Course Code: BCS-031
Course Title: Programming in C++

Question 1:

(a) What is Object Oriented Programming? Explain its features with example.

Ans:-

Object-oriented programming (OOP) is a programming paradigm that represents the concept of "objects" that have data fields (attributes that describe the object) and associated procedures known as methods.

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 behaviour. For example, the hand can grip something or a *Student* (*object*) can give the name or address.

Features of oops.

Class and object :-A class is a *blueprint* that describes characteristics and functions of entity.. For example, the class Dog would consist of traits shared by all dogs. A class is collection of the properties and methods are called members. Object is an instance of class.

Data abstraction: - it is a process that provides only essential features by hiding its background details. A data abstraction is a simple view of an object that includes required features and hides the unnecessary details.

Data encapsulation is a mechanism of bundling the data, and the functions that use them and **data abstraction** is a mechanism of exposing only the interfaces and hiding the implementation details from the user. C++ supports the properties of encapsulation and data hiding through the creation of user-defined types, called **classes**. We already have studied that a class can contain **private, protected** and **public** members. By default, all items defined in a class are private.

For example:-

```
class pixeles
{
private:
int a,b,c;
public:
void sum()
{
}
};
```

Inheritance: -it is a process to create a new class from existing class or classes. Class 'B' is a sub class that is derived from class 'A'. Thus class 'A' is a parent class and class 'B' is a child class. It provides reusability of code.

Polymorphism: - Generally, it is the ability to appear in different forms. In oops concept, it is the ability to process objects differently depending on their data types. It's the ability to redefine methods for derived classes.

Advantage:-

- Reusability of code
- A single variable can hold multiple data types.
- Easy to maintenance

(b) Write a C++ program to create Matrix class. This class should have functions to find the sum and difference of two matrices. (9 Marks)

Ans:

```
#include<iostream.h>
#include<conio.h>
class mat
{

int m,k,n, c, d, first[10][10], second[10][10], sum[10][10],sub[10][10];
public:
void input();
void mat_sum();
void mat_diff();
};
void mat::input()
{
 cout << "Enter the number of rows and columns of matrix ";
   cin >> m >> n;
   cout << "Enter the elements of first matrix\n";
   for (  c = 0 ; c < m ; c++ )
      for ( d = 0 ; d < n ; d++ )
             cin >> first[c][d];

   cout << "Enter the elements of second matrix\n";

   for ( c = 0 ; c < m ;c++ )
      for ( d = 0 ; d < n ; d++ )
                cin >> second[c][d];
}
void mat::mat_sum()
{
cout<<"\nsum of matrix\n";
for ( c = 0 ; c < m ; c++ )
{
for ( d = 0 ; d < n ; d++ )
{

 sum[c][d] = first[c][d] + second[c][d];
 cout << sum[c][d] << "\t";
}
cout<<endl;
}
}
void mat::mat_diff()
{
cout<<"\nDifferences\n";
for ( c = 0 ; c < m ; c++ )
{

for ( d = 0 ; d < n ; d++ )
{
```

```
sub[c][d] = first[c][d] - second[c][d];
  cout <<sub[c][d] << "\t";
}
cout<<endl;
}
}

void main()
{

 mat tt;
tt.input();
clrscr();
tt.mat_sum();
tt.mat_diff();
getch();
}
```

(c) Explain the usage of the following C++ operators with the help of an example program.
(a) Relational Operator
(b) Logical Operators
(c) Scope resolution operator

Ans:
Relational Operator

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.

<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical operator

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Example of Relational Operator & Logical operator

```
#include<iostream.h>
void main()
{
int a,b;
cout<<"enter the number a & b";
if(a>=50 && b>=60)
cout<<"yes";
else
cout<<"no";
getch();
}
```

Scope resolution operator
Scope resolution operator (::) is used to define a function outside a class or when we want to use a global variable but also has a local variable with same name.
Example:

```
#include <iostream.h>

char c = 'a';     // global variable

void main() {
  char c = 'b';   //local variable

  cout << "Local c: " << c << "\n";
```

```
 cout << "Global c: " << ::c << "\n";  //using scope resolution operator

getch();

}
```

Question 2:
(a) Define the class Teacher with all the basic attributes such as Name, Department,Subjects,
date_of_ joining, years_of_experience etc. Define constructor(s), member functions display_detail() for displaying the Teacher details. Use appropriate access control specifiers in this program. Also inherit Post_Graduate_Teacher from Teacher class.
(9 Marks)
Ans:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
class teacher
{
 char n[10],dept[20],sub[15],doj[25];
 int exp;
 public:
 teacher(char*,char*,char*,char*,int);
 void disp_details();
};
teacher:: teacher(char a[],char b[],char c[],char d[],int e)
{
strcpy(n,a);
strcpy(dept,b);
strcpy(sub,c);
strcpy(doj,d);
exp=e;
}
void teacher::disp_details()
{
cout<<"Teacher name="<<n<<endl;
cout<<"Teacher department="<<dept<<endl;
cout<<"Teacher subject="<<sub<<endl;
cout<<"Teacher DOJ="<<doj<<endl;
cout<<"Teacher experience="<<exp<<endl;
}
class PG:public teacher
{
int duration;
char stream[30];

public:
PG(char x[],char y[],char z[],char p[],int m,int d,char s[]):teacher(x,y,z,p,m)
{
```

```
duration=d;
strcpy(stream,s);
}
void disp()
{
cout<<"PG Details=";
cout<<"\nDuration=" <<duration;
cout<<"\nStream="<<stream;
}
};

void main()
{
PG t("Ashok","Science","Physics","12-Dec-2000",13,3,"Business");
clrscr();
t.disp_details();
t.disp();
getch();
}
```

(b) Explain the following terms in the context of object oriented programming. Also explain
how these concepts are implemented in C++ by giving an example program for each. (6 Marks)
(a) Virtual Function
(b) Operator Overloading

Ans:
Virtual Function: - virtual function must be a member of some classes .it is declare by "virtual keyboard". It allows override the function in derived class. Virtual function cannot be static member. Virtual function cannot be inline.

```
 #include<iostream.h>
#include<conio.h>
class animal
{
public:
virtual void print()
{
cout<<"this is an animal";
}
};
class dog:public animal
{
public:
```

```
void print()
{
cout<<"this is A DOG";
}
};
class cat:public animal
{
public:
void print()
{
cout<<"this is a cat";
}
};
void main()
{
clrscr();
animal a;
dog b;
cat c;
animal *p=new animal();
p=&a;
p->print();
p=&b;
p->print();
p=&c;
p->print();
getch();
}
```

[b] Operator overloading: - operator overloading is used to give a special meaning to the commonly used operator (such s +,-,* etc_ with respect to class. by operator overloading we control or defined how an operator should operate on data with respect to a class.

```
#include<iostream.h>
#include<conio.h>
class std
{
int a,b;
public:
std()
{
}
std(int x, int y)
{
a=x;
b=y;
}

std operator+(std ob1)
{
std ob3;
ob3.a=ob1.a+a;
ob3.b=ob1.b+b;
return ob3;
}
void disp()
{
cout<<a<<b;
}
};
void main()
{
clrscr();
std p(2,3);
std q(3,2);
std r;
r=p+q;
r.disp();
getch();
}
```

Question 3:
(a) What is polymorphism? What are different forms of polymorphism? Explain implementation of polymorphism with the help of a C++ program. (8 Marks)
Ans:
Polymorphism is the ability **to** use**e an operato**r or function in different ways. Polymorphism gives different meanings to the operator or functions. Ploy refers to many, signifies th**e** many uses of these operators and functions. A single function usage in many forms or an operator functioning in many ways can be called polymorphism.

There are two types of polymorphism.

1. **Static polymorphism: -** It involves binding of functions based on the number, type, and sequence of arguments. There are various parameters are specified in the function declaration, and therefore the function can be bound to calls at compile time. This form of association is called early binding.
2. **Dynamic polymorphism:** - A function is said to dynamic polymorphism when it exists in more than one and calls to its various forms are resolved on run time when the program is executed.

Example

```
#include<iostream.h>
#include<string.h>
#include<stdio.h>
#include<conio.h>
class addstring
{
private:
char *p;
public:
addstring()
{
}
addstring(char *k)
{
strcpy(p,k);
}
addstring operator+(addstring obj1)
{
addstring temp;
strcpy(temp.p,p);
strcat(temp.p,obj1.p);
return temp;
}
void display()
{
cout<<"The string is = "<<p;
}
};

void main()
{
clrscr();
addstring ob1("www.pixeles");
addstring ob2("india.com");
addstring ob3("");
ob3=ob1+ob2;
ob3.display();
getch();
}
```

**(b) What is access control specifier ? Explain the need of different access control specifiers with
example.**

Private, Public, and Protected Base Classes

Private

The private data and function can be accessed from within the class. I.e. only from member of the same class. They are not accessible to the outside world. Hence, the primary mechanism for hiding data is to put data and function in a class and make them private.

Public

Public members of a class are accessible from within or outside the class i.e. they can be accessed by any unction inside or outside the class. These public members are accessible from derived class and also from object outside the class.

Protected

The protected data and functions can be accessed by the member functions of the same class. Also, these functions can be accessed by the member function of the derived class. But, it is not accessible to the outside world.

Base class access inheritance rules.

Base Class	Private Derived	Public Derived	Protected Derived
Private Member	*private*	*private*	*private*
Public Member	*private*	*public*	*protected*
Protected Member	*private*	*protected*	*protected*

```
class A
{
private:
int x;
void Function1(void);
public:
int y;
void Function2(void);
protected:
int z;
void Function3(void);
};
class B : A {}; // A is a private base class of B
class C : private A {}; // A is a private base class of C
class D : public A {}; // A is a public base class of D
class E : protected A {}; // A is a protected base class of E
```

The behavior of these is as follows:-

- All the members of a private base class become *private* members of the derived class. So x, Function1, y, Function2, z, and Function3 all become private members of B and C.
- The members of a public base class keep their access characteristics in the derived class. So, x and Function1 becomes private members of D, y and Function2 become public members of D, and z and Function3 become protected members of D.
- The private members of a protected base class become *private* members of the derived class. Whereas, the public and protected members of a protected base class become *protected* members of the derived class. So, x and Function1 become private members of E, and y, Function2, z, and Function3 become protected members of E. It is also possible to individually exempt a base class member from the access

Question 4 :

(a) Explain the concept of copy constructor with the help of an example program. (4 Marks)

Ans:

The **copy constructor** is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. The copy constructor is used to:

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

Example:-

```
#include <iostream.h>
class copy
{
int a,b;
public:
```

```
copy()
{
a=10;
}
copy( copy & t)
{
b=t.a;
}
void disp()
{
cout<<"value of a is ="<<b;
}
};
void main()
{
copy t;
copy x(t);
x.disp();
}
```

(b) What is an exception? How an exception is different from an error? Explain advantage of Exceptions handling in C++, with the help of an example program. (6 Marks)

Ans:

An exception is a situation that would be unusual for the program that is being processed. As a programmer, we should anticipate any abnormal behavior that could be caused by the user entering wrong information that could otherwise lead to unpredictable results. The ability to deal with a program's eventual abnormal behavior is called exception handling. C++ provides three keywords to handle an exception.

1. **try** {*Behavior*}

   The **try** keyword is required. It lets the compiler know that you are anticipating an abnormal behavior and will try to deal with it.
2. **catch(***Argument***)** {*WhatToDo*}
   This section always follows the **try** section and there must not be any code between the **try**'s closing bracket and the **catch** section. The **catch** keyword is required and follows the **try** section. The **catch** behaves a little like a function. It uses an argument that is passed by the previous try section. The argument can be a regular variable or a class. If there is no argument to pass, the **catch** must at least take a three-period argument as in **catch(…)**.

   ```
   try {
   // Try the program flow
   }
   catch(Argument)
   {
   // Catch the exception
   }
   ```

3. **Throwing an error:** There are two main ways an abnormal program behaviour is transferred from the **try** block to the **catch** clause. This transfer is actually carried by the **throw** keyword. Unlike the **try** and **catch** blocks, the **throw** keyword is independent of a formal syntax but still follows some rules.

Example:-

```
#include <iostream.h>
void main()
{
int a,b, result;
cout << "please provide two numbers\n";
try
{
cout << "first number: ";
cin >> a;
cout << "second number: ";
cin >> b;
if( b == 0 )
throw;

// perform a division and display the result
result = a / b;
cout << result << "\n\n";
}
catch(...)
{
cout<< “b cannot be zero”;
}
}
```

(b) What is data stream? Explain stream hierarchy in C++. (5 Marks)

ANs:

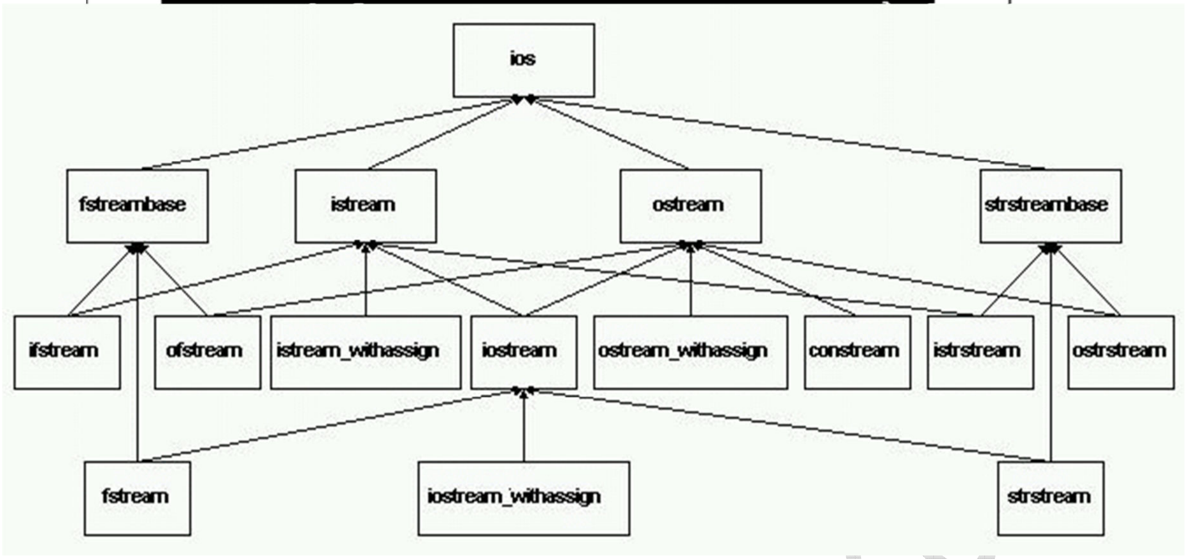
Stream: - it used to transfer of information in the form of a sequence of bytes. There are two types of stream operations:-

- Input: A stream that flows from an input device (i.e.: keyboard, disk drive, network connection) to main memory
- Output: A stream that flows from main memory to an output device (i.e.: screen, printer, disk drive, network connection)

There three pre-defined classes:-

- 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.

// writing on a text file



```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
#include<stdio.h>
void main()
{
char x;
clrscr();
ofstream ot;
ot.open("deepa.txt",ios::out);
cout<<"enter name";

do
{
x=getchar();
ot.put(x);
}while(x!='@');

getch();
}
```

Question 5:
(a) What is template? Explain advantage of using template in C++? Write C++ program to explain function template and class template. (7 Marks)
Ans:
Function templates are special functions that can operate with generic types. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type.
Advantages:

- Templates are easier to write. We can create generic version of class or function.
- It can provide a straightforward way of abstracting type information.
- Templates are type safe. it requires type-checking at compile time.

Example:- function Template

```
template <class T>
// function template
#include <iostream>
template <class T>
T GetMax (T a, T b) {
T result;
result = (a>b)? a : b;
return (result);
}

int main () {
int i=5, j=6, k;
long l=10, m=5, n;
k=GetMax<int>(i,j);
n=GetMax<long>(l,m);
cout << k << endl;
cout << n << endl;
return 0;
}
```

Class template

```
#include<iostream.h>
#include<conio.h>
template<class t>
class pix
{
public:
void disp(t a,t b)
{
cout<<"\nMy number is="<<a;
cout<<"\nYour Number Is="<<b;
}
};
void main()
{
clrscr();
pix<int> ob;
ob.disp(4,5);
pix<float>ot;
ot.disp(4.3,5.3);
getch();
}
```

(b) What is inheritance? Explain the different types of inheritance supported by C++? Explain whether constructors are inherited by derived class in C++ or not, write a program in support of your claim and show the output. (8 Marks)

Ans:

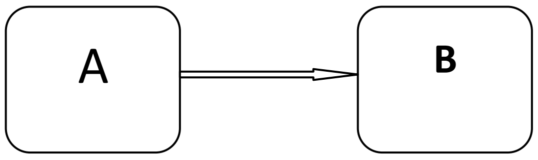
Inheritance is a process to create a new class from existing class or classes. Created new class is called sub class or child class or derived class and old class is known as parent class or super class or base class.

Advantage:

- Code reusability
- Function overriding
- Save compilation and programmer time
- Increase modularity

There are Five types of Inheritance:-

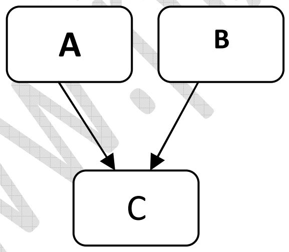
Single inheritance: - A class is derived from a Base class is called single inheritance class. Suppose Class 'A' is a base class and Class 'B' is derived from class 'A' then class 'A' is a super class and class 'B' is a sub class.



Syntax:-
Class A
{
….
…..
};
Class B: public A
{
….
}

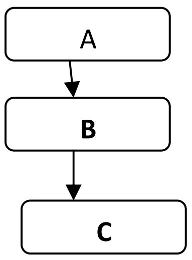
Multiple inheritances

A class is derived from more than one classes is called multiple inheritance. Here Class 'A' and Class 'B' are Base Classes and Class 'C' is derived from these two Base classes.

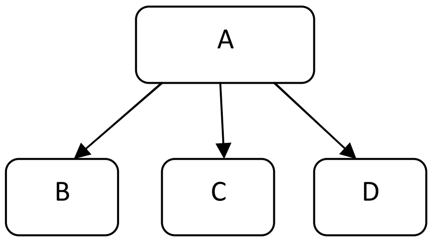


Multi-Level Inheritance

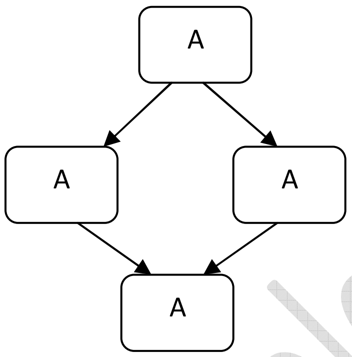
Class 'B' is derived from class 'A' and class 'C' is derived from Class 'B' and so on … is called Multi level inheritance.



Hierarchal Inheritance: - Super class and sub classes are arranged in tree structure and parent- child format.



Hybrid inheritance:-
It is combination of hierarchical and multiple inheritances.



Constructor can be inherited in derived class.

Example

```
#include<iostream.h>
#include<conio.h>
class pix
{
private :
int a,b,c;
public:
pix(int x)
{
a=x;
}
void disp()
{
cout<<"value of a="<<a;
}
};
```

```
class xyz:public pix
{
int b;
public:
xyz(int x, int y):pix(x)
{
b=y;
}
void show()
{
cout<<"values of b="<<b;
}
};
void main()
{
clrscr();
xyz p(4,5);
p.disp();
p.show();
getch();
}
```