initialization and clear-up of objects

- Operator overloading Languages that support programming with objects are said to the objects-based programming languages. They do not support inheritance and dynamic binding. Ada is a typical object-based programming language.

Object-Oriented programming language incorporates all of object-based programming features along with two additional features, namely, inheritance and dynamic binding.

Object-oriented programming can therefore be characterized by the following statements:

Object-based features + inheritance + dynamic binding

Applications of OOPs

- For Develop Graphical related application like computer and mobile games.
- To evaluate any kind of mathematical equation use C++ language.
- C++ Language are also used for design OS. Like window xp.
- Google also use C++ for Indexing
- Few parts of apple OS X are written in C++ programming language.
- Internet browser Firefox are written in C++ programming language
- All major applications of adobe systems are developed in C++ programming language. Like Photoshop, ImageReady, Illustrator and Adobe Premier.
- Some of the Google applications are also written in C++, including Google file system and Google Chromium.
- C++ are used for design database like MySQL.

Beginning with C++

What is C++: C++ is a general-purpose object-oriented programming (OOP) language, developed by Bjarne Stroustrup, and is an extension of the C language. It is therefore possible to code C++ in a "C style" or "object-oriented

style.".C++ is one of the most versatile languages in the world. It is used nearly everywhere for everything… systems programming (operating systems, device drivers, database engines, embedded, Internet of Things, etc.)

Applications of C++

- Client-Server System
- Object Oriented Database
- Object Oriented Distributed Database
- Real Time Systems Design
- Simulation and Modeling System
- Hypertext, Hypermedia
- Neural Networking and Parallel Programming
- Decision Support and Office Automation Systems
- CIM/CAD/CAM Systems
- AI and Expert Systems

C++ Statements

Preprocessor directives

A preprocessor directive begins with the character #. This must either be the first character of the line or the first character of the line after some leading whitespace.

Comments

Comments may be of the form:

/ / *Comment* \n

(Or)

/* comment */

- The first form allows a trailing comment on a single line, while the second form allows comments that span multiple lines.
- Comments may appear anywhere.

Declarations

- Declarations give the compiler information about the types, storage requirements and initial values of identifiers.
- **General form:** void type identifier intializer;

Function Declarations

void type identifier (formal_argument_list)

{

function_body

}

Executable statements

while (expression) statement

do statement while (expression)

for (expression_1; expression_2; expression_3) statement

equivalent to:

expression_1; while (expression_2) { **statement expression_3** }

switch (expression)

{

declarations

case constant_expression: statement

...

default: statement

}

break;

continue;

return;

return expression;

goto statement_label;

statement_label: executable_statement

Structure of C++ program

- Include section
- Class declaration
- Member function definition
- Main function program

Output Operator:

cout<<

- **Syntax:**

  cout<<argument;

- **Example:**

  cout<<"Welcome to C++";

Program:

```
#include<iostream.h>
void main()
{
  cout<<"Welcome to C++";
}
```

Compiling and Linking:

- Create the source code.
- Save the code with extension .cpp
- Compile the program with

Alt + F9 (or)

Select Compile option from Compile menu

- Run the program with

Ctrl + F9 (or)

Select Run option from Run menu

Input Operator:

Cin>>

- **Syntax**:

cin<<argument;

- **Example:**

cin>>a;

a is a variable.

Program:

```
#include<iostream.h>
void main()
{
  float n1,n2,sum,avg;
  cout<<"Enter the two No:";
  cin>>n1>>n2;
  sum=n1+n2;
  avg=sum/2;
  cout<<"Sum:"<<sum;
  cout<<"\nAverage:"<<avg;
}
```

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT: : OBJECT ORIETNED PROGRAMMING WITH C++
SEMESTER : I
SUBJECT CODE: 17CCP104 CLASS : I M.COM CA

UNIT 1

S.NO	QUESTIONS	OPTION 1	OPTION 2	OPTION 3	OPTION 4	ANSWER
1	Procedure oriented programming basically consist of	objects	functions	class	procedure	functions
2	OOPS stands for	object oriented	object oriented procedure	object	Functions	object oriented
3	Program is divided in to	objects	functions	class	Methods	objects
4	The basic run time entities in an object oriented system	Class	object	data abstraction	encapsulation	object
5	The wrapping up of data and functions in to a single unit is called	encapsulation	data abstraction	polymorphism	inheritance	encapsulation
6	The ">>" in c++ is known as	insertion	extraction operator	greater	lesser	extraction operator
7	The "<<" is known as	insertion operator	extraction operator	lesser	greater	insertion operator
8	c++ program would contain_____sections	2	6	4	3	4
9	c++ was developed by	Bjarne stroustrup	balagurusamy	denesh	ritche	Bjarne stroustrup
10	_____ is the process by which objects of one class acquire the properties of objects of another class.	Objects	Data abstraction	Encapsulation	Inheritance	Inheritance
11	When data and function are combined into one entity, we call it as _____	Polymorphism	Inheritance	Encapsulation	abstraction	Encapsulation
12	OOP language member function are called _____	Method	Data	Entity	Object	Method
13	Collection of similar objects are called_____	Function	Class	Operate	Procedure	Class
14	How many types of polymorphisms are supported by C++?	1	2	3	4	2
15	OOPS data is considered the most important factor and given a _____ status	Even	Odd	Prime	main	Prime
16	A local variable is defined within a _____	Class	Object	Function	procedure	Function
17	Objects are _____ of classes	Method	Function	Member	main	Member
18	Functions are subjected to 3 process _____	Declaration	Class	Definition	Prototype	Prototype
19	C++ is an _____	High level language	OOP language	Assembly	procedure language	OOP language
20	C++ introduce a new command symbol _____	Star	Double Slash	Single Slash	braket	Double Slash
21	C++ is a subset of _____	C language	Java	Visual Basic	ASP	C language
22	C++ program is a collection of _____	Function	Class	Operator	Experssions	Class
23	The smallest individual units in a program are known as _____	Tokens	Control Structure	Expressions	Keywords	Tokens
24	_____is a decision making statement	for	jump	break	if	if
25	Inline function needs more_____	variables	functions	memoryspace	control structures	memoryspace
26	Test is performed at the _____of the for loop.	top	middle	end	program terminates	top
27	_____ is used only at the time of run time	Dynamic Binding	Message Passing	None	polymorphism	Dynamic Binding
28	Which of the following is not a part of OOP.	Multitesting	Information hiding	Polymorphism	Inheritance	Multitesting
29	_____ refers to the the act of representing essential features without including the background details or explanations.	Encapsulation	Inheritance	Abstraction	Dynamic Binding	Abstraction
30	The keywords_____and _____ are known as visibility labels.	Statement & Function	Continue & Break	Public & int	Private & Public	Private & Public
31	Which of the following is not a part of OOP.	Multitesting	Information hiding	Polymorphism	Inheritance	Multitesting
32	Data is hidden and cannot be accessed by_____	Internal Function	External Function	Main Function	All Function	External Function
33	_____ is used only at the time of run time	Dynamic Binding	Message Passing	Inheritance	polymorphism	Dynamic Binding
34	OOPS follows _____ approach in program design	bottom-up	top-down	middle	top	bottom-up
35	Objects takeup _____ in the memory	space	address	memory	bytes	space
36	Attributes are sometimes called _____	data members	methods	messages	functions	data members
37	The functions operate on the datas are called_____	data members	methods	messages	classes	methods
38	_____ means that the code associated with a given procedure call is not known until the time of the call at run-time	late binding	Dynamic binding	Static binding	Method Binding	Dynamic binding
39	large programs are divided into smaller programs known as _____	functions	pointer	object	class	functions
40	In identifier, a declared keywords cannot be used as	variable function	variable names	definition	object	variable names
41	If the body of the statement in c++ has multiple statements that should be enclosed with	braces	rounded	square	bracket	braces
42	Function prototype is a declaration statement in the	called program	calling method	calling program	called method	calling program
43	The use of the samething for different purpose	overloading	sequence	class	overriding	overloading
44	The class declaration is similar to a	struct declaration	function declaration	object	method	struct declaration
45	The variable declared inside the class are known as	member functions	data members	functions	procedure	data members
46	_____ Passes Control Outside the Loop	Continuous Statement	Break Statement	Do while Statemen	comma statement	Break Statement
47	Operator which allows to assign the multiple meaning to Operation is known as _____	Operator Overloading	Over Loading	Method Overloadir	function overloading	Operator Overloading
48	Identifiers refers to the name of the variable	extraction	identifiers	constants	static	constants
49	The identifiers only _____ characters are permitted.	Numeric	Constants	Variable	static	Numeric
50	_____ refers to fixed value that do not change.	identifiers	constants	variables	dynamic	constants

51	The _____ variables must be defined outside the class.	public member	static member	private member	protected member	public member
52	We pass argument to the functions while calling them, is called as	Argument	Class	Parameters	methods	Parameters
53	The member functions can be defined either inside or _____ the class.	Inside	Outside	Middle side	Both side	Outside
54	We pass variables or constants and collect them in simple variable, called _____	Call by variable	Call by value	Call by words	Call by reference	Call by value
55	_____ is nothing but collection of elements which are all have the same name.	Function	Arrays	Difference	class	Arrays
56	_____ is the process by which object of one class acquire the properties of object of another class.	Data encapsulation	Polymorphism	Inheritance	abstraction	Inheritance
57	Opp"s allows decomposition of a problem into a number of entities called _____	Object	Class	Function	procedure	Class
58	The most important features of C++ is the _____	Objects	Functions	Class	procedure	Class
59	A data member of a class can be declared as a _____	Static	Structure	Const	method	Structure
60	_____ oriented programming basically consist of functions	Functions	procedures	variables	constants	procedures

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed University Established Under Section 3 of UGC Act 1956)

Coimbatore – 21

(For the candidates admitted from 2017 onwards)

DEPARTMENT OF COMMERCE (CA)

SUBJECT : OBJECT ORIENTED PROGRAMMING WITH C++
SEMESTER : I
SUBJECT CODE: 17CCP104 CLASS : I M.COM CA

POSSIBLE QUESTIONS – UNIT I

PART A (1 Mark)

(Online Examination)

PART B (2 Marks)

1. Define Objects
2. What is C++ Statements
3. List out the Basic Concepts of Oops
4. Specify object Oriented Languages
5. List out Applications of Oops
6. List out Applications of OOPs
7. What is Structure of C++ Program
8. Differentiate between POP and OOPs
9. What is Procedure Oriented Programming
10. What is Object Oriented Programming
11. What is C++ statements
12. What is data abstraction
13. Write any four features of OOPS

PART C (6 Marks)

1. Describe about Procedure Oriented Programming
2. Explain Object Oriented Programming
3. Write a Program for Structure of C++ Program
4. List out Benefits of Oops
5. Explain in detail about basic concepts of Object Oriented Programming.
6. Write a program to find average of two numbers
7. Differentiate between POP and OOPs.
8. Enumerate History of C++
9. Write a program to find largest of three given numbers.
10. Write in about object oriented languages

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT: : OBJECT ORIETNED PROGRAMMING WITH C++
SEMESTER : I
SUBJECT CODE: 17CCP104 CLASS : I M.COM CA
UNIT 2

S.NO	QUESTIONS	OPTION 1	OPTION 2	OPTION 3	OPTION 4	ANSWER
1	_____ is a scope resolution opertor.	:	:;	::	,	::
2	The class _____ describes the type and scope of its members	Functions	declaration	objects	methods	declaration
3	The class _____ describes how the class function are implemented	Function definition	declaration	arguments	objects	Function definition
4	The keywords private and public are known as _____ labels	Static	dynamic	visibility	const	visibility
5	The class members that have been declared as _____ can be accessed only from within the class	Private	public	static	protected	Private
6	The class members that have been declared as _____ can be accessed from outside the class also	Private	public	static	protected	public
7	The variables declared inside the class are called as _____	Function variables	data members	member function	functions	data members
8	The functions which are declared inside the class are known as _____	Member function	member variables	data variables	variables	Member function
9	The class variables are known as	Functions	members	objects	methods	objects
10	The symbol is called the scope resolution operator	>	::	<<	::*	::
11	A member function can call another member function directly, without using the _____ operator	Assignment	equal	dot	greater than	dot
12	A member function can call another member function directly, without using the _____ operator	Assignment	equal	dot	greater than	dot
13	A _____ member variable is initialized to zero when the first object of its class is created	Dynamic	constant	static	protected	static
14	_____ Variables are normally used to maintain values common to the entire class.	Private	protected	public	static	static
15	When a copy of the entire object is passed to the function it is called as _____	Pass by reference	pass by function	pass by pointer	pass by value	pass by value
16	When the address of the object is transferred to the function it is called as _____	pass by reference	pass by function	pass by pointer	pass by value	pass by reference
17	A _____ function can be invoked like a normal function without the help of any object	Void	friend	inline	main	friend
18	The _____ member variables must be defined outside the class.	Static	private	public	protected	Static
19	A friend function, although not a member function, has full access right to the _____ members of the class	Static	private	public	protected	private
20	_____ enables an object to initialize itself when it is created	Destructor	constructor	overloading	operator	constructor
21	_____ destroys the objects when they are no longer required	Destructor	constructor	overloading	operator	Destructor
22	The _____ is special because its name is the same as the class name.	Destructor	static	constructor	overloding	constructor
23	A constructor that accepts no parameters is called the _____ constructor	Copy	default	multiple	basic	default
24	Constructors are invoked automatically when the _____ are created	Datas	classes	objects	methods	objects
25	How many types of constructor are there in C++?	1	2	3	4	3
26	How many constructors can present in a class?	1	2	3	Multiple	Multiple
27	Constructors make _____ calls to the operators new and delete when memory allocation is required	Explicit	implicit	function	objects	implicit
28	The constructors that can take arguments are called _____ constructors	Copy	multiple	parameterized	constructor	parameterized
29	The constructor function can also be defined as _____ function	Friend	inline	default	main	inline
30	When a constructor can accept a reference to its own class as a parameter, in such cases it is called as _____ constructors	Multiple	copy	default	parameterized	Multiple
31	When more than one constructor function is defined in a class, then the constructor is said to be _____	Multiple	copy	default	overloaded	overloaded
32	C++ complier has a _____ constructor, which creates objects, even though it was not defined in the class.	Explicit	default	implicit	copy	implicit
33	A _____ constructor is used to declare and initialize an object from another object	Default	copy	multiple	parameterized	copy
34	The process of initializing through a copy constructor is known as _____ initialization	Overloaded	multiple	copy	parameterized	copy
35	A _____ constructor takes a reference to an object of the same class as itself as an argument	Delete	new	copy	Multiple	copy
36	Allocation of memory to objects at the time of their construction is known as _____ construction	Static	copy	dynamic	default	dynamic
37	We can create and use constant objects using _____ keyword before object declaration.	Static	new	const	delete	const
38	A destructor is preceded by _____ symbol	Dot	astrick	colon	tilde	tilde
39	_____ is used to allocate memory in the constructor	Delete	binding	free	new	new
40	Constructors and destructors are automatically inkoved by _____	operating system	main	complier	object	complier
41	Constructors is executed when _____	object is destroyed	object is declared	method is declared	method is destroyed	object is declared
42	The destructor is executed when	object goes out of scope	when object is not used	when object contains nothing	when method do nothing	object goes out of scope
43	The members of a class are by default	protected	private	public	public & private	private
44	The _____ is executed at the end of the function when objects are of no used or goes out of scope	destructor	constructor	inheritance	copy constructor	destructor
45	Static member variables can be defined _____ the class.	inside	outside	above	below	outside
46	A function can also return an _____	object	class	methods	variables	object
47	A constructor has the same name as that of a _____ name	method	object	c lass	variables	class
48	C++ provides a special member function called _____	Destructor	Constructor	variables	methods	Constructor
49	_____ enables an object to intialize itself when it is Created.	Constructor	Destructor	Class	variables	Constructor
50	C++ provides another member function called the _____ that destroys the object when they are no longer required.	Destructor	Constructor	member	functions	Destructor
51	_____ are data items whose values cannot be changed.	Identifiers	Constants	Integer	variables	Constants
52	How many instances of an abstract class can be created?	1	5	13	0	0

53	By default , member of a class are _____	public	private	protected	public & private	private
54	We can define the member function _____ or outside the class	inside	outside	main function	public & private	inside
55	Find odd man out _____	int	float	char	array	array
56	enum stands for _____	enumeration	e-numeration	enum	e-num	euumeration
57	C++ allows us to have _____ of objects	array	method	Class	function	array
58	<conio.h> stands for	console input and output	standard input and output	console input	console output	console input and output
59	_____ operator is used for destructor	^	&	*	~	~
60	Class should end with	;	:	::	,	;

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed University Established Under Section 3 of UGC Act 1956)

Coimbatore – 21

(For the candidates admitted from 2017 onwards)

DEPARTMENT OF COMMERCE (CA)

SUBJECT : OBJECT ORIENTED PROGRAMMING WITH C++

SEMESTER : I

SUBJECT CODE: 17CCP104 **CLASS : I M.COM CA**

POSSIBLE QUESTIONS – UNIT II

PART A (1 Mark)

(Online Examinations)

PART B (2 Marks)

1. Define Tokens
2. Define Keywords
3. Define Identifiers
4. Define User-defined datatypes
5. Define basic-defined datatypes
6. Define datatypes
7. What is Operators in C++
8. Define Operator Overloading
9. Define Control Structures
10. What is inline Functions
11. What is Call by reference
12. What is return by reference
13. Define friend Functions
14. Define Virtual Functions
15. Write the syntax for loop

PART C (6 Marks)

1. Describe about basic data types
2. Write about different types of Operators in C++
3. Enumerate Control Structures with example.
4. Explain about Inline functions
5. Explain Friend Functions with example
6. Describe about function overloading
7. Describe – Functions in C++
8. Write a program using operators in C++
9. Elucidate on looping statements in C++.
10. Explain: i)Inline function ii) Function overloading

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT : **OBJECT ORIENTED PROGRAMMING WITH C++**
SEMESTER : I
SUBJECT CODE: **17CCP104** **CLASS** : **I M.COM CA**

UNIT III

Classes and objects – Introduction- Specifying a Class – Defining Member Function – Nesting of Member Functions - Private Member Functions- Arrays within a Class – Static Data Members- Static Member Functions- Array of Objects – Objects as Function Arguments- Friendly Functions- Pointers to Members. Constructors and Destructors

Unit – III

Classes and objects : Specifying a class

- Class is composed of three things: a name, attributes, and operations.
- Class is a way to bind the data and its associated functions together

Class specification has 2 parts:

- Class Declaration.
- Class function definitions

Access Specifies:

➢ The Status of the accessibility of the data members are determined by the Access Specifies

➢ There are 3 access specifies

- Public
- Private
- Protected

Public:

It allows functions and data to be accessible to any part of the program.

Private:

It allows functions and data cannot be accessible to any part of the program except the class where it is declared.

Protected

It allows functions and data to be accessible to only the derived classes.

Class Declaration:

Syntax:

```
class class_name
{
    private:
        variable declaration;
        function declaration;
    public:
        variable declaration;
        function declaration;
}
```

Example:

```
class book
{
    int pgno;
    public:
        void getpage();
}
```

Creation of Objects:

Once the class is created, one or more objects can be created from the class as objects are instance of the class.

Just as we declare a variable of data type int as:

int x;

Objects are also declared as:

class_name followed_by object_name;

Example:

exforsys e1;

This declares e1 to be an object of class exforsys.

Accessing Class Members:

Creating Object:

Syntax:

classname object_name;

Example:

book i;

Accessing Methods:

Syntax:

object.function_name(argument)

Example:

i.getpage();

Defining member functions

Defining a Member

- Definition in 2 places
- Outside the class definition.
- Inside the class definition.

Outside the Class Definition

Syntax:

```
return_type class_name :: function_name(argument list)
{
        - - - - - - - - - -
        body of function
```

```
            - - - - - - - - - -
    }
```

return_type Type of value function will return.

class_name:: A program may contain more than one class and these classes may have similar member functions. Class_name:: tells the compiler which class the function belongs to and the scope of the member function is restricted to the class_name.

function_name Can be any valid C++ identifier.

argument list Represents the type and number of value function will take, values are sent by the calling statement.

Example of defining member function outside class

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

class Employee
{
    int Id;
    char Name[25];
    int Age;
    long Salary;

    public:
    void GetData();
    void PutData();
```

```
};

void Employee :: GetData()          //Statement 1 : Defining GetData()
{
      cout<<"\n\tEnter Employee Id : ";
      cin>>Id;

      cout<<"\n\tEnter Employee Name : ";
      cin>>Name;

      cout<<"\n\tEnter Employee Age : ";
      cin>>Age;

      cout<<"\n\tEnter Employee Salary : ";
      cin>>Salary;
}

void Employee :: PutData()          //Statement 2 : Defining PutData()
{
      cout<<"\n\nEmployee Id : "<<Id;
      cout<<"\nEmployee Name : "<<Name;
      cout<<"\nEmployee Age : "<<Age;
      cout<<"\nEmployee Salary : "<<Salary;
}

void main()
{
```

```
        Employee E;          //Statement 3 : Creating Object

        E.GetData();         //Statement 4 : Calling GetData()
        E.PutData();         //Statement 5 : Calling PutData()

    }
```

Output :

```
        Enter Employee Id : 1
        Enter Employee Name : Kumar
        Enter Employee Age : 29
        Enter Employee Salary : 45000

        Employee Id : 1
        Employee Name : Kumar
        Employee Age : 29
        Employee Salary : 45000
```

Inside the Class Definition:

Syntax:

```
return_type function_name(argument list)
{
        - - - - - - - - - -
        body of function
        - - - - - - - - - -
}
```

return_type Type of value function will return.

function_name Can be any valid C++ identifier.

argument list Represents the type and number of value function will take, values are sent by the calling statement.

Definition of member function inside class is similar to defining normal function. There is no need to tell compiler about the class the function belongs to because the definition of member function is already in the class.

Example of defining member function inside class

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

class Employee
{
      int Id;
      char Name[25];
      int Age;
      long Salary;

      public:
      void GetData()          //Statement 1 : Defining GetData()
      {
           cout<<"\n\tEnter Employee Id : ";
           cin>>Id;

           cout<<"\n\tEnter Employee Name : ";
           cin>>Name;
```

```
            cout<<"\n\tEnter Employee Age : ";
            cin>>Age;

            cout<<"\n\tEnter Employee Salary : ";
            cin>>Salary;
        }

        void PutData()          //Statement 2 : Defining PutData()
        {
            cout<<"\n\nEmployee Id : "<<Id;
            cout<<"\nEmployee Name : "<<Name;
            cout<<"\nEmployee Age : "<<Age;
            cout<<"\nEmployee Salary : "<<Salary;
        }

    };


    void main()
    {

        Employee E;          //Statement 3 : Creating Object

        E.GetData();         //Statement 4 : Calling GetData()
        E.PutData();         //Statement 5 : Calling PutData()

    }
```

Output :

```
Enter Employee Id : 1
Enter Employee Name : Kumar
Enter Employee Age : 29
Enter Employee Salary : 45000

Employee Id : 1
Employee Name : Kumar
Employee Age : 29
Employee Salary : 45000
```

Static data members

Static Variables (Static Data members):

- By default it is initialized to zero
- Only one copy of that variable is created for entire class
- It is visible only within the class, but lifetime is the entire program
- static is the keyword used to declare static data members

Syntax:

static datatype variable=value

Example:

```
static int i=5;
```

Program:

```
#include<iostream.h>
void main()
{
  static int i=3;
  while(i)
```

```
  {
    cout<<i<<"\n";
    i--;
    main();
  }
}
```

Static member functions

Static Member Functions:

- A static function can have access to only other static members declared in the class
- A static member function can be called using the class name

Syntax:

classname::functionname

Program:

```
#include<iostream.h>
class test
{
  int code;
  static int cn;
  public:
    void set()
    {
      code=++cn;
    }
    void showcode()
    {
      cout<<"\nObject number:"<<code<<"\n";
    }
```

```
    static void showcount()
    {
      cout<<"\ncount:"<<cn;
    }
};
int test::cn;
void main()
{
  test t1,t2;
  t1.set();
  t2.set();
  test::showcount();
  test t3;
  t3.set();
  test::showcount();
  t1.showcode();
  t2.showcode();
  t3.showcode();
}
```

Array of objects

- An object of class represents a single record in memory,
- an array is a collection of similar type, therefore an array can be a collection of class type..

Syntax:

Class name object[size];

Example:

Item i[50];

Program:

```
#include<iostream.h>
class employee
{
   char name[30];
   float age;
   public:
     void get();
     void put();
};
void employee::get()
{
  cout<<"Enter name:";
  cin>>name;
  cout<<"Enter age:";
  cin>>age;
}
void employee::put()
{
  cout<<"Name:"<<name<<"\n";
  cout<<"Age:"<<age<<"\n";
}
void main()
{
  employee e[50];
  int n;
  cout<<"Enter the No of Employees:";
  cin>>n;
  cout<<"Enter the Details:";
  for(int i=0;i<n;i++)
```

```
    e[i].get();
  cout<<"\nDetails of"<<n<<"Employees";
  for(i=0;i<n;i++)
  {
    cout<<"\nEmployee"<<i+1<<"\n";
    e[i].put();
  }
}
```

Friendly functions

Private members are accessed only within the class they are declared. Friend function is used to access the private and protected members of different classes. It works as bridge between classes.

- Friend function must be declared with **friend** keyword.
- Friend function must be declare in all the classes from which we need to access private or protected members.
- Friend function will be defined outside the class without specifying the class name.
- Friend function will be invoked like normal function, without any object.

Syntax:

friend returntype functionname(Arg list)

Program:

```
#include<iostream.h>
class second;
class first
{
```

```
  int a;
  public:
    void set(int x)
    {
      a=x;
    }
    friend void max(first,second);
};
class second
{
  int b;
  public :
    void set(int y)
    {
      b=y;
    }
    friend void max(first,second);
};
void max(first m,second n)
{
  if(m.a>=n.b)
    cout<<m.a;
  else
    cout<<n.b;
}
void main()
{
  first f;
  f.set(10);
```

```
  second s;
  s.set(20);
  max(f,s);
}
```

Constructors and destructors: Constructors

Constructors:

Constructor is a special function used to initialize class data members or we can say constructor is used to initialize the object of class.

- Constructor name class name must be same.
- Constructor doesn't return value.
- Constructor is invoked automatically, when the object of class is created.

Characteristics

- Should be in the public section
- Invoked automatically
- Do not have any return value
- Cannot be inherited
- Can have arguments
- Cannot be virtual
- Cannot refer to their address
- Make implicit calls to the operators new and delete.

Types of Constructor

- Default Constructor
- Parameterize Constructor
- Copy Constructor

Construct without parameter is called default constructor.

Example of C++ default constructor

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

class Student
{
                                        int Roll;
                                        char Name[25];
                                        float Marks;

                                        public:

                                        Student()          //Default Constructor
                                        {
                                                Roll = 1;
                                                strcpy(Name,"Kumar");
                                                Marks = 78.42;
                                        }

                                        void Display()
                                        {
                                                cout<<"\n\tRoll : "<<Roll;
                                                cout<<"\n\tName : "<<Name;
                                                cout<<"\n\tMarks : "<<Marks;
                                        }
};


void main()
{
```

```
                                        Student S;        //Creating Object

                                        S.Display();      //Displaying Student
Details

        }


        Output :

                                Roll : 1
                                Name : Kumar
                                Marks : 78.42
```

Construct without parameter is called default constructor.

Example of C++ default constructor

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

        class Student
        {

                                        int Roll;
                                        char Name[25];
                                        float Marks;

                                        public:
```

```
                                    Student()          //Default
Constructor
                                    {
                                            Roll = 1;
                                            strcpy(Name,"Kumar");
                                            Marks = 78.42;
                                    }

                                    void Display()
                                    {
                                            cout<<"\n\tRoll : "<<Roll;
                                            cout<<"\n\tName : "<<Name;
                                            cout<<"\n\tMarks : "<<Marks;
                                    }
        };


        void main()
        {

                                    Student S;         //Creating Object

                                    S.Display();       //Displaying
Student Details

        }
```

Output :

Roll : 1
Name : Kumar
Marks : 78.42

Parameterized Constructors

- Parameters are arguments to the Constructor
- This can be done in 2 ways
- By calling the Constructor explicitly
- Class-name obj=constructorname(arg list);
- By calling the Constructor implicitly
- Class-name obj(arg list);

Program:

```
#include<iostream.h>
class assign
{
  int a,b;
  public:
    assign(int x,int y)
    {
      a=x;
      b=y;
    }
    void show()
    {
      cout<<"\nNumber1:"<<a;
      cout<<"\nNumber2:"<<b;
    }
```

```
};
void main()
{
  assign f(34,35);
  f.show();
  assign s=assign(75,76);
  s.show();
}
```

Copy Constructor

- A reference variable is used as an argument to copy constructor
- Constructor contains the address value of another object or a variable as its argument.

Program

```
#include<iostream.h>
class copy
{
  int y;
  public:
     copy()
     {
                cout<<"\nNo Arguments";
      }
      copy(int i)
      {
                 y=i;
       }
copy(copy &x)
       {
```

```
                y=x.y;
        }
        void display()
        {
                cout<<"\nValues:"<<y;
        }
};
void main()
{
 copy a(111);
 copy b(a);
 copy c=a;
 copy d;  d=a;
 cout<<"\nValue of a:";
 a.display();
 cout<<"\nValue of b:";
 b.display();
 cout<<"\nValue of c:";
 c.display();
 cout<<"\nValue of d:";
 d.display();
}
```

Multiple constructors in a class

- More than one Constructor with in a class is called Multiple Constructor
- It is also known as Constructor Overloading

Program

```
#include<iostream.h>
class Method
{
  public:
     Method()
     {
                cout<<"\nNo Arguments";
      }
      Method(int i)
      {
                cout<<"\nInteger Argument:"<<i;
       }
       Method(double i)
       {
                 cout<<"\nDouble Argument:"<<i;
       }


       Method(char i)
       {
                 cout<<"\nCharacter Argument:"<<i;
       }
};
void main()
{
 Method b;
 Method b1(5);
 Method b2(6.5);
 Method b3('c');
```

```
}
```

Dynamic Constructor

Basically, it's a way of constructing an object based on the run-time type of some existing object. It basically uses standard virtual functions/polymorphism.

```
class base
{
public:
virtual base* create() = 0;
virtual base* clone() = 0;
protected:
base();
base(const base&);
};
virtual der1 : public base
{
public:
base* create() { return new der1; }
base* clone() { return new der1(*this); }
};
virtual der2 : public base
{
public:
base* create() { return new der2; }
base* clone() { return new der2(*this); }
};
int main()
{
base* b = new der1;
```

```
base* b1 = b->create();
base* b2 = b->clone();
}
```

Destructor

Constructor allocates the memory for an object. Destructor deallocate the memory occupied by an object. Like constructor, destructor name and class name must be same, preceded by a tilde(~) sign. Destructor take no argument and have no return value.

Constructor is invoked automatically when the object created. Destructor is invoked when the object goes out of scope. In other words, Destructor is invoked, when compiler comes out form the function where an object is created.

- Destructor destroys the objects that have been created by a constructor
- It is also a special member function
- Its also same name as the class name preceded by a tilde

Syntax:

```
~class-name();
```

- It never takes any arguments and have no return value
- Automatically invoked by compiler at the end of program
- delete is used for free memory

Program:

```
#include<iostream.h>
int count=0;
class copy
{
```

```
  int y;
  public:
     copy()
     {
                count++;
                cout<<"\nNo Objects Created:"<<count;
      }
      ~copy()
      {         cout<<"\nNo Objects Deleted:"<<count;
                count--;
       }
};
void main()
{
 copy a,b,c,d;
}
```

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT: : OBJECT ORIETNED PROGRAMMING WITH C++
SEMESTER : I
SUBJECT CODE: 17CCP104 CLASS : I M.COM CA
UNIT 3

S.NO	QUESTIONS	OPTION 1	OPTION 2	OPTION 3	OPTION 4	ANSWER
1	Unary operators require _____Operand	One	Two	Three	Four	One
2	Binary Operator requires _____Operand	One	Two	Three	Four	Two
3	_____ functions may be used in place of member functions for overloading a binary operator.	Inline	Member	Conversion	Friend	Friend
4	The operator that cannot be overloaded is _____	Single of	+	-	=	Single of
5	The duplication of inherited members due to these multiple paths can be avoided by making one common base class as _____	Virtual Base Class	Abstract Class	derived class	none of these	Virtual Base Class
6	_____ is called compile time polymorphism.	Operator overloading	Function overloading	Overloading unary operator	Overloading binary operator	Operator overloading
7	_____ feature can be used to add two user-defined operator data types.	Function	Overloading	Arrays	Pointers	Overloading
8	The mechanism of giving special meaning to an operator is called as _____Overloading	Operator	Function	Value	Constants	Operator
9	Operator overloading is done with the help of a special function called _____ function.	Conversion	Operator	User-defined	In-built.	Operator
10	_____ functions must either be member functions or friend functions.	Operator	User-defined	Static Member	Overloading	Operator
11	The overloading operator must have atleast _____ operand that is of user-defined data type.	Two	Three	One	Four	One
12	_____ operator function should be a class member.	Arithmetic	Relational	Casting	Overloading	Casting
13	The casting operator must not have any _____	Arguments	Member	Return type	Operator	Arguments
14	The casting operator function must not specify a _____ type.	User-defined type	Return	Member	In-built	Return
15	The operator that cannot be overloaded is _____.	Casting	Binary	Unary	Scope resolution	Scope resolution
16	The friend function cannot be used to overload _____ operator.	+	-	)	::	)
17	_____ operator cannot be overloaded by friend function.	[]	*	.	?:	[]
18	The operator that cannot be overloaded by friend function is _____	.	::	->	Single of	.
19	Operator overloading is called _____	Function Overloading	Compile time polymorphism	Casting operator function	Temporary object	Compile time polymorphism
20	Overloading feature can add two _____ data types.	In-built	Enumerated	User-defined	Static	User-defined
21	The mechanism of deriving a new class from an old one is called	Operator overloading	Inheritance	Polymorphism	Access mechanism	Inheritance
22	_____ provides the concept of reusability.	Overloading	Message passing	Data abstraction	Inheritance	Inheritance
23	C++ can be reused using	Inheritance	Encapsulation	Polymorphism	Overloading	Inheritance
24	Inheritance provides the concept of _____.	Derived class	Subclass	Virtual base class	Reusability	Reusability
25	The _____ class inherits some or all of the properties of base class.	Abstract class	Father class	Derived class	Child class	Derived class
26	A derived class with only one base class is called _____ inheritance.	Single	Multi-level	Multiple	Hierarchical	Single
27	The derived class inherits some or all of the properties of _____ class.	Member	Base	Father	Ancestor	Base
28	A derived class can have only one _____ class.	Derived	Base	Child	Member	Base
29	_____ class inherits some or all of the properties of base class.	Base	Virtual base	Subclass	Derived	Derived
30	A class that inherits properties from more than one class is known as _____ inheritance.	Multiple	Multilevel	Single	Hybrid	Multiple
31	The class that can be derived from another derived class is known as _____ inheritance.	Hierarchical	Single	Multi-level	Hybrid	Multi-level
32	When the properties of one class are inherited by more than one class, it is called _____ inheritance.	Single	Hybrid	Multiple	Hierarchical	Hierarchical
33	A _____ can inherit properties from more than one class.	Class	Member class	Inheritance	Base class	Class
34	A class can be derived from another _____ class.	Member	Common base	Derived	Indirect base class	Derived
35	_____ of one class can be inherited by more than class.	Functions	Properties	Friend	Subclass	Properties
36	A private member of a class cannot be inherited either in public mode or in _____ mode.	Private	Protected	Visibility	Nesting	Private
37	A protected member inherited in public mode becomes _____	Highly protected	Private	Public	Protected	Protected
38	A protected member inherited in private mode becomes _____	Visibility	Private	Protected	Public	Private
39	A _____ member of a class cannot be inherited in public mode.	Public	Protected	Private	Access	Private
40	A member inherited in public mode becomes _____ in the derived class.	Private	Class	Public	Protected	Protected
41	A protected member inherited in _____ mode becomes private in the derived class.	Protected	Visibility	Private	Public	Private
42	A public member inherited in _____ mode becomes public.	Private	Public	Visibility	Protected	Public
43	A public member inherited in private mode becomes _____	Private	Public	Protected	Visibility	Private
44	The _____ functions of a friend class can directly access the private and protected data.	Inline	Friend	Virtual	Static members	Friend
45	A _____ member inherited in public mode becomes public in the derived class.	Protected	Private	Static	Public	Public
46	A public member inherited in _____ mode become private in the derived class.	Visibility	Private	Protected	Public	Private
47	The private and protected class can directly access the _____ functions of a friend class.	Virtual	Inline	Member	Static member	Member
48	The member functions of a _____ class can directly access only the protected and public data.	Indirect Base	Ancestor	Base	Derived	Derived
49	The member functions of a _____ class can access the private data.	Base	Derived	Ancestor	Indirect base	Base
50	_____ inheritance may lead to duplication of inherited members from a 'grand parent' base class.	Multiple	Multipath	Hybrid	Single	Multipath
51	Duplication of inherited members of _____ inheritance can be avoided by making the common base class, a virtual base class.	Single	Multi-level	Multipath	Hierarchical	Multipath
52	In _____ inheritance, the base classes are constructed in the order in which they appear in the declaration of the derived class.	Hybrid	Multipath	Hierarchical	Multiple	Multiple
53	In multi-level inheritance, the _____ are executed in the order of inheritance.	Derivations	Constructors	Destructors	Containership	Constructors
54	A class that contains objects of other classes is known as _____.	Indirect base class	Nesting	Subclass	Inheritance	Nesting
55	The _____ section of constructor function is used to assign initial values to its data members.	Initialization	Declaration	Argument	Assignment	Assignment

56	The grand parent class is sometimes referred to as _____class.	Ancestor	Virtual base	Indirect base	Direct base	Indirect base
57	_____ may arise in single inheritance application.	Ambiguity	Visibility	Nesting	Derivation	Ambiguity
58	A _____ that contains objects of other classes is known as containership.	Function	Friend	Class	Subclass	Class
59	A member declared as _____ cannot be accessed by the function outside the class.	Private	Protected	Public	Visibility	Protected
60	The public member of a class can be accessed by its own objects using the _____ operator.	Scope resolution	Relational	Arithmetic	Dot	Dot

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 21
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT : OBJECT ORIENTED PROGRAMMING WITH C++
SEMESTER : I
SUBJECT CODE: 17CCP104 CLASS : I M.COM CA

POSSIBLE QUESTIONS – UNIT III

PART A (1 Mark)

(Online Examinations)

PART B (2 Marks)

1. Define Class.
2. Define Object.
3. How will you Specify a class
4. How will you define member functions
5. What is pointers to Members
6. Define Constructor
7. Define Destructor
8. How many types of Constructor
9. Define Static data members
10. What is Static Member Function
11. What is Nesting of Member Functions
12. What is Jump Students
13. What is Goto Statements

PART C (6 Marks)

1. Explain Member functions in a class
2. Explain Static Data members with example.
3. Explain Static Member functions with example
4. Describe Constructor and Destructor
5. Discuss Private Member Functions
6. Write a program to subtract two numbers (member functions should be define outside the class)
7. Elucidate dynamic destructor with example.
8. Explain member function and nesting of member function
9. What is constructor? Explain the types of constructor with an suitable example
10. Engrave on friend function with suitable example.

Enable | Enlighten | Enrich
(Deemed to be University)
(Under Section 3 of UGC Act 1956)

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT : **OBJECT ORIENTED PROGRAMMING WITH C++**
SEMESTER : I
SUBJECT CODE: **17CCP104** **CLASS** : **I M.COM CA**

UNIT IV

Operator Overloading – Type Conversions – Introduction – Defining Operator Overloading– Manipulation of String using Operators – Rules for Overloading Operators- Types Conversions. Inheritance – Abstract Classes.

Unit – IV

Operator overloading: Defining operator overloading

- The process of giving special meaning to a method or an operator is called Operator Overloading
- Overloading is the process of adding an extra or additional operation to an existing operation
- Overloading consist of same name but differ in their argument list, Number of argument or both.
- There are two types of overloading
  - Function overloading
  - Operator overloading

Method Overloading

- Change the meaning of a function
- The name of the function is same but differ in their operation differ in their arguments list
- Function overloading is done by using various number arguments to a function
- Function perform different operation based on the requirements

Program:

```
#include<iostream.h>
class over
{
  public:
    void add(int a,int b)
    {
                cout<<"\nAddition of integer:"<<a+b;
     }
    void add(double a,double b)
```

```
    {
            cout<<"\nAddition of double:"<<a+b;
     }

    void add(int a,double b)
    {
            cout<<"\nAddition of integer & double:"<<a+b;
     }
void add(double a,int b)
    {
            cout<<"\nAddition of double and integer:"<<a+b;
     }
     void add(int a)
    {
            cout<<"\nOne Argument:"<<a;
     }
};
void main()
{
   over b;
   b.add(5,6);
   b.add(8.2,7.8);
   b.add(7,8.3);
   b.add(8.3,7);
   b.add(111);
}
```

Operator Overloading

- Mechanism of giving special meaning to an operator
- It creates a new definition for most c++ operators

- Semantics of an operator is extended
- It does not change the meaning of the operator

Rules for Overloading Operators:

1. Only existing operators can be overloaded. New operators cannot be created

2. The overloaded operator must have at least one operand that is of user-defined type

3. Can not be able to change the predefined meaning of the Operator.

4. An overloaded operator follows the syntax rules of the original operators. They can not be overridden

5. Some Operators that can not be overloaded.

6. Certain Operators can not be overloaded using the friend Function.

Operators Cannot be Overloaded

- Membership operators (.)
- Pointer-to-member operator (.*)
- Scope resolution operator (::)
- Size of operator (sizeof)
- Conditional operator (?:)

Operators Cannot be Overloaded Using friend Function

- Assignment operator (=)
- Function call operator (())
- Subscripting operator ([])
- Class member access operator (->)

Defining Operator Overloading

- Done with the help of a special function, *operator function,* which describes the task

Syntax:

- **Declaration:**

  RT operator operatorsymbol(argument list)

- **Definition:**

```
RT classname :: operator(op-arglist)
{
   function body
}
```

Example:

```
void operator –()
```

- Operator function must be either member functions or friend function
- Difference: a friend function will have only 1 argument for unary operators and 2 arguments for binary operator

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

Example

```
#include<iostream.h>
class Add
{
  int lat,log;
  public:
    Add(){}
    Add(int l,int lt)
    {
       lat=l;
```

```
      log=lt;
    }
    void show()
    {
      cout<<lat<<" ";
      cout<<log<<" ";
    }
    Add operator -(Add o);
};
Add Add::operator -(Add o)
{
  Add t;
  t.lat=o.lat+lat;
  t.log=o.log+log;
  return t;
}

void main()
{
  Add a(10,20),b(30,50);
  a.show();
  b.show();
  a=a-b;
  a.show();
}
```

Overloading unary operators

Overloading Unary Operators:

- The operator has only one Operand.

- Unary operators are unary +, unary –,++,--, this operator changes the sign of the operand.

Program

```
#include<iostreams.h>
class space
{
  int x;
  int y;
  int z;
  public:
    void get(int a,int b,int c);
    void display(void);
    void operator -();
};
void space::get(int a,int b,int c)
{
  x=a;
  y=b;
  z=c;
}
void space::display(void)
{
  cout<<x<<"\n";
  cout<<y<<"\n";
  cout<<z<<"\n";
}
void space::operator -()
{
  x=-x;
```

```
  y=-y;
  z=-z;
}
void main()
{
  space s;
  s.getdata(10,-20,30);
  cout<<"\nValues before Call Operator Overloading\n";
  s.display();
  -s;
  cout<<"\nValues After Call Operator Overloading\n";
  s.display();
}
```

Overloading binary operators

- The operator has two Operand.

Program:

```
#include<iostreams.h>
class Time
{
  int h,m;
  public:
    Time(){}

    Time(int hr,int min)
    {
       h=hr;
       m=min;
    }
    void display(void);
```

```
    Time operator+(Time);
};
void Time::display(void)
{
  cout<<h<<"hours and"<<m<<" Min\n";
}
Time Time::operator+(Time t)
{
  Time t1;
  t1.m=m+t.m;
  int bal=t1.m/60;
  t1.m=t1.m%60;
  t1.h=h+t.h+bal;
  return(t1);
}
void main()
{
  Time h1,h2,h3;
  h1=Time(2,50);
  h2=Time(2,50);
  h3=h1+h2;
  cout<<"\nTime t1:";
  h1.display();
  cout<<"\nTime t2:";
   h2.display();
  cout<<"\nTime t3:";
  h3.display();
}
```

Overloading binary operators using friends

- Non member function of a class can be able to access the private members of a class through friend function
- Friend Function are created with the keyword friend
- A friend function requires two arguments to be explicitly passed to it.

Program:

```
#include <iostream.h>
#include <conio.h>
class Point
{
  int x, y;
 public:
  Point()
  {}
  Point(int px, int py)
  {
    x = px;
    y = py;
  }
  void show()
  {
    cout << x << " ";
    cout << y << "\n";
  }
  friend Point operator+(Point op1, Point op2); // now a friend
  Point operator=(Point op2);
};
```

```
// Now, + is overloaded using friend function.
Point operator+(Point op1, Point op2)
{
  Point temp;
  temp.x = op1.x + op2.x;
  temp.y = op1.y + op2.y;
  return temp;
}
// Overload assignment for Point.
Point Point::operator=(Point op2)
{
  x = op2.x;
  y = op2.y;
  return op2; // i.e., return object that generated call
}
int main()
{
  clrscr();
  Point ob1(10, 20), ob2( 5, 30);
  ob1 = ob1 + ob2;
  ob1.show();
  return 0;
}
```

type conversions

- Type Conversion is the process of change the data type of variable.
- When constants and variables of different types are mixed in an expression, C applies automatic type conversion to the operand as per certain rules.

- In the Assignment Operation the type of data to the right of an assignment operator is automatically converted to the type of the variable on the left.

Example:
```
int m;
float x=5.89;
m=x;
```

The value of m is 5 since the fraction part is truncated.

- The Compiler does not support automatic type conversion for user defined data types. Since the type conversion should be performed explicitly.
- Three types of situations arises in the data conversion between incompatible types:

❖ Conversion from basic type to class type

❖ Conversion from class type to basic type

❖ Conversion from one class type to another class type.

Conversion from basic type to class type

- In this left hand operand of = operator is always a class object.

Example:
```
String s1,s2;
char *name1="C++ Programming";
char *name2="C Programming";
s1=String(name1);
s2=name2;
```

String is the class, name1 is char variable which is converted explicitly in the statement

```
s1=String(name1);
```

name2 is char variable which is converted implicitly by call the constructor in the statement

```
s2=name2;
```

The constructor used for type conversion takes a single argument whose type is to be converted. This conversion is done by overloaded = operator.

Conversion from class type to basic type

- In this left hand operand of = operator is always a variable type or basic type.
- The constructor performs a fine job in conversion from basic to class type.
- The conversion of class to basic model is done by overloaded casting operator.
- The general form of an overloaded casting operator function is

```
operator typename()
{
    ..........
    ..........//function statement
}
```

Example:

```
String :: operator double()
{
        double d=0;
        d=s[0]+s[1];
        return(d);
}
```

String is a class converted to basic type double, where s is a String class object

```
String s1,s2;
float c1,c2;
c1=float(s1);
c2=s2;
```

c1 is float variable , String is the class which is converted explicitly in the statement

c1=float(s1);

c2 is float variable , String is the class which is converted implicitly in the statement

c2=s2;

The casting operator function should satisfy the following conditions:

- It must be a class member
- It must not specify a return type
- It must not have any arguments.

Conversion from one class type to another class type.

Example:

S1= S2 // objects of different types

- S1 and S2 are the object of two different classes class X and class Y.
- The class Y type is converted into X type.
- Y is known as the source class and X is known as the designation class.
- This type of conversion is performed using either a contractor or a conversion function
- Casting function is of the form

operator typename()

Type Conversion Table

Conversions Required	Conversion takes place in	
	Source class	Designation class
Basic →class	Not applicable	Constructor
Class→Basic	Casting operator	Not applicable
Class→class	Casting operator	Constructor

Inheritance :- Inheritance

- Sharing the properties of one class by the other.
- Ability of a new class to be created, from an existing class by extending it, is called inheritance.
- Different kinds of objects often have a certain amount in common with each other.
- Object-oriented programming allows classes to inherit commonly used state and behavior from other classes.
- A class which provides the data is called Base class
- A class receives the data is called Derived class
- No changes are made to the base class

Advantage of Inheritance:

- Reusability of code
- Save a lot of time and efforts, retyping the same
- Data and methods of super class are physically available to its subclasses

Forms of Inheritance

In C++ there are 5 forms of inheritance.

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Defining derived classes

- Derived class can be defined by specifying the relationship with the base class in addition to its own details.
- : (colon) operator is used for inheritance.

Syntax:

```
class derived-class-name : visibility-mode base-class-name
{
        ..............
        ..............  //derived class member functions
        ..............
};
```

- The colon indicates that the derived-class-name is derived from the base-class-name.
- The visibility-mode is optional, if presents private or public or protected access specifies can be specified
- By default visibility-mode is private.
- The visibility-mode specifies whether the features of the base class are privately derived or publicly derived.

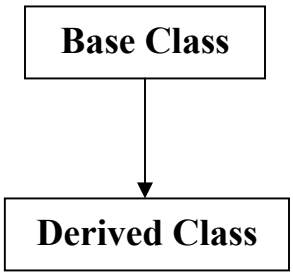
Example:

```
class ABC
{
        ..............
        ..............// base class members
}
class der : private ABC //Privately inherited from class ABC
{
        ..............
        ..............// derived class members
}
```

```
class der : public ABC //Publicly inherited from class ABC
{
            ..............
            ..............// derived class members
}
class der :  ABC //Privately inherited from class ABC by default
{
            ..............
            ..............// derived class members
}
```

Single, multilevel, multiple, hierarchical inheritance

- Single inheritance consist of single base class and single derived class



Syntax:

```
class derived-class-name : visibility-mode base-class-name
{
            ..............
```

```
            ..............// derived class members
}
```

The colon (:), indicates that the class derived-class-name is derived from the class base-class-name.

Program:

```
#include<iostream.h>
class Rectangle
{
  private:
    float length ; // This can't be inherited
  public:
    float breadth ; // The data and member functions are inheritable
  void Enter_lb(void)
  {
     cout << "\n Enter the length of the rectangle : ";
     cin >> length ;
     cout << "\n Enter the breadth of the rectangle : ";
     cin >> breadth ;
  }
  float Enter_l(void)
  {
     return length ;
  }
}; // End of the class definition
class Rectangle1 : public Rectangle
{
   private:
     float area ;
```

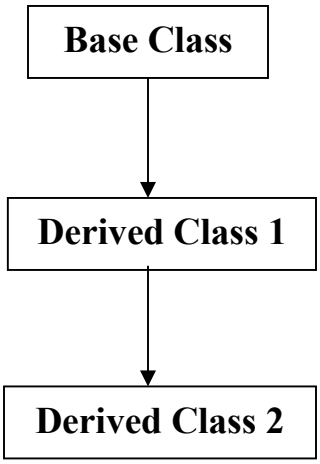
```
    public:
      void Rec_area(void)
      {
        area = Enter_l( ) * breadth ;
      }
      void Display(void)
      {
        cout << "\n Length = " << Enter_l( ) ;
        cout << "\n Breadth = " << breadth ;
        cout << "\n Area = " << area ;
       }
};
void main(void)
{
  Rectangle1 r1 ;
  r1.Enter_lb( );
  r1.Rec_area( );
  r1.Display( );
}
```

Visibility of Inherited Members

Base class visibility	**Derived class visibility**		
	public derivation	**private derivation**	**protected derivation**
private	Not Inherited	Not Inherited	Not Inherited
protected	protected	private	protected
public	public	private	protected

Multilevel Inheritance:

- C++ also provides the facility of multilevel inheritance, according to which the derived class can also be derived by another class, which in turn can further be inherited by another and so on.



Syntax:

```
class derived-class-name1 : visibility-mode base-class-name
{
        ..............
        ..............// derived class members
}
class derived-class-name2 : visibility-mode derived-class-name1
{
        ..............
        ..............// derived class members
}
```

derived-class-name1 is inherited from base-class-name then the derived-class-name2 is inherited from derived-class-name1.

Program:

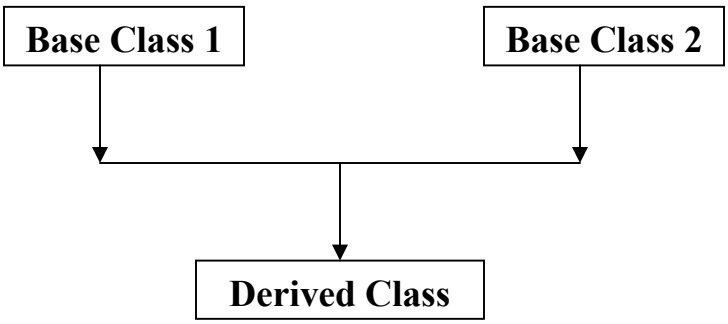
```
#include<iostream.h>
class Base
{
   protected:
      int b;
   public:
      void EnterData( )
      {
         cout << "\n Enter the value of b: ";
         cin >> b;
      }
      void DisplayData( )
      {
        cout << "\n b = " << b;
      }
};
class Derive1 : public Base
{
    protected:
      int d1;
    public:
      void EnterData( )
      {
         Base:: EnterData( );
         cout << "\n Enter the value of d1: ";
         cin >> d1;
```

```
    }
    void DisplayData( )
    {
       Base::DisplayData();
       cout << "\n d1 = " << d1;
    }
};
class Derive2 : public Derive1
{
   private:
     int d2;
   public:
     void EnterData( )
     {
        Derive1::EnterData( );
        cout << "\n Enter the value of d2: ";  cin >> d2;
     }
     void DisplayData( )
     {
        Derive1::DisplayData( );

             cout << "\n d2 = " << d2;
     }
};
int main( )
{
  Derive2 objd2;
  objd2.EnterData( );
  objd2.DisplayData( );
```

```
  return 0;
}
```

Multiple Inheritance

When a class is inherited from more than one base class, it is known as multiple inheritance.



Syntax:

class derived-class-name : visibility-mode base-class-name1, visibility-mode base-class-name2

{

…….

…….// derived class members

}

derived-class-name is derived from two base classes namely base-class-name1 and base-class-name1

Program:

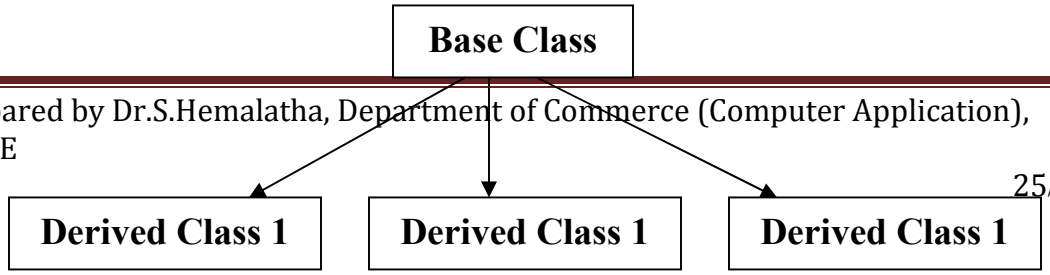
```
#include<iostream.h>
class Circle // First base class
{
  protected:
    float radius ;
  public:
```

```
    void Enter_r(void)
    {
      cout << "\n\t Enter the radius: "; cin >> radius ;
    }
    void Display_ca(void)
    {
         cout << "\t The area = " << (22/7 * radius*radius) ;
    }
};
class Rectangle // Second base class
{
   protected:
   float length, breadth ;
   public:
         void Enter_lb(void)
         {
           cout << "\t Enter the length : ";
           cin >> length ;
           cout <<"\t Enter the breadth : " ;
           cin >> breadth ;
         }
         void Display_ar(void)
         {
            cout << "\t The area = " << (length * breadth);
         }
};
class Cylinder : public Circle, public Rectangle // Derived class,
inherited
{ // from classes Circle & Rectangle
```

```
   public:
        void volume_cy(void)
        {
          cout << "\t The volume of the cylinder is: "<< (22/7*
radius*radius*length) ;
        }
};
void main(void)
{
   Circle c ;
   cout << "\n Getting the radius of the circle\n" ;
   c.Enter_r( );
   c.Display_ca( );
   Rectangle r ;
   cout << "\n\n Getting the length and breadth of the rectangle\n\n";
   r.Enter_lb( );
   r.Display_ar( );
   Cylinder cy ;
   cout << "\n\n Getting the height and radius of the cylinder\n";
   cy.Enter_r( );
   cy.Enter_lb( );
   cy.volume_cy( );
}
```

Hierarchical Inheritance:

- When two or more classes are derived from a single base class, then Inheritance is called the hierarchical inheritance.
- In this type there exists a hierarchical relation in the inheritance.



Syntax:

```
class derived-class-name1 : visibility-mode base-class-name
{
            ..............
            ..............// derived class members
}
class derived-class-name2 : visibility-mode base-class-name
{
            ..............
            ..............// derived class members
}
```

derived-class-name1, derived-class-name2 are two derived class derived from the class base-class-name.

Example:

```
#include<iostream.h>
const int len = 20 ;
class student
{
   private:
      char F_name[len] , L_name[len] ;
      int age,roll_no ;
   public:
      void Enter_data(void)
      {
         cout << "\n\t Enter the first name: " ; cin >> F_name ;
         cout << "\t Enter the second name: "; cin >> L_name ;
```

```
        cout << "\t Enter the age: " ; cin >> age ;
        cout << "\t Enter the roll_no: " ; cin >> roll_no ;
    }
    void Display_data(void)
    {
        cout << "\n\t First Name = " << F_name ;
        cout << "\n\t Last Name = " << L_name ;
        cout << "\n\t Age = " << age ;
        cout << "\n\t Roll Number = " << roll_no ;
    }
};
class arts : public student
{
  private:
    char asub1[len] ;
    char asub2[len] ;
    char asub3[len] ;
  public:
    void Enter_data(void)
    {
        student :: Enter_data( );
        cout << "\t Enter the subject1 of the arts student: ";
        cin >> asub1 ;
        cout << "\t Enter the subject2 of the arts student: ";
        cin >> asub2 ;
        cout << "\t Enter the subject3 of the arts student: ";
        cin >> asub3 ;
    }
    void Display_data(void)
```

```
    {
        student :: Display_data( );
        cout << "\n\t Subject1 of the arts student = " << asub1 ;
        cout << "\n\t Subject2 of the arts student = " << asub2 ;
        cout << "\n\t Subject3 of the arts student = " << asub3 ;
    }
};
class science : public student
{
   private:
     char csub1[len], csub2[len], csub3[len] ;
   public:
     void Enter_data(void)
     {
        student :: Enter_data( );
        cout << "\t Enter the subject1 of the science student: ";
        cin >> csub1;
        cout << "\t Enter the subject2 of the science student: ";
        cin >> csub2 ;
        cout << "\t Enter the subject3 of the science student: ";
        cin >> csub3 ;
     }
     void Display_data(void)
     {
         student :: Display_data( );
         cout << "\n\t Subject1 of the science student = " << csub1 ;
         cout << "\n\t Subject2 of the science student = " << csub2 ;
         cout << "\n\t Subject3 of the science student = " << csub3 ;
     }
```

```
};
void main(void)
{
   arts a ;
   cout << "\n Entering details of the arts student\n" ;
   a.Enter_data( );
   cout << "\n Displaying the details of the arts student\n" ;
   a.Display_data( );
   science s ;
   cout << "\n\n Entering details of the science student\n" ;
   s.Enter_data( );
   cout << "\n Displaying the details of the science student\n" ;
   s.Display_data( );
}
```

Hybrid inheritance

- Combination of multiple and multilevel inheritance is called hybrid inheritance.

Syntax:

```
class derived-class-name1 : visibility-mode base-class-name
{
            ……..
            ……..// derived class members
}
class derived-class-name2 : visibility-mode base-class-name
{
            ……..
            ……..// derived class members
```

```
}
class derived-class-name3 : visibility-mode derived-class-name1, visibility-mode derived-class-name2
{
            ..............
            ..............// derived class members
}
```

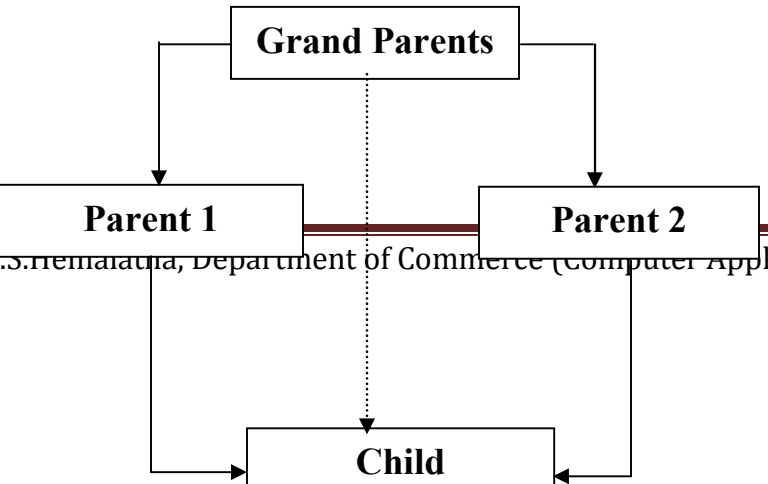
Example:

```
#include<iostream.h>
#include<conio.h>
class stu
{
  protected:
      int rno;
  public:
      void get_no(int a)
      {
        rno=a;
      }
      void put_no(void)
      {
        cout<<"Roll no"<<rno<<"\n";
      }
};
class test:public stu
{
   protected:
      float part1,part2;
   public:
```

```
        void get_mark(float x,float y)
        {
          part1=x;
          part2=y;
        }
        void put_marks()
        {
           cout<<"Marks
obtained:"<<"part1="<<part1<<"\n"<<"part2="<<part2<<"\n";
        }
};
class sports
{
    protected:
        float score;
    public:
      void getscore(float s)
        {
           score=s;
        }
        void putscore(void)
        {
           cout<<"sports:"<<score<<"\n";
        }
};
class result: public test, public sports
{
    float total;
    public:
```

```
        void display(void);
};
void result::display(void)
{
   total=part1+part2+score;
   put_no();
   put_marks();
   putscore();
   cout<<"Total Score="<<total<<"\n";
}
int main()
{
  clrscr();
  result stu;
  stu.get_no(111);
  stu.get_mark(27.5,33.0);
  stu.getscore(10.0);
  stu.display();
  return 0;
}
```

Virtual base classes



- In some situations which requires the use of both multiple and multilevel inheritance
- Consider a situation where all three kinds if inheritance, namely multiple, multilevel and hierarchical inheritance are involved.
- In the above figure 'Child' has two base classes 'Parent1' and 'Parent2' which themselves have common base class 'Grand Parents'.
- The 'Child' inherits the traits of 'Grand Parent' via two separate paths.
- It can also inherit directly as shown by broken line.
- The 'Grand Parents' is sometimes referred as indirect base class.
- In the above case there exist a problem all the public and protected members of 'Grand Parents' are inherited into 'Child' twice, first via 'Parent 1' and again via 'Parent 2'. This introduces ambiguity and should be avoided.
- The duplication of inherited members due to these multiple paths can be avoided by making the common base class as virtual base class while declaring the direct or intermediate base class.

Syntax:

```
class base-class-name
{
            ……..
            ……..// base class members Grand Parents
```

```
}
  class derived-class-name1 : virtual visibility-mode base-class-name
{
            ..............
            ..............// derived class members Parent1
}
class derived-class-name2 : visibility-mode virtual base-class-name
{
            ..............
            ..............// derived class members Parent2
}
class derived-class-name3 : visibility-mode derived-class-name1,
visibility-mode derived-class-name2
{
            ..............
            ..............// derived class members Child
}
```

- When a class is made 'virtual' base class, c++ takes necessary care to see that only one copy of that class is inherited, regardless of how many inheritance paths exist between the virtual base class and a derived class.

Program:

```
#include<iostream.h>
#include<conio.h>
class stu
{
  protected:
      int rno;
  public:
```

```
        void get_no(int a)
        {
          rno=a;
        }
        void put_no(void)
        {
          cout<<"Roll no"<<rno<<"\n";
        }
};
class test:virtual public stu//Virtually inherited
{
   protected:
        float part1,part2;
   public:
        void get_mark(float x,float y)
        {
          part1=x;
          part2=y;
        }
        void put_marks()
        {
           cout<<"Marks
obtained:\npart1="<<part1<<"\n"<<"part2="<<part2<<"\n";
        }
};
class sports: public virtual stu
{
   protected:
        float score;
```

```
    public:
       void getscore(float s)
         {
            score=s;
         }
         void putscore(void)
         {
            cout<<"sports:"<<score<<"\n";
         }
};
class result: public test, public sports
{
    float total;
    public:
         void display(void);
};
void result::display(void)
{
   total=part1+part2+score;
   put_no();
   put_marks();
   putscore();
   cout<<"Total Score="<<total<<"\n";
}
int main()
{
  clrscr();
  result stu;
  stu.get_no(123);
```

```
  stu.get_mark(27.5,33.0);
  stu.getscore(6.0);
  stu.display();
  return 0;
}
```

Abstract classes

- abstract keyword is used to create abstract class.
- An abstract class is one that is not used to create object
- An abstract class is designed only to act as a base class.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT: : OBJECT ORIETNED PROGRAMMING WITH C++
SEMESTER :
SUBJECT CODE: 17CCP104 CLASS : I M.COM CA
UNIT 4

S.NO	QUESTIONS	OPTION 1	OPTION 2	OPTION 3	OPTION 4	ANSWER
1	_____ is the process of using the same name for two or more functions	Function Overloading	Operator Overloading	Default Function	Constructors	Function Overloading
2	The library function isalpha( ) requires the header file _____	<ctype.h>	<math.h>	<time.h>	<stdlib.h>	<ctype.h>
3	Which of the following language feature is not an access specifier in C++?	Public	Private	Protected	Internal	Internal
4	Which of the following is not a type of inheritance?	Multiple	Multilevel	Distributive	Hierarchical	Distributive
5	How many instances of an abstract class can be created?	1	5	3	0	0
6	Which of the following concepts provides facility of using object of one class inside another class?	Encapsulation	Inheritance	Abstraction	Composition	Inheritance
7	For a binary member operator function, the left operand is passed through _____	pointer	reference	this	parameter	this
8	A class that is inherited is called as _____ class	derived	child	base	abstract	base
9	Which of the following type of class allows only one object of it to be created?	Virtual class	Abstract class	Singleton class	Friend class	Singleton class
10	Which of the following correctly describes overloading of functions?	Virtual polymorphism	Transient polymorphism	Ad-hoc polymorphism	Pseudo polymorphism	Ad-hoc polymorphism
11	The scope resolution operator is _____	a comma	a semicolon	a colon	two colons	two colons
12	The feature that allows you to use the same function name for separate functions that have different argument lists is called _____	overriding	overloading	constructing	destructing	overloading
13	Inheritance occurs when a class adopts all the traits of _____	an object	a parent class	a variable	a function	a parent class
14	Template classes that have already been written to perform common class tasks are called _____	container classes	receptacle classes	repository classes	alembic classes	container classes
15	The most common operation used in constructors is _____	addition	overloading	assignment	polymorphism	assignment
16	An operator function is created using _____ keyword	iterator	allocator	constructor	operator	operator
17	Overloading involves writing two or more functions with _____	different names and different argument lists	different names and the same argument list	the same name and different argument lists	the same name and the same argument list	the same name and different argument lists
18	Reusability of the code can be achieved in CPP through _____	Polymorphism	Encapsulation	Inheritance	Message Passing	Inheritance
19	class X, class Y and class Z are derived from class BASE. This is _____ inheritance	Multiple	Multilevel	Hierarchical	Single	Hierarchical
20	During a class inheritance in CPP, if the visibility mode or mode of derivation is not provided, then by default visibility mode is _____	public	protected	private	Friend	private
21	If the derived class is struct, then default visibility mode is _____	public	protected	private	struct can't inherit class	public
22	When a child class inherits traits from more than one parent class, this type of inheritance is called _____ inheritance	Hierarchical	Hybrid	Multilevel	Multiple	Multiple
23	Within parentheses, catch blocks can have _____	no arguments	one argument	two arguments	as many arguments as necessary	one argument
24	Which of the following are available only in the class hierarchy chain?	Public data members	Private data members	Protected data members	Member functions	Protected data members
25	Function templates _____	must have exactly one parameter	may have more than one parameter as long as they are of the same type	may have more than one parameter of any type	may not have parameters	may have more than one parameter of any type
26	When a function performs tasks based on a decision, it has _____	functional cohesion	coincidental cohesion	logical cohesion	no cohesion	logical cohesion
27	When the compiler places a copy of a small function's statements directly into a program, the function is said to be _____	overloaded	mangled	inline	redundant	inline
28	A C++ _____ is a program that runs in a DOS window	algorithm	cast application	console application	source application	console application
29	_____ variables remain in memory until the program ends	Area	Global	Local	Reference	Global
30	The end of string is recognized by _____	the null character	the newline character	the $ sign	the / sign	the null character
31	Variable names known only to the procedure in which they are declared are _____	local	global	recent	internal	local
32	Which of the following tells C++ to display numbers with zero decimal places?	setiosflags(0)	setiosflags(zero)	setprecision(0)	setprecision(zero)	setprecision(0)
33	Code that has already been tested is said to be _____	inherited	reusable	reliable	polymorphic	reliable
34	Which of the following statements is correct?	Base class pointer cannot point to derived class	Derived class pointer cannot point to base class	Pointer to derived class cannot be created	Pointer to base class cannot be created	Derived class pointer cannot point to base class
35	The two operators && an || are _____	arithmetic operators	equality operators	logical operators	relational operators	logical operators
36	What is the return type of the conversion operator?	void	int	float	no return type	no return type
37	Why we use the "dynamic_cast" type conversion?	result of the type conversion is a valid	to be used in low memory	result of the type conversion is a invalid	to check that operators and operands	result of the type conversion is a valid
38	How many parameters does a conversion operator may take?	0	1	2	as many as possible	0
39	When a class serves as a base class to others, _____	all of its members are inherited	all of its members are inherited, except for any private members	all of its members are inherited, except for any protected members	None of its members is inherited unless specifically "listed	all of its members are inherited, except for any private members
40	Which of the following C++ expressions will find the square root of the number 16?	pow (16, 2) (b) root (16, 2)	sqroot (16)	sqrt (16, 2)	sqrt (16)	sqrt (16)
41	You construct a class in two sections, known as the _____	header and body	type and parameters	declaration and implementation	pointer and variable	declaration and implementation
42	The statement i*=3 is equivalent to _____	i = 3*	i = 3	i* = 3	i = i * 3	i = i * 3
43	The complement operator is represented by the symbol _____	~	^	!	<	~
44	The last statement in a value-returning function is always _____	};	result expression;	return;	return expression;	return expression;
45	Which of the following is an invalid visibility label while inheriting a class?	public	private	protected	friend	friend
46	Which of the following is used to make an abstract class?	Declaring it abstract using static keyword	Declaring it abstract using virtual keyword	Making at least one member function as virtual function	Making at least one member function as pure virtual function	Making at least one member function as pure virtual function
47	Procedural cohesion is similar to sequential cohesion, except that with procedural cohesion _____	the tasks are not done in order	the tasks are simpler	the tasks share data	the tasks do not share data	the tasks do not share data
48	The extraction operator >> is a(n) _____	overloaded function	C++ class	C++ object	static reference variable	overloaded function
49	Reference variables and const class member _____	must be assigned values in any derived class	must never be initialized in a base class	must be initialized, rather than assigned values	must not exit if a class is to be a base class	must be initialized, rather than assigned values
50	A predefined function that may be used to handle memory allocation errors is _____	handle_error	set_new_handler	new_fix	memory_error	set_new_handler
51	The directives for the preprocessors begin with _____	ampersand symbol (&)	two Slashes (//)	number Sign (#)	less than symbol (<)	number Sign (#)
52	The memory address of the first element of an array is called _____	floor address	foundation address	first address	base address	base address
53	_____ variables cannot be reinitialized	int	float	static	register	static
54	A statement that unconditionally modifies the execution flow to proceed from a remote labeled statement is _____	if	while	goto	if else	goto
55	Which of the following operator has highest precedence?	[ ]	!	++	/	[ ]
56	Which of the following operator has lowest precedence?	>>=	>>	++	[ ]	>>=
57	_____ is a short hand operator	-	+=	++	>>	+=
58	_____ operator is used for finding the size of the variable	:	?:	.	sizeof	sizeof
59	The result of a relational or logical expression is of type _____	integer	float	boolean	character	boolean
60	Expression C=i++ causes _____	value of i assigned to C and then i incremented by 1	i to be incremented by 1 and then value of i assigned to C	value of i assigned to C	i to be incremented by 1	value of i assigned to C and then i incremented by 1.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed University Established Under Section 3 of UGC Act 1956)

Coimbatore – 21

(For the candidates admitted from 2017 onwards)

DEPARTMENT OF COMMERCE (CA)

SUBJECT : OBJECT ORIENTED PROGRAMMING WITH C++

SEMESTER : I

SUBJECT CODE: 17CCP104 **CLASS : I M.COM CA**

POSSIBLE QUESTIONS – UNIT IV

Part A (1 Mark)

(Online Examinations)

PART B (2 Marks)

1. What is Type Conversion
2. Define operator overloading
3. What is Unary Operator
4. What is Binary Operator
5. What are manipulation of String Using Operators
6. List out any 3 rules for Operator Overloading
7. Define Inheritance
8. What are different types of inheritance
9. What is Abstract class
10. What is Virtual Base Class
11. What is Multiple Inheritance
12. How will you define derived class?

PART C (6 Marks)

1. Write a Program for Operator overloading using simple variables
2. Describe Overloading Unary & Binary operators
3. Write in detail – Rules for Operator Overloading
4. Explain Inheritance with example.
5. Write a program using simple statements for inheritance concept.
6. Explain binary Operators with example.
7. Differentiate between multiple and multilevel inheritance.
8. Write a program for operator overloading.
9. Explain Unary Operators with example.
10. Describe virtual function with suitable example

Enable | Enlighten | Enrich
(Deemed to be University)
(Under Section 3 of UGC Act 1956)

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT : **OBJECT ORIENTED PROGRAMMING WITH C++**
SEMESTER : I
SUBJECT CODE: **17CCP104** **CLASS** : **I M.COM CA**

UNIT V

Pointers, Virtual functions and Polymorphism – Pointers to Objects – This Pointer – Pointer to Derived Classes – Virtual Functions. Working with Files – Classes for File Stream Operations Opening and Closing a File

Unit – V

Pointers: Pointers to objects

- Object pointers are useful in creating objects at run time.
- Pointer objects are useful to access the public members of an object.

Syntax:

classname *ptr-variable;

Example:

```
#include<iostream.h>
class item
{
  int code;
  float price;
  public:
    void getdata(int a, float b)
    {
      code=a;
      price=b;
    }
    void show()
    {
        cout<<"Code:"<<code;
        cout<<"\nPrice:"<<price;
    }
};
void main()
{
   item *p=new item[3];
```

```
  item *d=p;
  int x,i;
  float y;
  for(i=0;i<3;i++)
  {
    cout<<"Input Code and Price for Item"<<i+1;
    cin>>x>>y;
    p->getdata(x,y);
    p++;
  }
  for(i=0;i<3;i++)
  {
    cout<<"Item"<<i+1<<endl;
    d->show();
    d++;
  }
}
```

this pointer

- this is used to represent current object.
- this is a keyword.
- this is an object that invokes a member function.
- This unique pointer is automatically passed to a member function when it is called.
- The pointer this acts as an implicit argument to all the member functions.

Syntax:

this - > variable or function name;

Example:

```
this->a;
return this;
```

Program:

```
#include<iostream.h>
class person
{
  char name[20];
  float age;
  public:
    person(char *s, float a)
    {
      strcpy(name,s);
      age=a;
    }
    person &person :: greater(person &x)
    {
       if(x.age>=age)
           return x;
       else
           return *this;
    }
    void display()
    {
      cout<<"Name : "<<name;
      cout<<"\nAge  : "<<age;
    }
};
void main()
```

```
{
  person p1("Raja",42.78),p2("Ram",35.8),p3("Arun",45.3);
  person p=p1.greater(p3);
  cout<<"Elder Person is :\n ";
  p.display();
  p=p1.greater(p2);
  cout<<"Elder Person is :\n ";
  p.display();
}
```

Pointers to derived classes

- Pointers are used not only to base class but also to derived class.
- Pointers to objects of a base class are type-compatible with pointers to objects of derived class.
- A single pointer variable can be made to point to object belonging to different classes.

Example:

```
B *ptr;
B b;
D d;
ptr=&b;
```

Program:

```
#include<iostream.h>
class BC
{
  public :
    int b;
    void show()
    {
```

```
      cout<<"b = "<<b<<endl;
    }
};
class DC: public BC
{
  public:
    int d;
    void show()
    {
      cout<<"b = "<<b<<endl;
      cout<<"d = "<<d<<endl;
    }
};
void main()
{
  BC *ptr;
  BC base;
  cout<<"Base Class Pointer Call"<<endl;
  ptr=&base;
  ptr->b=100;
  ptr->show();
  cout<<"Base Class Pointer Call using Derived Class"<<endl;
  DC dc;
  ptr=&dc;
  ptr->b=200;
  ptr->show();
  cout<<"Derived Class Pointer Call"<<endl;
  DC *dptr;
  dptr=&dc;
```

```
  dptr->d=100;
  dptr->show();
  cout<<"Using (DC*)bptr)\n";
  ((DC *)ptr)->d=200;
  ((DC *)ptr)->show();
}
```

Virtual functions

- The appropriate member function could be selected while the program is running. This is known as runtime Polymorphism
- Runtime Polymorphism is achieving through virtual function.
- Virtual function is created using the keyword virtual.
- When the function made virtual, C++ determines which function to use at run time based on the type of the object pointed to by the base pointer, rather than the type of the pointer.
- By making the base class pointer to point to different objects, can execute different versions of the virtual function.

Syntax:

```
virtual RT function-name(arg list)
{
        .......
        .......
}
```

Example:

```
virtual void show()
{
        ........
        ........
}
```

Program:

```
#include<iostream.h>
class Base
{
  public :
    void display()
    {
      cout<<"\nDisplay Base Class";
    }
    virtual void show()
    {
      cout<<"Base Class Show";
    }
};
class Derived: public Base
{
  public :
    void display()
    {
      cout<<"\nDisplay Derived Class";
    }
    virtual void show()
    {
      cout<<"Derived Class Show";
    }
};
void main()
{
  Base *ptr;
```

```
  Base b;
  Derived d;
  cout<<"\nPointer point to Base Class";
  ptr=&b;
  ptr->display();
  ptr->show();
  cout<<"\nPointer point to Derived Class";
  ptr=&d;
  ptr->display();
  ptr->show();
}
```

Rules:

1. The virtual function must be members of some class.
2. They can not be static members.
3. They are accessed by using object pointers.
4. A virtual function can be a friend of another class.
5. A virtual function in a base class must be defined, even through it may not be used.
6. The prototype of the base class version of a virtual function and all the derived class versions must be identical. If two functions with the same name have different prototypes, C++ considers them as overloaded function.
7. Can have virtual constructors not have virtual destructors.
8. While a base class pointer can point to any type of derived object, the reverse is not true.
9. When a base class pointer points to a derived class, incrementing or decrementing it will not make it to point to the next object of the derived class

10. If a virtual function is defined in the base class, it need not be necessarily redefined in the derived class.

Pure Virtual Function:

- A pure virtual function is a function declared in a base class that has no definition relative to the base class.
- In this case the compiler requires the derived class to either define the function or redeclare it as pure virtual function.

Syntax:

virtual RT function-name()=0;

Example:

virtual void show()=0;

Managing Console I/O Operations:

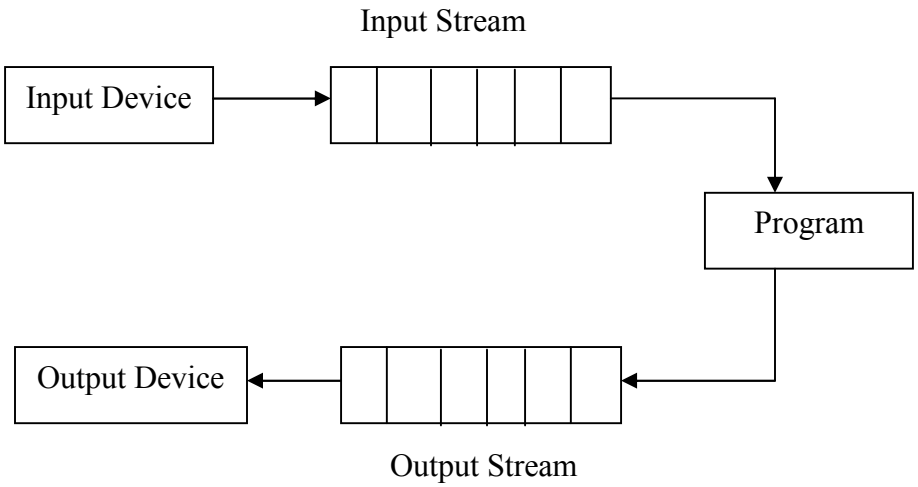
- C++ provides rich set of I/O functions and operations to manage console I/O operations.
- C++ uses the concept stream and stream classes to implement its I/O operations with the console and disk files.

Managing console I/O operations :- C++ streams

C++ Streams:

- The I/O system in C++ is designed to work with a wide variety of devices including terminals, disks, and tape drives.
- The I/O stream supplies an interface to the programmer that independent of the actual device being accessed. This interface is known as stream.
- A stream is a sequence of bytes.
- It acts either as a source from which the input data can be obtained or as a designation to which data can be sent.

- The source stream that provides data to the program is called the input stream and the designation stream that receives output from the program is called the output stream.

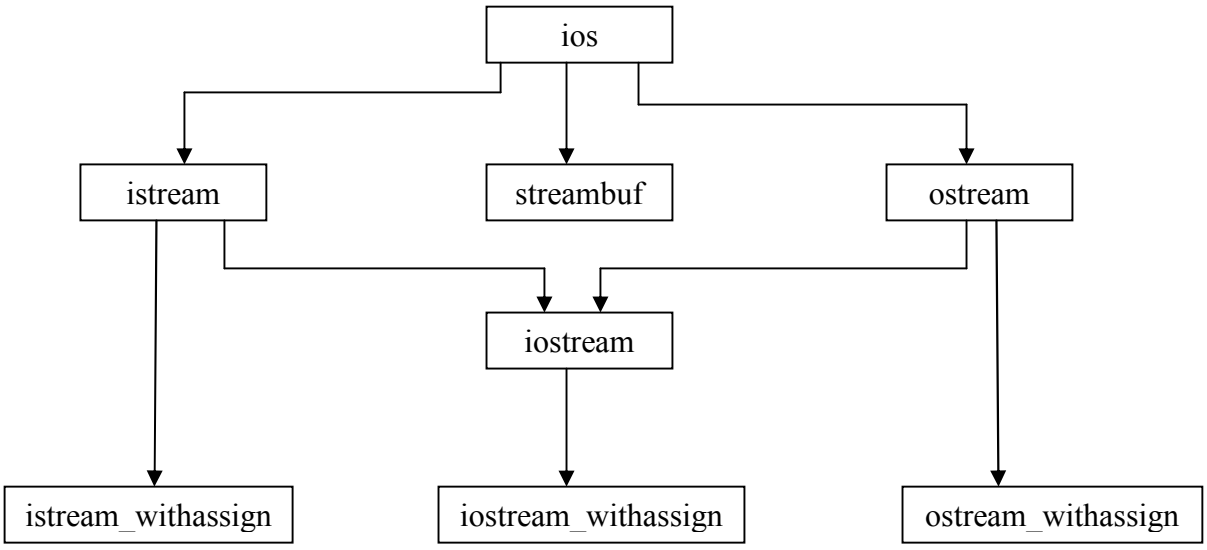


- The data in the input stream can come from the keyboard or any other storage devices.
- The data in the output stream can go to the screen or any other storage devices.
- A stream is an interface between the program and the I/O devices.
- C++ contains pre-defined streams that are automatically opened when a program begins its execution.
- cin and cout also belongs to such streams.
- cin represents the input stream connected to the standard input device.
- cout represents the output stream connected to the standard output device.

C++ stream classes

- The C++ I/O system contains a hierarchy of classes that are used to define various streams to deal with console and disk files. These classes are called stream classes.

- The header file iostream should be included in all the programs that communicate with console unit.



- ios is the base class for istream and ostream.
- istream and ostream are the base class of iosteam.
- ios is declared as virtual so only one copy of its members are inherited by the iostream
- ios provides the basic support for all I/O operations.
- The class istream provides the facilities for formatted and unformatted Input.
- The class ostream provides the facilities for formatted output.

Unformatted I/O operations

Overloaded operators >> and <<

- The objects cin and cout for the input and output of data of various types.
- This is done by overloading the operator >> and <<.
- The operator >> is overloaded in the istream class and << overloaded in the ostream.

Example:

```
cin>>a;
cout<<a;
```

put() and get() Functions:

- The classes istream and ostream defines two member functions get() and put() to handle the single character input/output operations.
- There are two type of get() function
  - get(char *) – assign the input to a variable
  - get(void) – returns the input character.

Example:

```
cin.get(c); //read and assign the value for c
c=cin.get(); // returns the read character to c
```

- put() is a member of ostream class, can be used to output a line of text, character by character.

Syntax:

```
cin.put(char *);
```

Example:

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

Program:

```
#include<iostream.h>
void main()
{
  char c;
  int ct=0;
```

```
  cout<<"Input Text:\n";
  cin.get(c);
  cout.put(c);
  while(c!='\n')
  {
     cout.put(c);
     c=cin.get();
     ct++;
  }
  cout<<"Number of Characters Entered : "<<ct;
}
```

getline() and write() Function:

- getline() and write() is used to read the text line by line.
- The getline() function reads a whole line of text that ends with a newline character.

Syntax:

cin.getline(line,size)

- line- variable name
- The reading is terminated as soon as either the newline character '\n' is encountered or size-1 characters are read.

Example:

```
char name[20];
cin.getline(name,20);
```

Program:

```
#include<iostream.h>
void main()
{
  char name[25];
  cout<<"\nEnter the name:\n";
```

```
  cin.getline(name,20);
  cout<<"\nName : "<<name;
}
```

- The write() function display an entire line

Syntax:

cout.write(line,size)

- line- variable name
- The writing is terminated as soon as either the newline character '\n' is encountered or size-1 characters are written.

Example:

```
char name[20]=”welcome”;
cout.write(name,20);
```

Program:

```
#include<iostream.h>
#include<string.h>
void main()
{
  char name[25];
  int i;
  cout<<"\nEnter the name:\n";
  cin.getline(name,20);
  int l1=strlen(name);
  for(i=1;i<l1;i++)
  {
     cout.write(name,i);
     cout<<endl;
  }
  for(i=l1;i>0;i--)
  {
```

```
    cout.write(name,i);
    cout<<endl;
  }
}
```

Formatted console I/O operations

- C++ support a number of features that could be used for formatting the output
- It includes
  - ios class function and flags.
  - Manipulators.
  - User defined output functions.

ios Class Function and Flags:

- The ios class contain a large number of member functions that would help us to format the output in a number of ways.

The members are

Function	**Task**
width()	To specify the required fields size for displaying an output value
precision()	To specify the number of digits to be displayed after the decimal point of a float values.
fill()	To specify a character that is used to fill the unused portion of a field
setf()	To specify format flags that can control the form of output display.
unsetf()	To clear the flags specified

width():

- To specify the required fields size for displaying an output value

Syntax:

cout.width(size)

Example:

```
cout.width(5);
```

Program:

```
#include<iostream.h>
void main()
{
  int item[4]={7,12,80,89};
  int cost[4]={75,100,125,90};
  cout.width(5);
  cout<<"Items";
  cout.width(8);
  cout<<"Cost";
  cout.width(20);
  cout<<"Total Values\n";
  int sum=0;
  for(int i=0;i<4;i++)
  {
    cout.width(5);
    cout<<item[i];
    cout.width(8);
    cout<<cost[i];
    cout.width(15);
    cout<<cost[i]*item[i]<<endl;
    sum=sum+cost[i]*item[i];
  }
  cout<<"\nGrand Total=";
```

```
  cout.width(2);
  cout<<sum<<"\n";
}
```

Output:

```
Items    Cost       Total Values
   7      75          525
  12      100          1200
  80      125         10000
  89       90         8010

Grand Total=19735
```

precision():

- To specify the number of digits to be displayed after the decimal point of a float values.

Syntax:

```
cout.precision(size)
```

Example:

```
cout.precision(2);
```

Program:

```
#include<iostream.h>
#include<math.h>
void main()
{
  cout.precision(3);
  cout.width(7);
  cout<<"Values";
  cout.width(15);
  cout<<"Squrt values\n";
```

```
  for(int i=1;i<5;i++)
  {
    cout.width(5);     cout<<i;     cout.width(13);
    cout<<sqrt(i)<<endl;
  }
}
```

Output:

```
Values  Squrt values
    1           1
    2       1.414
    3       1.732
    4           2
```

fill():

- To specify a character that is used to fill the unused portion of a field

Syntax:

```
cout.fill(character)
```

Example:

```
cout.fill('*');
```

Program:

```
#include<iostream.h>
void main()
{
  int a=5679;   cout.width(10);
  cout.fill('*');   cout<<"Values";   cout<<a<<"\n";
}
```

Output:

```
****Values5679
```

setf():

- To specify format flags that can control the form of output display.

Syntax:

cout.setf(arg1,arg2);

- arg1-formatting flags defined in the class ios.
- arg2-formatting flags defined in the class ios. It is also known as bit fields

Flags and Bit Fields

Format required	**Flag (arg1)**	**Bit-field(arg2)**
Left-justified O/P	ios::left	ios::adjustfield
Right- justified O/P	ios::right	ios::adjustfield
Padding after sign or base Indicator(+##20)	ios::internal	ios::adjustfield
Scientific notation	ios::scientific	ios::floatfield
Fixed point notation	ios::fixed	ios::floatfield
Decimal base	ios::dec	ios::basefield
Octal base	ios::oct	ios::basefield
Hexadecimal base	ios::hex	ios::basefield

Flags Without Bit Fields

Flag	**Meaning**
ios::showbase	Use base indicator on output
ios::showpos	Print + before positive numbers
ios::showpoint	Show trailing decimal point and zeroes
ios::uppercase	Use uppercase letters for hex output
ios::skipus	Skip white space on input
ios::unitbuf	Flush all streams after insertion
ios::stdio	Flush stdout and stderr after

	insertion

Example:

cout.setf(ios::left. ios::adjustfield);

Program:

```
#include<iostream.h>
#include<math.h>
void main()
{
  cout.fill('*');
  cout.setf(ios::left,ios::adjustfield);
  cout.width(10);   cout<<"Values";
  cout.setf(ios::right,ios::adjustfield);
  cout.width(15);
  cout<<"Sqrt Of Value \n";
  cout.fill('.');   cout.precision(4);
  cout.setf(ios::showpoint);   cout.setf(ios::showpos);
  cout.setf(ios::fixed,ios::floatfield);
  for(int n=1;n<10;n++)
  {
      cout.setf(ios::internal,ios::adjustfield);
      cout.width(5); cout<<n;
      cout.setf(ios::right,ios::adjustfield);
      cout.width(20);
      cout<<sqrt(n)<<endl;
  }
  cout.setf(ios::scientific,ios::floatfield);
  cout<<"sqrt(100)= "<<sqrt(100)<<"\n";
}
```

Output:

Values****Sqrt Of Value

+...1.....+1.0000

+...2.....+1.4142

+...3.....+1.7321

+...4.....+2.0000

+...5.....+2.2361

+...6.....+2.4495

+...7.....+2.6458

+...8.....+2.8284

+...9.....+3.0000

sqrt(100)= +1.0000e+01

Managing output with manipulators

- The header file iomanip provides a set of functions called manipulators which can be used to manipulate the output formats.
- Provide the same features as that of the ios member functions and flags.
- The various manipulators are

Manipulator	**Meaning**	**Equivalent**
setw(int w)	Set the field width to w	width()
setprecision(int d)	Set the floating point precision to d	precision()
setfill(char c)	Set the fill character to c	fill()
setiosflags(long f)	Set the format flag f	setf()
resetiosflags(long f)	Clear the flag specified by f	unsetf()
endl	Insert new line and flush stream	"\n"

Program:

```
#include<iostream.h>
#include<iomanip.h>
void main()
{
  cout.setf(ios::showpoint);
  cout<<setw(5)<<"n"<<setw(15)<<"Inverse of n"<<setw(15)<<"Sum of terms"<<endl<<endl;
  double term, sum;
  for(int n=1;n<10;n++)
  {
     term=1.0/float(n);
     sum=sum+term;
cout<<setw(5)<<n<<setw(14)<<setprecision(2)<<setiosflags(ios::scientific)<<term<<setw(13)<<resetiosflags(ios::scientific)<<sum<<endl;
   }
}
```

Output:

```
n   Inverse of n   Sum of terms

 1      1.00e+00        1.00
 2      5.00e-01        1.50
 3      3.33e-01        1.83
 4      2.50e-01        2.08
 5      2.00e-01        2.28
 6      1.67e-01        2.45
 7      1.43e-01        2.59
 8      1.25e-01        2.72
 9      1.11e-01        2.83
```

Designing Our Own Manipulators:

- Own manipulators are designed for certain special purposes.
- The general form is

```
ostream &manipulator-name(ostream &output)
{
        ……
        …….(code)
}
```

Example:

```
ostream &unit(ostream &output)
{
        output<<" inches";
        return output;
}
```

Program:

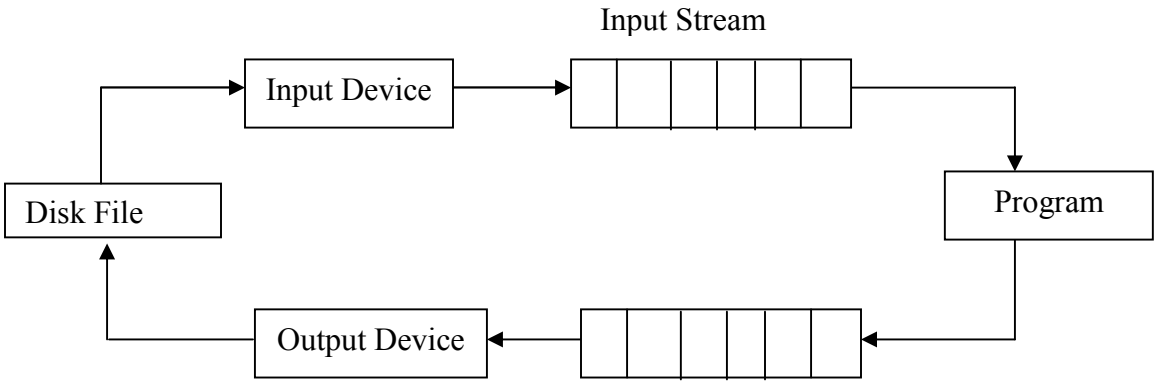
```
#include<iostream.h>
#include<iomanip.h>
ostream &unit(ostream &output)
{
        output<<" inches";
        return output;
}
void main()
{
  cout<<"Height : 56 "<<unit;
}
```

Output: Height :
56 inches

Files: Classes for file stream operations

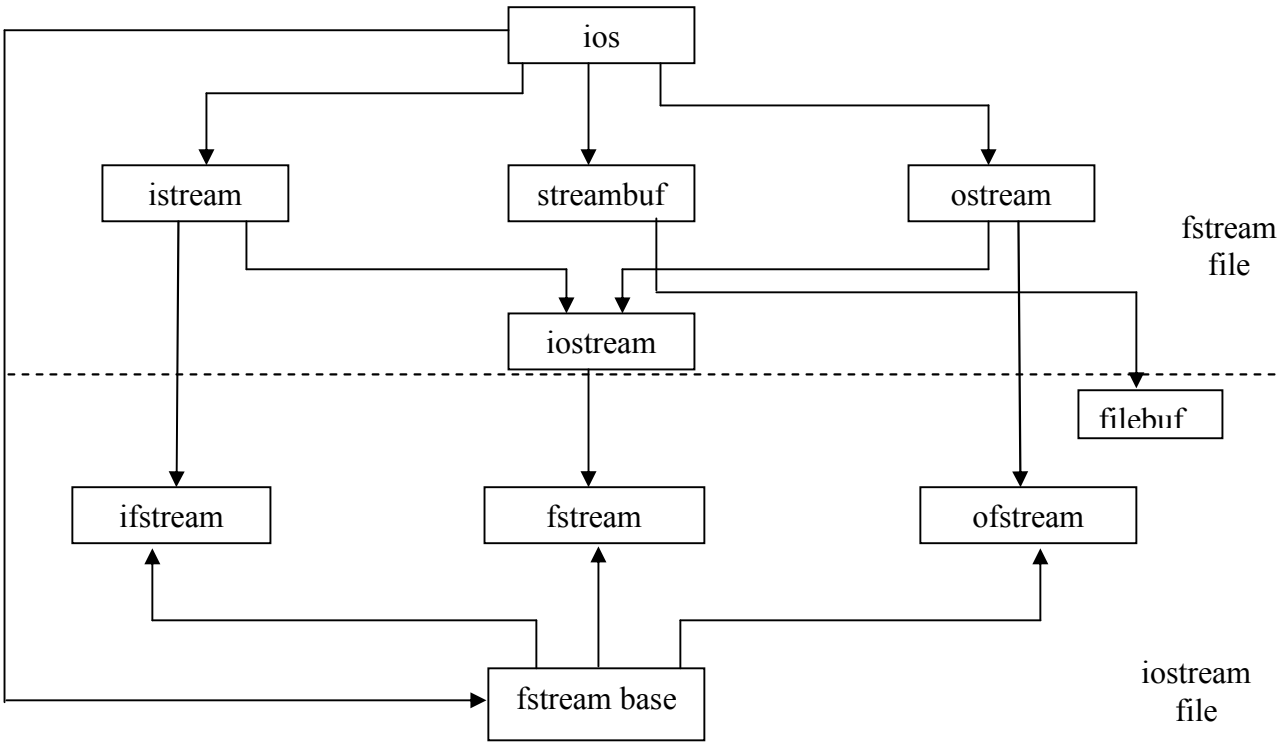
Working With Files:

- The data is stored in secondary devices using the concept of File.
- A File is a collection of related data stored in a particular area on the disk.
- Programs typically involves either or both of the following kinds of data communication:
  - Data transfer between the console unit and the program.
  - Data transfer between the program and the disk



Classes for File Stream Operations:

- The C++ I/O system contains a set of classes that define the file handling methods.
- File handling class includes ifstream, ofstream, and fstream. These classes are derived from the corresponding iostream class.
- These are the class designed to manage the disk files
- All the classes are declared in fstream so all the program should include this header file



File Operations:

- Open file
- Read and Write Operations
- Closing a file

Opening and closing a file

- Use a disk file requires
- Suitable name for the file.
- Data type and structure.
- Purpose
- Opening method

Opening Files Using Constructor:

- A constructor is used to initialize an object while it is being created.

- A file name is used to initialize the file stream object.

Steps for Creating Object:

- Create a file stream object to manage the stream using appropriate class. The class ofstream is used to create the output stream and the class ifstream to create the input stream
- Initialize the file object with the desired filename.

Example:

ofstream outfile("result");

- This create outfile as ofstream object that manages the output stream. This statement also opens the file result and attaches to the output stream outfile.

ifstream infile("data")

- This create infile as ifstream object that manages the input stream. This statement also opens the file data and attaches to the input stream infile.

Program:

```
#include<iostream.h>
#include<fstream.h>
void main()
{
  ofstream outf("Item");
  char name[30];
  float cost;
  cout<<"Enter the Item Name: ";
  cin>>name;
  outf<<name<<"\n";
  cout<<"Enter the Item Cost: ";
  cin>>cost;
  outf<<cost<<"\n";
```

```
  outf.close();
  ifstream inf("Item");
  inf>>name;
  inf>>cost;
  cout<<"\n Item Name:"<<name<<"\n";
  cout<<"Item cost:"<<cost<<"\n";
  inf.close();
}
```

Opening a Files Using open():

- The function open() can be used to open multiple files that use the same stream object.

Syntax:

File-stream-class stream-object;

stream-object.open("file name");

Example:

```
ofstream outfile;
outfile.open("data");
……….
……….
outfile.close();
```

Program:

```
#include<iostream.h>
#include<fstream.h>
void main()
{
  ofstream outf;
  char name[30];
  int i;
  outf.open("prog");
```

```
cout<<"Enter the 3 Programming Language:\n ";
for(i=0;i<3;i++)
{
   cin>>name;
   outf<<name<<"\n";
}
outf.close();
outf.open("soft");
cout<<"Enter the 3 softwares:\n ";
for(i=0;i<3;i++)
{
   cin>>name;
   outf<<name<<"\n";
}
outf.close();
ifstream inf;
inf.open("prog");
cout<<"Programming Language:\n";
while(inf)
{
  inf.getline(name,50);
  cout<<name<<"\n";
}
inf.close();
inf.open("soft");
cout<<"Software:\n";
while(inf)
{
  inf.getline(name,50);
```

```
    cout<<name<<"\n";
  }
  inf.close();
}
```

Detecting End of File:

- eof() function is used to detect end of File.
- It is the member function of ios class.
- It returns a non-zero value if the end-of-file condition is encountered and a zero otherwise.

Example:

```
if(fileobj.eof() !=0)
{ exit(0);}
```

File Modes:

- istream and ostream constructors and function open() to create new files as well as to open the existing files.
- open() method takes two arguments one for file name and other for mode.

Syntax:

Stream-object.open("file-name",mode);

- mode specifies the purpose for which the file is opened.
- The default mode values are:
- ios::in for ifstream functions meaning open for reading only.
- ios::out for ofstream functions meaning open for writing only.

File Mode Parameters:

Parameter	**Meaning**
ios::app	Append to end of file
ios::ate	Go to end of the 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	Opens fails if the file already exist
ios::out	Open file for writing only
ios::trunc	Delete the contents of the file if it exists.

- Opening a file in ios::out mode also open it in the ios::trunc mode by default.
- ios::app and ios::ate takes to the end of the file when it is opened
- The difference between ios::app and ios::ate is ios::app allows us to add data to the end of the file but ios::app mode permits to add data or to modify data anywhere in the file.
- ios::app can be used only with the file capable of output.
- Creating a stream using ifstream implies input and creating a stream using ofstream implies output. So in this cases it is not necessary to provide the mode parameters.
- The fstream class does not provide a mode by default and therefore it is necessary to provide the mode explicitly when using an object of fstream class.
- The mode can combine two or more parameters using the bitwise OR operator

fout.open("data",ios:app | ios::nocreate)

File Pointers and Their Manipulation

All I/O streams objects have, at least, one internal stream pointer: ifstream, like istream, has a pointer known as the get pointer that points to the element to be read in the next input operation.

ofstream, like ostream, has a pointer known as the put pointer that points to the location where the next element has to be written.

Finally, fstream, inherits both, the get and the put pointers, from iostream (which is itself derived from both istream and ostream).

File Manipulators

seekg() moves get pointer(input) to a specified location

seekp() moves put pointer (output) to a specified location

tellg() gives the current position of the get pointer

tellp() gives the current position of the put pointer

The other prototype for these functions is:

seekg(offset, refposition);

seekp(offset, refposition);

The parameter offset represents the number of bytes the file pointer is to be moved from the location specified by the parameter refposition.

The refposition takes one of the following three constants defined in the ios class.

ios::beg- start of the file

ios::cur- current position of the pointer

ios::end- end of the file

example: file.seekg(-10, ios::cur);

Sequential input and output operations

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

put() and get() function:

- The function put() writes a single character to the associated stream.

- The function get() reads a single character to the associated stream.

Syntax:

File-object.get(character)

File-object.put(character)

Program:

```
#include<iostream.h>
#include<fstream.h>
#include<string.h>
void main()
{
  fstream file;
  char name[30];
  int i;
  cout<<"Enter Name: ";
  cin>>name;
  int l=strlen(name);
  file.open("text",ios::in | ios::out);
  for(i=0;i<l;i++)
  {
    file.put(name[i]);
  }
  file.seekg(0);
  char c;
  while(file)
  {
     file.get(c);
     cout<<c;
  }
```

```
  file.close();
}
```

write() and read() function:

- The function write() and read() handles the data in binary form. This means that the values stored in the disk file in the same format in which they stored in the internal memory.
- An int takes two bytes to store its value in the binary form, irrespective of its size.
- The binary format is more accurate for storing the numbers in the exact internal representation.
- The binary format is much faster to saving the data to.

Syntax:

```
inFile-object.read((char *) &v, sizeof(v))
outFile-object.write((char *) &v, sizeof(v))
```

- The first argument is the address of variable v.
- The second argument is the length of the variable in bytes.
- The address of the variable must be cast to type char *.

Program:

```
#include<iostream.h>
#include<fstream.h>
#include<iomanip.h>
void main()
{
  float height[5]={176,182,167.89,177.9,160.24};
  ofstream ofile;
  int i;
  ofile.open("data");
  ofile.write((char *) &height, sizeof(height));
  ofile.close();
```

```
  ifstream infile;
  infile.open("data");
  infile.read((char *) &height, sizeof(height));
  for(i=0;i<5;i++)
  {
     cout.setf(ios::showpoint);
     cout<<setw(10)<<setprecision(2)<<height[i]<<endl;
  }
  infile.close();
}
```

Reading and Writing a Class Object:

- C++ supports features for writing to and reading from the disk files objects directly.
- The binary input and output functions read() and write() are designed to do exactly this job.
- These functions handle the entire structure of an object as a single unit, using the computer's internal representation of data.
- For instance, the function write() copies a class object from the memory byte by byte with no conversion.
- Only data members are written to the disk file and the member functions are not.
- The length of the object is obtained by sizeof operator.

Program:

```
#include<iostream.h>
#include<fstream.h>
#include<iomanip.h>
class Inventory
{
  char name[20];
```

```
  int code;
  float cost;
  public:
    void readdata();
    void show();
};
void Inventory::readdata()
{
   cout<<"Enter Name: ";
   cin>>name;
   cout<<"Enter Code: ";
   cin>>code;
   cout<<"Enter Cost: ";
   cin>>cost;
}
void Inventory::show()
{
   cout<<setiosflags(ios::left)<<setw(10)<<name
       <<setiosflags(ios::right)<<setw(10)<<code
       <<setprecision(2)<<setw(10)<<cost<<endl;
}
void main()
{
  Inventory item[3];
  fstream file;
  file.open("stock.dat",ios::in |ios::out);
  cout<<"Enter Details of Items\n";
  for(int i=0;i<3;i++)
  {
```

```
    item[i].readdata();
    file.write((char *) &item[i], sizeof(item[i]));
  }
  file.seekg(0);
  cout<<"\n\nOutput\n\n";
  for(i=0;i<3;i++)
  {
    file.read((char *) &item[i], sizeof(item[i]));
    item[i].show();
  }
  file.close();
}
```

updating a file random access

- Updating is a routine take in the maintenance of any data file.
- Updating include the following task.
- Displaying the contents of a file.
- Modifying an existing item.
- Adding a new item.
- Deleting an existing item.

Program:

```
#include<iostream.h>
#include<fstream.h>
#include<iomanip.h>
class Inventory
{
  char name[20];
  int code;
  float cost;
```

```
  public:
    void readdata();
    void show();
};
void Inventory::readdata()
{
   cout<<"Enter Name: ";
   cin>>name;
   cout<<"Enter Code: ";
   cin>>code;
   cout<<"Enter Cost: ";
   cin>>cost;
}
void Inventory::show()
{
   cout<<setiosflags(ios::left)<<setw(10)<<name
       <<setiosflags(ios::right)<<setw(10)<<code
       <<setprecision(2)<<setw(10)<<cost<<endl;
}
void main()
{
  Inventory item;
  fstream file;
  file.open("stock.dat",ios::ate| ios::in |ios::out |ios::binary);
  file.seekg(0,ios::beg);
  cout<<"\nCurrent Contant of File\n";
  while(file.read((char *) &item, sizeof(item)))
  {
     item.show();
```

```
}
file.clear();
cout<<"\nAdd An Item\n";
item.readdata();
char ch;
cin.get(ch);
file.write((char *) &item, sizeof(item));
file.seekg(0);
cout<<"\nContant of File After Appended\n";
while(file.read((char *) &item, sizeof(item)))
{
  item.show();
}
int ls=file.tellg();
int n=ls/sizeof(item);
cout<<"\nNumber of Objects="<<n;
cout<<"\nTotal bytes in the file="<<ls;
cout<<"Modify An Item";
int no;
cout<<"\nEnter the Object Number to Update : ";
cin>>no;
cin.get(ch);
int loc=(no-1)*sizeof(item);
if(file.eof())
  file.clear();
file.seekp(loc);
cout<<"\nEnter New values of object:\n";
item.readdata();
cin.get(ch);
```

```
  file.write((char *) &item, sizeof(item))<<flush;
  file.seekg(0);
  cout<<"\nContant of File After Modified\n";
  while(file.read((char *) &item, sizeof(item)))
  {
     item.show();
  }
  file.close();
}
```

Command-line Arguments:

- C++ support a feature of supply of arguments to the main() function.
- The command-line arguments are achieved by the arguments of the main() function.

Syntax:

main(int argc, char *argv[])

- argc known as argument counter, represents the number of arguments in the command line.
- argv known as argument vector, is an array of char type pointers that pointers that points to the command line arguments.
- The size of this array will be equal to the value of argc.
- Arguments are supplied at the time of invoking the program.

Example:

C:\>program-file-name first-file second-file

- Program-file-name is the name of the file containing the program to be executed.
- first-file and second-file are the file names passed to the program as command-line arguments.

- The first argument is always the file name and contains the program to be executed.
- The value of argc would be 3 and the argv would be an array of 3 pointers to strings
  - argv[0] →program-file-name
  - argv[1] →first-file-name //used for reading purpose
  - argv[2] →second-file-name //used for writing purpose

Templates and Exceptions:- Templates

Templates:

- Templates is one of the features added to C++ recently.
- It is a new concept which enables us to define generic classes and functions and thus provides support for generic programming.
- Generic programming is an approach where generic types are used as parameters in algorithms so that they work for a variety of suitable data types and data structure.
- A template can be used to create a family of classes or functions.
- For example, a class template for an array class would enable us to create arrays of various data types such as int array and float array.
- Similarly, define a template for a function, say mul(), that would help us create various versions of mul() for multiplying int, float and double type values.
- A template can be considered as a kind of macro.
- When an object of a specific type is defined for actual use, the template definition for that class is substitute with required data type. Since a template 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 sometimes called parameterized classes or functions.

Class templates

Class Templates:

- A simple process to create a generic class using a template with anonymous type.
- template is the keyword used to create Template
- The class template definition is very similar to an ordinary class definition expect the prefix template<class T> and the use of type T.
- This prefix tells the compiler that is going to declare a template and use T as a type name in the declaration.

Syntax:

```
template <class T>
class class-name
{
        //……
        //class member specification
        //with anonymous type T
        //wherever appropriate
        //……..
};
```

Example:

```
int size=3;
template<class T>
class vector
{
  T* v;
  int size;
  public:
    vector()
    {
```

```
    v=new T[size];
    for(int i=0;i<3;i++)
      v[i]=0;
  }
  vector(T* a)
  {
    for(int i=0;i<size;i++)
      v[i]=a[i];
  }
  T operator *(vector &y)
  {
    T sum=0;
    for(int i=0;i<size;i++)
      sum+=this->v[i]*y.v[i];
    return sum;
  }
};
```

Class Templates with Multiple Parameters:

- More than one generic data type in a class template.
- It is declared as a comma separated list within the template specification .

Syntax:

```
template <class T1, class T2,...,class Tn>
class class-name
{
        ......
        ......
        ......//body of the class
};
```

Program:

```
#include<iostream.h>
template<class T1, class T2>
class Test
{
  T1 a;
  T2 b;
  public:
    Test(T1 x, T2 y)
    {
      a=x;
       b=y;
    }
    void show()
    {
       cout<<"\na : "<<a<<"\nb : "<<b;
    }
};
void main()
{
  Test <float, int> t1(1.23,123);
  Test <int, char> t2(100,'M');
  t1.show();
  t2.show();
}
```

Function templates

- Defining function Templates that could be used to create a family of functions with different argument types.

Syntax:

```
template <class T>
return-type function-name(argument of type T)
{
        //……
        //body of function
        //with type T
        //wherever appropriate
        //……..
}
```

- The function template syntax is similar to that of the class template expect that defining functions instead of classes.
- Use template parameter T as and when necessary in the function body and its argument list.

Program:

```
#include<iostream.h>
template<class T>
void swap(T &x, T &y)
{
  T temp=x;
  x=y;
  y=temp;
}
void fun(int m,int n,float a,float b)
{
  cout<<"\nm and n before swap: "<<m<<" "<<n;
  swap(m,n);
  cout<<"\nm and n after swap: "<<m<<" "<<n;
  cout<<"\na and b before swap: "<<a<<" "<<b;
```

```
   swap(a,b);
   cout<<"\na and b after swap: "<<a<<" "<<b;
}
void main()
{
   fun(100,200,11.53,33.44);
}
```

Function Templates with Multiple Parameters:

- Use more than one generic data type in the template statement using a comma-separated list.

Syntax:

```
template <class T1, class T2,...,class Tn>
return-type function-name(arguments of types T1,T2,....)
{
        ......
        ......
        ......//body of the function
}
```

Program:

Overloading of Template Functions:

- A template function may be overloaded either by template functions or ordinary functions of its name.
- The overloading resolution is accomplished as follows:
  - Call an ordinary function that has an exact match.
  - Call a template function that could be created with an exact match.
  - Try normal overloading resolution to ordinary functions and call the one that matches.

- An error is generated if no match is found.
- No automatic conversions are applied to arguments on the template functions.

Program:

```
#include<iostream.h>
template<class T>
void display(T x)
{
  cout<<"\nTemplate method : "<<x;
}
void display(int x)
{
  cout<<"\nExplicit method : "<<x;
}

void main()
{
  display(11.53);
  display(44);
  display("welcome");
}
```

Member function templates

- All the member functions were defined as inline is not necessary.
- Define members outside that class is also possible.
- The member function of the template classes are parameterized by the type arguments and functions must be defined by the function templates.

Syntax:

```
template <class T>
return-type class-name<T>:: function-name(argument list)
{
        ……
        ……
        ……//body of the function
}
```

Example:

```
template<class T>
class vector
{
   T* v;
   int size=3;
   public:
      vector(int m);
      vector(T* a);
      T operator*(vector &y);
};
template<class T>
vector<T>::vector(int m)
{
   v=new T[size];
   for(int i=0;i<size;i++)
       v[i]=0;
}
template<class T>
vector<T>::vector(T* a)
{
   for(int i=0;i<size;i++)
```

```
    v[i]=a[i];
}
template<class T>
vector<T>::operator *(vector &y)
{
  T sum=0;
  for(int i=0;i<size;i++)
    sum+=this->v[i]*y.v[i];
  return sum;
}
```

Exception handling

Exceptions are run-time anomalies, such as division by zero, that require immediate handling when encountered by your program. The C++ language provides built-in support for raising and handling exceptions. With C++ exception handling, your program can communicate unexpected events to a higher execution context that is better able to recover from such abnormal events. These exceptions are handled by code that is outside the normal flow of control

The C++ language provides built-in support for handling anomalous situations, known as exceptions, which may occur during the execution of your program. The try, throw, and catch statements implement exception handling. With C++ exception handling, your program can communicate unexpected events to a higher execution context that is better able to recover from such abnormal events. These exceptions are handled by code that is outside the normal flow of control. The Microsoft

C++ compiler implements the C++ exception handling model based on the ANSI C++ standard.

The following syntax shows a try block and its handlers:

```
try {
   // code that could throw an exception
}
[ catch (exception-declaration) {
   // code that executes when exception-declaration is thrown
   // in the try block
}
[catch (exception-declaration) {
   // code that handles another exception type
} ] . . . ]
// The following syntax shows a throw expression:
throw [expression]
```

C++ also provides a way to explicitly specify whether a function can throw exceptions. You can use exception specifications in function declarations to indicate that a function can throw an exception. For example, an exception specification throw(...) tells the compiler that a function can throw an exception, but doesn't specify the type, as in this example:

```
void MyFunc() throw(...) {
   throw 1;
}
```

Sample Programs in c++

Prime Number

```
#include<iostream.h>
#include<conio.h>
int main()
{
clrscr();
int st_no,end_no,div,no_div=0;
cout<<"Enter the starting no ";
cin>>st_no ;
cout<<"Enter the enging no ";
cin>>end_no ;

        while(st_no<=end_no)
        {  div=st_no;
                no_div=0;

                while(div>=1)
                {
                        if(st_no%div==0)
                        {
                        no_div=       no_div+ 1    ;
                        }
                        div--;
                }

if(no_div<=2)
{
cout<<st_no<< "  IS  PRIME"<<endl<<endl;
```

```
}
st_no++;
}
return 0;
}
```

String Program

```
#include <iostream>
#include <string>
#include <fstream>
#include <conio.h>
using namespace std;
int main()
{
ifstream file;
string s, city, bigstring, substring;
int count = 0;
file.open("c:\\cities.txt");
cout << "Enter all or part of a city name: ";
        getline(cin,city);
        if (bigstring.find(substring) != -1)
        while (getline(file,s))
{
        cout << s << endl;
        count++
        }
        cout << "There were " << count << " matches in the file" << endl;
getch();
        return 0;
```

```
}
```

Palindrome

```
#include <iostream>
#include <deque>
#include <string>
#include <cctype>
using namespace std;
int main()
{
	string input;
	deque<string> stackOne;
	deque<string> stackTwo;
	cout << "Enter a text . Do not include spaces or punctuation.\n";
	getline(cin, input);
	stackOne.push_front(input);
	while(!stackOne.empty())
	{
		input = stackOne.front();//retrieve the user entered input
		stackTwo.push_front(input);  // put input into stack two at
the front
	}
	if(stackOne == stackTwo)
	{
		cout << "It is a palindrome." << endl;
	}
	else
		cout << "It is not a palindrome." << endl;
	return 0;
}
```

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT: : OBJECT ORIETNED PROGRAMMING WITH C++
SEMESTER : I
SUBJECT CODE: 17CCP104 **CLASS : I M.COM CA**
UNIT 5

S.NO	QUESTIONS	OPTION 1	OPTION 2	OPTION 3	OPTION 4	ANSWER
1	Storing a class definition in a separate file is an example of	polymorphism	name mangling	implementation hiding	inheritance	implementation hiding
2	Which of the following ways are legal to access a class data member using this pointer?	this->x	this.x	*this.x	*this-x	this->x
3	When you declare a pointer, you must give it a _____	type	type and name	type, name, and value	name and value	type and name
4	An address is a _____, while a pointer is a _____	array, variable	constant, variable	variable, position	variable, location	variable, location
5	The file iostream includes _____	the declarations of the basic standard input-output library	the streams of includes and outputs of program effect	mathematical functions are included	string functions are included	the declarations of the basic standard input-output library
6	eof( ) is the function used for _____	asserting no errors in a file	appending data to a file	counting the amount of data in a file	checking for end of file	checking for end of file
7	Which of the following header files is not used for file handling?	Ofstream.h	Fstream.h	Ifstream.h	Consolе.h	Consolе.h
8	ios::app is a file open mode for _____	opening a file	open the file for input	open file for output	seek to end of file before each write	seek to end of file before each write
9	ios::trunc is a file open mode for _____	opening a file with its existing contents	open file for output	seek to end of file before each write	delete the file's current contents upon file open	delete the file's current contents upon file open
10	_____ is a member function of fstream class	put	find	resize	replace	put
11	Which of the following is not a member function of fstream class?	flush	put	find	write	find
12	A direct access file is _____	a file in which recoreds are arranged in a way they are inserted in a file	a file in which records are arranged in a particular order	files which are stored on a direct access storage medium	a file in which records are arranged in sorted order	files which are stored on a direct access storage medium
13	Which is referred by pointers to member?	static members of class objects	Non-static members of class objects	Refering to whole class	the address of a static member	Non-static members of class objects
14	What should be used to point to a static class member?	Smart pointer	Dynamic pointer	Normal pointer	Pointer variable	Normal pointer
15	Which is the best design choice for using pointer to member function?	Interface	Class	Structure	Object	Interface
16	What is the operation for .*?	It combines the first operand and the second operand	It seperates the first operand and the second operand	It reduces the data size	It increase the data size	It combines the first operand and the second operand
17	Which member function is used to determine whether the stream object is currently associated with a file?	is_open	buf	string	char	is_open
18	_____ is a stream connected to standard output	Cin	Gets	Out	Cout	Cout
19	The size of operator returns the size of variable or type in _____	bits	nibble	bytes	char	bytes
20	Which of the following gives the memory address of integer variable a ?	*a;	a;	&a;	address(a);	&a;
21	Passing a variable pointer as a constant _____	protects the contents pointed to by the pointer from change	eliminates the need to name the pointer in the function	eliminates the need to give the pointer a type in the function	causes a copy of the pointer to be produced in the function	protects the contents pointed to by the pointer from change
22	Recursive Functions _____	easier to code	executable faster than iterative ones	takes less main storage space	necessary to solve a certain class of problems	necessary to solve a certain class of problems
23	How "Late binding" is implemented in C++?	Using C++ tables	Using Virtual tables	Using Indexed virtual tables	Using polymorphic tables	Using Virtual tables
24	To which does the function pointer point to?	variable	constants	function	absolute variables	function
25	What is the default calling convention for a compiler in c++?	__cdecl	__stdcall	__pascal	__fastcall	__cdecl
26	What is meaning of following declaration? int(*ptr[5])();	ptr is pointer to function	ptr is array of pointer to function	ptr is pointer to such function which return type is array	ptr is pointer to array of function	ptr is array of pointer to function
27	Which keyword is used to define the user defined data types?	def	union	typedef	type	typedef
28	What is the syntax of user-defined data types?	typedef_existing data type_new name	typedef_new name_existing data type	def_existing data type_new name	def_new name _new existing data type	typedef_existing data type_new name
29	How many types of user-defined data type are in c++?	1	2	3	4	3
30	Syntax for Pure Virtual Function is _____	virtual void show()==0	void virtual show()==0	virtual void show()=0	void virtual show()=0	virtual void show()=0
31	_____ is a member function that is declared within a base class and redefined by derived class.	static function	virtual function	friend function	const member function	virtual function
32	When a virtual function is redefined by the derived class, it is called_____.	Overloading	Overriding	Rewriting	operator overloading	Overriding
33	If a class contains pure virtual function, then it is termed as _____ .	Virtual class	Sealed class	Pure Local class	Abstract Class	Abstract Class
34	A virtual function that has no definition within the base class is called_____.	Pure virtual function	Pure static function	Pure Const function	Friend function	Pure virtual function
35	External documentation includes _____	a printout of the program's code	flowcharts	IPO charts	pseudocode	a printout of the program's code
36	A variable's _____ indicates how long the variable remains in the computer's memory	area	extent	lifetime	reach	lifetime
37	The function whose prototype is void getData(Item *thing); receives	a pointer to a structure	a reference to a structure	a copy of a structure	structure variable	a pointer to a structure
38	You may override the class access specifier for _____	public members	public and protected members	any specific class members you choose	no class members	any specific class members you choose
39	The number of structures than can be declared in a single statement is	one	two	three	unlimited	unlimited
40	A derived class may also be called a	subclass	super class	parent class	base class	subclass
41	Any output manipulator function you create _____	should take as an argument an instance of ostream as a reference	should return void	must be a member function of the ostream class	must inherit ostream	should take as an argument an instance of ostream as a reference
42	Which of the following stream manipulators advances the cursor to the next line on the computer screen?	adin	advin	odlin	endl	endl
43	6.5 is a _____ constant	character literal	named literal	numeric literal	string literal	numeric literal
44	The time and memory involved in calling a function represent the function's _____	prototype	overhead	cost	burden	burden
45	A pointer is _____	the address of a variable	an indication of the variable to be accessed next	a variable for storing addresses	the data type of an address variable	a variable for storing addresses
46	Block statements are also called as _____ statements	group	multiple	compound	logical	compound
47	_____ statements are simply a group of related statements that are treated as a single unit	Block	Multiple	Related	Logical	Block
48	The code and data are called _____ of the class	instances	instance variables	members	object	members
49	The keyword _____ is used to declare a generic function	generic	template	virtual	friend	template
50	_____ function is a short function that gets or sets the value of a private instance variable	Inline	Accessor	Member	Recursive	Accessor
51	Each object has its own copies of the class _____	data	code	function	instance variables	instance variables
52	A reference parameter is declared by preceding with _____	*	&	&&	_	&
53	The elements of an array can be accessed by providing integer expression called _____	superscripts	elements	values	subscripts	subscripts
54	The keyword virtual indicates that _____	a derived class has public access to a base class	more than one base class exists	a base class should be used only once in inheritance	a derived class should have more than one base class constructed	a base class should be used only once in inheritance
55	Which of the following is an access specifier?	particular	shielded	protected	safe	protected
56	The contents of two pointers that point to adjacent of type float differ by _____	one bytes	two bytes	three bytes	four bytes	four bytes
57	The code class Descendant : virtual public Ancestor indicates that _____	the members of Ancestor will be included more than once in Descendant	the members of Ancestor will be included only once in Descendant	the members of Descendant will be included more than once in Ancestor	the members of Descendant will be included only once in Ancestor	the members of Ancestor will be included only once in Descendant
58	The arguments that determine the state of the cout object are called _____	classes	manipulators	format flags or state flags	state controllers	format flags or state flags
59	A measure of the strength of the connection between two functions is _____	cohesion	coupling	dependence	subjection	coupling
60	We can output text to an object of class ofstream using the insertion operator A« because	the ofstream class is a stream	the insertion operator works with all classes	we are actually outputting to cout	the insertion operator is overloaded in ofstream	the insertion operator is overloaded in ofstream

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 21
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT : OBJECT ORIENTED PROGRAMMING WITH C++
SEMESTER : I
SUBJECT CODE: 17CCP104 CLASS : I M.COM CA

POSSIBLE QUESTIONS – UNIT V

PART A (1 Mark)

(Online Examinations)

PART B (2 Marks)

1. Define Pointer to object
2. Define to Derived classes
3. What is Virtual Functions
4. How pointer to derived Classes used in a program
5. Define File
6. List out File Stream Operation'
7. How will you Open and Close a file
8. What is File Pointer
9. What are Sequential I/O operations
10. How files are manipulated.
11. What are streams?

PART C (6 Marks)

1. Describe on file operations.
2. Explain Virtual Functions with example.
3. Describe about Pointers to Object
4. Describe about Pointers to Derived Classes.
5. Explain file stream Operations.
6. Explain file pointers and their manipulators.
7. List out and explain Sequential I/O operations.
8. Write a program using file operation (to open and close a file).
9. Describe file pointers with example.
10. Differentiate between pointers to object and pointers to derived class.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2017 onwards)
DEPARTMENT OF COMMERCE (CA)

SUBJECT : **OBJECT ORIENTED PROGRAMMING WITH C++**
SEMESTER : I
SUBJECT CODE: **17CCP104** **CLASS** : **I M.COM CA**

UNIT II

Tokens and Control Structures- Tokens , Expressions and Control Structures – Token – Keywords – Identifiers – Basic and User – Defined Data Types – Operators in C++ - Operator Overloading- Operator Precedence – Control Structure Functions in C++ - the Main Function – Call By Reference – Return by Reference – In line Function – Function Overloading.

Unit – II

Tokens:

Smallest unit of a Program is called Token.

- Keywords
- Identifiers
- Constants
- Strings
- Operators
- Special Symbols

Keywords

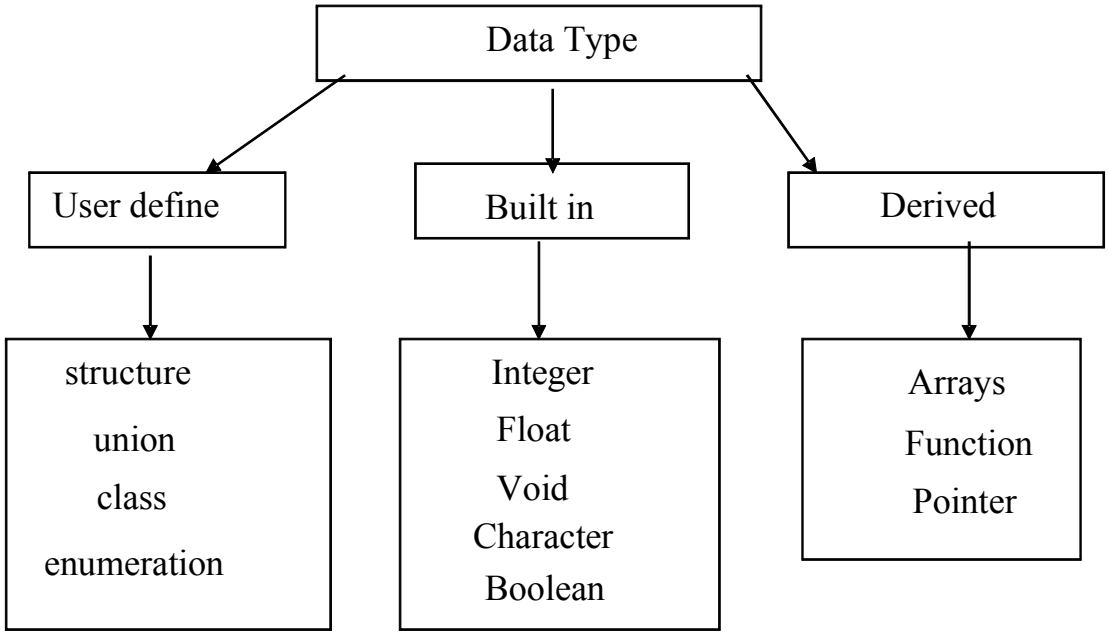
- Keywords are reserved words.
- Has its predefine meaning.
- C++ consist of c keywords and additional keywords of its own

Keyword List:

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	

Data Types:



Integer Type : Integer data type are like whole numbers, they also include negative numbers but does not support decimal numbers.

Type	**Storage size**	**Value range**
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295

short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Float-point Type : Float data type allows user to store decimal values in a variable.

Type	**Storage size**	**Value range**	**Precision**
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Character Type : Character data type is used to store only one letter, digit, symbol at a time.

Type	**Storage size**	**Value range**

char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127

Array : Array is a collection of similar data type. A single variable can hold only one value at a time, If we want a variable to store more than one value of same type we use array.

Pointers : A normal variable is used to store value. A pointer variable is used to store address / reference of another variable.

Variables

Variables are used to store values. variable name is the name of memory location where value is stored. It must be alphanumeric, only underscore is allowed in a variable name. It is composed of letters, digits and only underscore. It must begin with alphabet or underscore. It cannot be begin with numeric.

Declaration of Variable

Declaration will allocate memory for specified variable with garbage value.

Syntax :

```
Data-Type Variable-name;
```

Examples :

```
int a;
float b;
char c;
```

Variable (Identifiers):

- Identifiers are user define name space
- It change its value during the execution of the program
- It refers the names of variables, functions, arrays, classes, etc

Rules:

1. Only alphabetic characters, digits and underscores are permitted.
2. Cannot start with digits.
3. Uppercase and lowercase are distinct
4. Keyword should not be as a variable name.

Example:

A, welcome

Initialization of Variable

Initialization means assigning value to declared variable. Every value will overwrite the previous value.

Examples :

```
a = 10;
b = 4.5;
c = 'a';
```

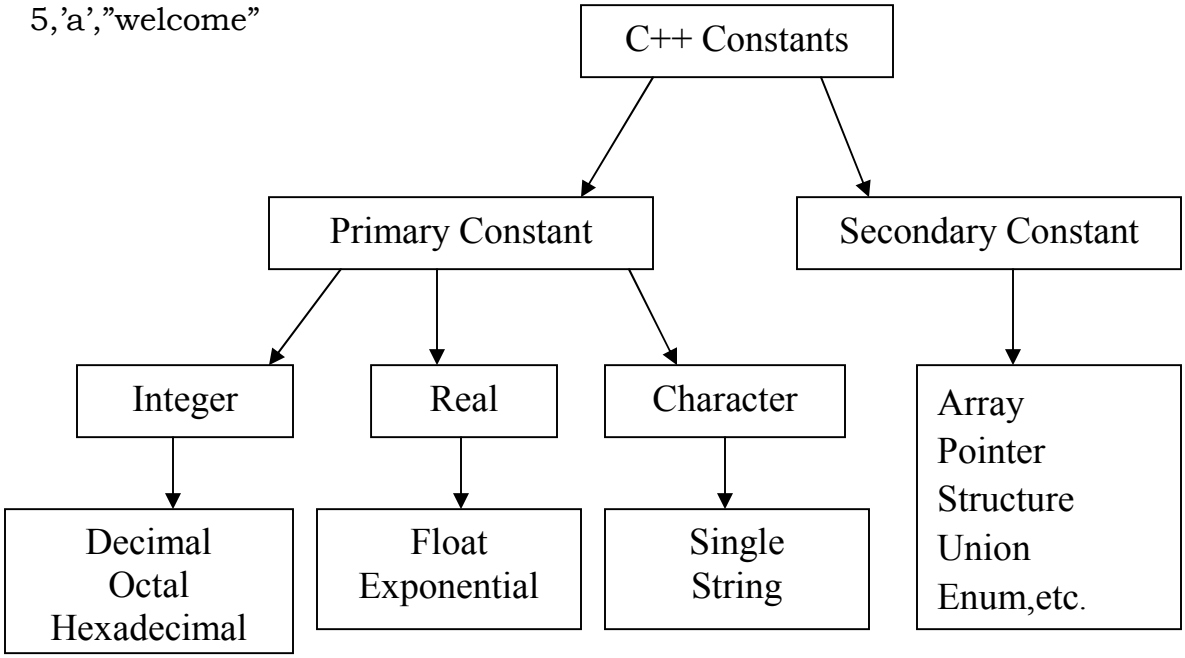
Character value must be enclosed with single quotes.

Constants:

- It does not change its value during the execution of the program

Example:

5,'a',"welcome"



Operators:

- Arithmetic operators
- Assignment operators
- Increment/Decrement
- Comparison operators
- Logical operators
- Bitwise operators

Assignment operator is used to copy value from right to left variable.

Operator	**Name**	**Description**	**Example**

=	Equal sign	Copy value from right to left.	X = Y, Now both X and Y have 5
+=	Plus Equal to	Plus Equal to operator will return the addition of right operand and left operand.	X += Y is similar to X = X + Y, now X is 7
-=	Minus Equal to	Minus Equal to operator will return the subtraction of right operand from left operand.	X -= Y is similar to X = X - Y, now X is 3
*=	Multiply Equal to	Multiply Equal to operator will return the product of right operand and left operand.	X *= Y is similar to X = X * Y, now X is 10
/=	Division Equal to	Division Equal to operator will divide right operand by left operand and return the quotient.	X /= Y is similar to X = X / Y, now X is 2.5
%=	Modulus or Mod Equal to	Modulus Equal to operator will divide right operand by left operand and return the mod (Remainder).	X %= Y is similar to X = X % Y, now X is 1

Arithmetic operators are used for mathematical operations.

Operator	Name	Description	Example
+	Plus	Return the addition of left and right operands.	(X + Y) will return 7
-	Minus	Return the difference b/w right operand from left operand.	(X - Y) will return 3
*	Multiply	Return the product of left and right operands.	(X * Y) will return 10
/	Division	Return the Quetiont from left operand by right operand.	(X / Y) will return 2(both are int, int doesn't support decimal)
%	Modulus or Mod	Return the Modulus (Remainder) from left operand by right operand.	(X % Y) will return 1

Relational operators are used for checking conditions whether the given condition is true or false. If the condition is true, it will return non-zero value, if the condition is false, it will return 0.

Operator	Name	Description	Example
>	Greater then	Check whether the left operand is greater then right operand or not.	(X > Y) will return true
<	Smaller then	Check whether the left operand is smaller then right operand or not.	(X < Y) will return false
>=	Greater then or Equal to	Check whether the left operand is greater or equal to right operand or not.	(X >= Y) will return true
<=	Smaller then or Equal to	Check whether the left operand is smaller or equal to right operand or not.	(X <= Y) will return false
==	Equal to	Check whether the both operands are equal or not.	(X == Y) will return false
!=	Not Equal to	Check whether the both operands are equal or not.	(X != Y) will return true

Logical operators are used in situation when we have more then one condition in a single if statement.

Operator	Name	Description	Example
&&	AND	Return true if all conditions are true, return false if any of the condition is false.	if(X > Y && Y < X) will return true
||	OR	Return false if all conditions are false, return true if any of the condition is true.	if(X > Y || X < Y) will return true
!	NOT	Return true if condition if false, return false if condition is true.	if(!(X>y)) will return false

The conditional operator is also known as **ternary operator**. It is called ternary operator because it takes three arguments. First is condition, second and third is value. The conditional operator check the condition, if condition is true, it will return second value, if condition is false, it will return third value.

Syntax :

```
val = condition ? val1 : val2;
```

Example :

```
void main()
{
```

```
        int X=5,Y=2,lrg;

        lrg = (X>Y) ? X : Y;

        cout << "\nLargest number is : " << lrg;
    }
```

Output :

```
        Largest number is : 5
```

Binary operators are those operators that works with at least two operands such as (Arithmetic operators) +, -, *, /, %.

Unary operators are those operators that works with singal operands such as (Increment or Decrement operators) ++ and --.

Special Operators:

- ::- Scope resolution operator
- >>-Insertion Operator
- <<-Extraction Operator
- ::*- Pointer-to-member decelerator

- ->*- Pointer-to-member operator
- .*- Pointer to member operator
- new-Memory management operator
- delete- Memory release operator
- endl- Line feed operator
- sew- Memory allocation operator
- setw- Field width operator

Scope Resolution Operator:

:: - Used to access values or methods.

Syntax:

::variable name;

:: function name;

Examples:

::a;

::add();

Program:

```
#include<iostream.h>
int m=10;
void main()
{
  int m=20;
  {
    int m=40;
    cout<<"Value of m in inner block:"<<m;
    cout<<"Value of m in outter block:"<<::m;
  }
  cout<<"Value of m in inner block:"<<m;
  cout<<"Value of m in outter block:"<<::m;
```

}

Manipulator:

endl:

used instead of ”\n”.

Example:

cout<<“welcome to csc”<<endl;

setw:

used for neat alignment during display.

Syntax:

setw (int value)

Example:

cout<<setw(5);

Program:

```
#include<iostream.h>
#include<iomanip.h>
#include<conio.h>
void main()
{
   int m1=50,m2=500,m3=5000;
   clrscr();
   cout<<setw(5)<<"m1:"<<setw(5)<<m1<<endl;
   cout<<setw(5)<<"m2:"<<setw(5)<<m2<<endl;
   cout<<setw(5)<<"m3:"<<setw(5)<<m3<<endl;
}
```

Output:

```
 m1:   50
 m2:  500
 m3: 5000
```

Type Cast:

- Convert the data type of a variable to some other data type variable

Syntax:

(type name) expression //c

type name (expression) //c++

Example:

float (i);

Decision making statements

if ..else, jump, goto, break, continue- switch case statements

Control Structure:

- Sequence structure
- Selection structure
- Loop structure

Selection Structure:

- if statement
- if…else
- Nested if
- if… else ladder
- switch statement

if Statement:

if statement takes condition in parenthesis and a block of statements within braces. If condition is true, it will return non-zero value, and statements given in if block will get execute.

Syntax:

```
if(condition)
{
        True Block;
 }
```

Next Statement;

Example:

```
if(i>10)
   cout<<"i greater than 10";
```

if ...else Statement

if statement takes condition in parenthesis and a block of statements within braces. If condition is true, it will return non-zero value, and statements given in if block will get execute. If condition is false, it will returns zero, and statements given in else block will get execute.

Syntax:

```
if(condition)
{
        True Block;
}
else
{
        False Block;
}
Next Statement;
```

Example:

```
if(i>10)
   cout<<"i greater than 10";
else
   cout<<"i less than 10";
```

Nested if Statement:

In nested if-else, one if-else statement contains another if-else statement.

Syntax:

```
if(condition 1)
```

```
    {
            if(condition 2)
            {
                    True block
            }
            else
            {
                    False block condition 2;
            }
     }
     else
       {
            False block condition 1;
        }
     Next Statement;
```

Example:

```
if(m1>40)
{
            if(m2>40)
            {
                    cout<<"pass";
            }
            else
            {
                    cout<<"Fail";
            }
}
else
            {
```

```
                cout<<"Fail";
        }
```

if... else ladder:

if-else ladder is used for checking multiple conditions, if the first condition will not satisfy, compiler will jump to else block and check the next condition, whether it is true or not and so on.

Syntax:

```
if(condition 1)
{
        True block-1
}
else if(condition 2)
{
        True block -2;
}
else
{
        False block;
}
 Next Statement;
```

Example:

```
if(a>b)
{
        if(a>c)
        {
                cout<<"A is Greatest";
        }
        else
        {
```

```
                cout<<"C is Greatest";
        }
}
else
if(b>c)
{
        cout<<"B is Greatest";
}
else
{
        cout<<"C is Greatest";
}
```

Switch Case:

Syntax:

```
        switch(expression)
        {
                case exp1:
                        Statements
                case exp2:
                        Statements
                ……….
                ……….
                default:
                        Statements
        }
```

Example

```
i=4;
switch(i)
{
```

```
        case 1:
                cout<<"one";
        case 2:
                cout<<"two";
        case 3:
                cout<<"three";
        default:
                cout<<"Wrong Choice";
}
```

do-while – while statement, for statement

Loop Structure

Entry control:

Entry control Structure checks the condition First and the Statement is executed

- while loop
- for loop

Exit control:

Exit control Structure First the Statement is Executed and then checks the condition

- do… while

While Loop:

While loop is also called entry control loop because, in while loop, compiler will 1st check the condition, whether it is true or false, if condition is true then execute the statements.

Syntax:

```
while(Condition)
{
        Statement Block
}
```

Example:

```
while(i<5)
{
        cout<<"Welcome";
        i++;
}
```

For Loop:

In for loop has initialization, condition and increment/decrement all together. Initialization will be done once at the beginning of loop. Then, the condition is checked by the compiler. If the condition is false, for loop is terminated. But, if condition is true then, the statements are executed until condition is false.

Syntax:

```
for(initialization ; condition checking ; Increment/Decrement)
{
        Statement Block
}
```

Example:

```
for(i=1;i<5;i++)
{
        cout<<"Welcome";
}
```

Do..While

The do-while loop is also called exit control loop because, in do-while loop, compiler will 1st execute the statements, then check the condition, whether it is true or false.

Syntax:

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

Example:

```
 do
 {
        cout<<"Welcome";
        i++;
 }while(i<5);
```

Difference b/w while loop and do-while loop

while loop	**do-while loop**
It is entry control loop.	It is exit control loop.
In this loop condition is checked before loop execution.	In this loop condition is checked at the end of loop.
It will never execute loop if condition is false.	It will executes loop at least once when the initial condition is false.
There is no semicolon at the end of while statement	There is semicolon at the end of while statement.

Jump Statements in C++

Jump statements are used to interrupt the normal flow of program.

Types of Jump Statements

- Break
- Continue
- GoTo

Break Statement

The break statement is used inside loop or switch statement. When compiler finds the break statement inside a loop, compiler will abort the loop and continue to execute statements followed by loop.

Example of break statement

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

```
void main()
{
      int a=1;

      while(a<=10)
       {
          if(a==5)
            break;

           cout << "\nStatement " << a;
          a++;
       }
           cout << "\nEnd of Program.";
}
```

Output :

```
Statement 1
Statement 2
Statement 3
Statement 4
End of Program.
```

Continue Statement

The continue statement is also used inside loop. When compiler finds the break statement inside a loop, compiler will skip all the followling statements in the loop and resume the loop.

Example of continue statement

```
#include<iostream.h>

void main()
{
    int a=0;

    while(a<10)
    {

        a++;

        if(a==5)
          continue;

         cout << "\nStatement " << a;

    }
         cout << "\nEnd of Program.";
}
```

Output :

```
Statement 1
Statement 2
Statemnet 3
Statement 4
```

```
Statement 6
Statement 7
Statement 8
Statement 9
Statement 10
End of Program.
```

Goto Statement

The goto statement is a jump statement which jumps from one point to another point within a function.

Syntax of goto statement

```
goto label;
    - - - - - - - - - -
    - - - - - - - - - -
label:
    - - - - - - - - - -
    - - - - - - - - - -
```

In the above syntax, label is an identifier. When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code after it.

Example of goto statement

```
#include<iostream.h>

void main()
```

```
{
        cout << "\nStatement 1.";
        cout << "\nStatement 2.";
        cout << "\nStatement 3.";

        goto last;

        cout << "\nStatement 4.";
        cout << "\nStatement 5.";

        last:

        cout << "\nEnd of Program.";
}
```

Output :

```
Statement 1.
Statement 2.
Statement 3.
End of Program.
```

Function in C++

A function is a block of codes that performs a specific task and may return value. The main() function is the first user defined function invoked by the compiler. While it is possible to write any code within main function, it leads number of problems. The program may become too large and complex and it is difficult to test, debugg and maintain the complex code. For that reason, We use function to place independent code in separate modules called function or

subprogram. In order to make a program using function, we need to perform the followling three steps.

- Function declaration
- Function definition
- Function call

Function declaration

Like variables, all the functions must be declared. Function declaration statement includes function name, what function will take and what function will return.

Syntax :

```
return-type function-name(argument list);
```

return-type : type of value function will return.

function-name : any valid C++ identifier.

argument list : represents the type and number of value function will take, values are sent by the calling statement.

Example for declaration of function

If we want to return the sum of two integer numbers and function will take two numbers as argument then the function declaration statement will be:

```
int Add(int, int);
```

Function definition

Function definition includes the actual working or implementation.

Syntax for defining function

```
return-type function-name(argument list)
{
        - - - - - - - - - -
        body of function
        - - - - - - - - - -
}
```

The body of function contains the number of statements to perform specific task.

Example for definition of function

The body of function for calculating sum of two integer numbers.

```
int Add(int x,int y)
{
    int sum;

     sum = x + y;

    return sum;
}
```

Function call or Function invoke

To execute function we must call it. A function can be called or invoked by using function name followed by list of arguments (values) that function definition will recieve to perform task.

Syntax for calling or invoke function

```
var = function-name(val1,val2...n);
```

var can be any variable that will recieve value returning from function definition.

Example for calling or invoke function

Considering the above example, function calling statement should be :

```
int rs;
rs = Add(10,20);        //calling statement
cout << "\nThe sum is : " << rs;
```

Passing argument to a function

Like normal variable, pointer variable can be passed as function argument and it can return from function.

There are two approaches to passing argument to a function:

- Call by Value
- Call by Reference/Address

Call by Value

In this approach, the values are passed as function argument to the definition of function.

Example of call by value

```
#include<iostream.h>

void main()
{
```

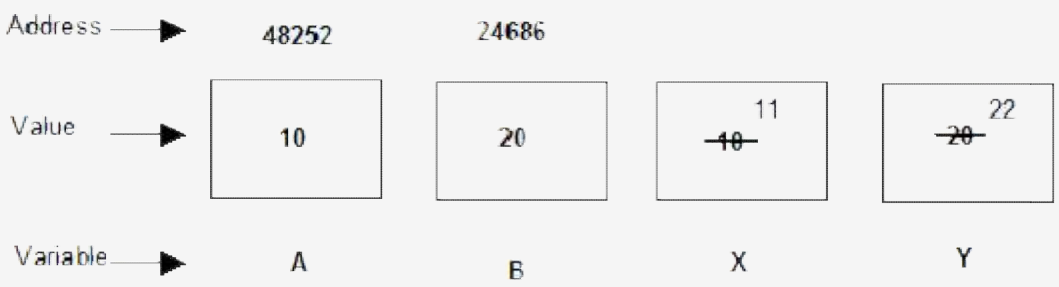
```
        int A=10,B=20;

        cout << "\nValues before calling";
        cout << "\nA : " << A;
        cout << "\nB : " << B;

        fun(A,B);                       //Statement     1

        cout << "\nValues after calling";
        cout << "\nA : " << A;
        cout << "\nB : " << B;



    }

    void fun(int X,int Y)               //Statement     2
    {
        X=11;
        Y=22;
    }
```

Output :

```
        Values before calling
        A : 10
        B : 20
        Values after  calling
        A : 10
```

```
B : 20
```

In the above example, statement 1 is passing the values of A and B to the calling function fun(). fun() will recieve the value of A and B and put it into X and Y respectively. X and Y are value type variables and are local to fun(). Any changes made by value type variables X and Y will not effect the values of A and B.



Call by Reference

In this approach, the references/addresses are passed as function argument to the definition of function.

Example of call by reference

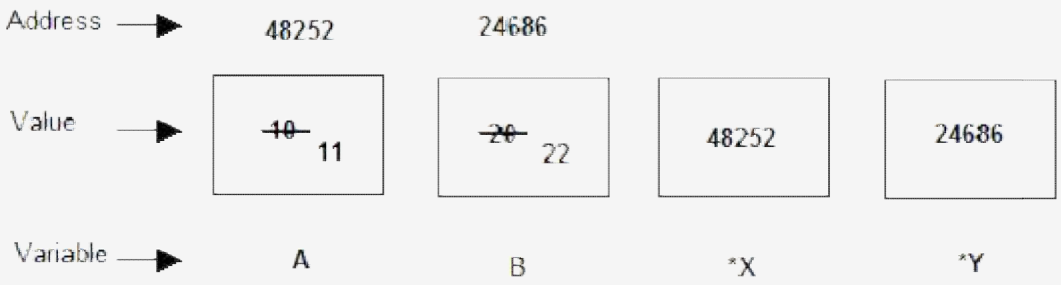
```
#include<iostream.h>

void main()
{
    int A=10,B=20;
```

```
        cout << "\nValues before calling";
        cout << "\nA : " << A;
        cout << "\nB : " << B;

        fun(&A,&B);                    //Statement    1

        cout << "\nValues after calling";
        cout << "\nA : " << A;
        cout << "\nB : " << B;


    }

    void fun(int *X,int *Y)            //Statement    2
    {
        *X=11;
        *Y=22;
    }
```

Output :

```
        Values before calling
        A : 10
        B : 20
        Values after  calling
        A : 11
        B : 22
```

In the above example, statement 1 is passing the reference of A and B to the calling function fun(). fun() must have pointer formal arguments to recieve the reference of A and B. In statement 2 *X and *Y is recieving the reference A and B. *X and *Y are reference type variables and are local to fun(). Any changes made by reference type variables *X and *Y will change the values of A and B respectively.



Difference between Call by Value and Call by Reference.

Call by Value	**Call by Reference**
The actual arguments can be variable or constant.	The actual arguments can only be variable.
The values of actual argument are sent to formal argument which are normal variables.	The reference of actual argument are sent to formal argument which are pointer variables.
Any changes made by formal	Any changes made by formal

arguments will not reflect to actual arguments.	arguments will reflect to actual arguments.

Inline functions

One of the advantages of using function is to save memory space by making common block for the code we need to execute many times. When compiler invoke / call a function, it takes extra time to execute such as jumping to the function definition, saving registers, passing value to argument and returning value to calling function. This extra time can be avoidable for large functions but for small functions we use inline function to save extra time.

When we make an inline function, compiler will replace all the calling statements with the function definition at run-time.

- Expand itself code during the execution of the program.
- Keyword: **inline**

Syntax:

```
inline function
{
    function body
 }
```

Example of inline function

```
#include<iostream.h>
#include<conio.h>
inline  int add(int a,int b)
{
  return(a+b);
```

```
}
void main()
{
   int m1,m2;
   clrscr();
   cout<<"Enter the first number:";
   cin>>m1;
   cout<<"Enter the Second number:";
   cin>>m2;
   cout<<"Addition Result:"<<add(m1,m2)<<endl;
}
```

Function overloading

More than one function with same name, with different signature in a class or in a same scope is called function overloading. Signature of function includes:

- Number of arguments
- Type of arguments
- Sequence of arguments

Example of function overloading

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

class CalculateArea
{
```

```
public:
void Area(int r)          //Overloaded Function 1
{
     cout<<"\n\tArea of Circle is : "<<3.14*r*r;
}

void Area(int l,int b)          //Overloaded Function 2
{
     cout<<"\n\tArea of Rectangle is : "<<l*b;
}

void Area(float l,int b)          //Overloaded Function 3
{
     cout<<"\n\tArea of Rectangle is : "<<l*b;
}

void Area(int l,float b)          //Overloaded Function 4
{
     cout<<"\n\tArea of Rectangle is : "<<l*b;
}
```

```
};


void main()
{

      CalculateArea C;

      C.Area(5);        //Statement 1
      C.Area(5,3);      //Statement 2
      C.Area(7,2.1f);   //Statement 3
      C.Area(4.7f,2);   //Statement 4

}
```

Output :

```
      Area of Circle is : 78.5
      Area of Rectangle is : 15
```

```
Area of Rectangle is : 14.7

Area of Rectangle is : 29.4
```

Example Program 2:

```
#include<iostream.h>
#include<conio.h>
int volume(int);
double volume(double,int);
long volume(long,int,int);
void main()
{
    clrscr();
    cout<<"Function with one argument:"<<volume(10)<<"\n";
    cout<<"Function with two argument:"<<volume(5.6,10)<<endl;
    cout<<"Function with three argument:"<<volume(56l,67,89)<<endl;
}
 int volume(int s)
{
  return(s*s*s);
}
double volume(double r,int h)
{
  return(3.14*r*r*h);
}
long volume(long l,int b,int h)
{
  return(l*b*h);
}
```