BCS-031: Solved Assignment

1. **(a) What is Object Oriented Programming (OOP) paradigm? Explain basic concepts OOP.**

Ans:-

Object-oriented programming (oop) is a programming paradigm that represents the concept of "object" 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 opps.

Class and object:- A class is a blueprint that describes characteristics and function 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:-it is a machine of bundling the data , and the functions that use them and data abstraction is a mechanism of exposing only the interface and hiding the implementation details from the user. C++ supports the properties of encapsulation and the 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 pix
{
Private:
int a,c;b
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 object differently depending on their data types. It's the ability to redefine methods for derived classes.

Advantage:-

- Reusability of code
- Easy to maintenance.

(b) Explain different data types available in C++ programming.

Ans:-

C++ supports a large number of data types. Thus built in or basic data types supported by c++ are integer, floating point and character type. A brief discussion on these three data types.

1. **BUILET-IN DATA TYPES**

   A. **Integer data types (int):-** it accepts numeric data such as numbers.

- **Short**
- **Long**
- **Unsigned**

B. Floating point data type (float) :- A floating point number has a decimal point. Even if has an integral value, it must include a decimal point at the end. These numbers are used for measuring quantities.

Examples of valid floating point numbers are : 43.5, -54.9, and 45.087.

C. Void data type :-

It is used for following purposes:

- It specifics the return type of a function when the function is not returning any value.

D. **Char data type :-** It accepts one character or small integer. **1** byte Signed: -128 to 127 Unsigned: 0 to 255

2. **Derived data types:-** c++ also four types of derived data types. As the name suggests, derived data types are basically derived from the built-in data types. There are four derived data types. these are:-
   - ❖ **Array**
   - ❖ **Function**
   - ❖ **Pointer**
   - ❖ **Reference**
3. **User Defined data types:-**c++ also permits four types of user defined data types. As the name suggests, user defined data types are defined by the programmers during the coding of software development. There are four user defined data types. These are:-
   - ❖ **Structure**
   - ❖ **Union**
   - ❖ **Class**
   - ❖ **Enumerator**

(c) Explain the use of followings in C++ programming with an example program for each.

(i) While:- while () loop checks the condition on the top of loop first. If condition evaluates true then executes the statement. Otherwise, ignore it and terminate the loop.

For example:-

```
#include<conio.h>
#include<iostream.h>
int main()
{
int n=0;
while(n<10)
{
cout<<n;
n++;
}
cout<<"Pixeles classes";
return 0;
}
```

(ii) for:-it has three statements separated by semicolon (:)

For example:-

```
#includes<iostream.h>
#include<conio.h>
void main()
{
int i;
clrscr ();
for (i=0;i<=5;i++)
```

```
{
cout<<"PIXELES CLASES FOR BCA & MCA  \n RESULT= "<<i;
}
cout<<endl;
getch();
}
```

(d) Explain use of any two I/O formatting flags in C++ with example.
Ans:-

C++ defines some format flags for standard input and output, which can be manipulated with the flags, setf, and unsetf functions. For example,

- cout.setf(ios_base::left, ios_base::adjustfield);
- void stream::unsetf(fmtflags flags);

The function unsetf() uses flags to clear the io_stream_format_flags associated with the current stream.

```
#include <iostream.h>
main()
{
ios_base::fmtflags original_flags = cout.flags();
cout<< 812<<'|';
cout.setf(ios_base::left,ios_base::adjustfield);
cout.width(10);
cout<< 813 << 815 << '\n';
cout.unsetf(ios_base::adjustfield);
cout.precision(2);
cout.setf(ios_base::uppercase|ios_base::scientific);
cout << 831.0 << ' ' << 8e2;

cout.flags(original_flags);
}
```

2. (a) What is constructor? Define Book class with the basic features of a book. Define the default constructor, parameterised constructor, member functions display_book_details() for displaying the details of book. Use appropriate access control specifies in this program.
Ans:

Constructor

- Constructor is a special member function of class having same name of class
- It is called automatically when create an object.
- It doesn't have any return type even void.
- It can be parameterized.
- It can be overloaded.
- it can has default arguments
- It is declare as public always.

Default constructor: - constructors with no arguments or all are default arguments are called default constructor. If a constructor is not declare in class then compiler define it implicitly is also called default constructor.

```
class sum
{
sum() //default constructor
or
sum(int a=3, int b=4)
}
```

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

class book
{
int bid;
char bname[10],auth[10];
int price;
public:
book(int id, char *bn,char *au,int p)
{
bid=id;
strcpy(bname,bn);
strcpy(auth,au);
price=p;
}
void disp_book_details();
};
void book::disp_book_details()
{
cout<<"Book ID="<<bid<<endl;
cout<<"Book Name="<<bname<<endl;
cout<<"Author="<<auth<<endl;
cout<<"Price="<<price<<endl;
}

void main()
{
book ob(1,"Word","Deep",120);
ob.disp_book_details();
getch();
}
```

(b) Explain the following in detail with the help of examples, in context of C++ programming.

(i) Destructor :- it is also special member of class having same name but is defined with (tilde) ~ sign.

- It can be parameterised.
- It can't be overloaded.
- It does not return a value even void.
- It is called automatically when delete an object.

Example:-

```
#include<iostream.h>
#include<conio.h>
classpix
{
public:
pix()
{
cout<<"\n This is constructor";
}
```

```
~pix()
{
cout<<"\n This is destructor";
}
};
void main()
{
clrscr();
pix *op=new pix();
delete op;
getch();
}
```

(ii) Virtual Functions-

- Virtual function is member function of a class.
- Virtual function is declared with the keyword virtual.
- Virtual function takes a different functionality in the derived class.
- It is used to implement run time polymorphism.

For example:-

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

Class pixeles
{
public:
virtual void disp()
{
cout<<"PIXELES CLASSES FOR BCA & MCA" ;
}
};
Class india : public pixeles
{
public:
void disp()
{
cout<<"\n     A SUCCESS KEY";
}
};
class success : public pixeles
{
public:
void disp()
{
cout<<"\n www.pixelesindia.com";
}
};
void main()
 {
clrscr();
pixeles op;
```

```
india pp;
success sll;
op.disp();
pp.disp();
ll.disp();
pixels *pt;
pt= &pp;
pt->disp();
pt=&sll
pt->disp();

getch();
 }
```

(iii) Inline function:- C++ **inline** function is powerful concept that is commonly used with classes. If a function is inlined, the compiler places a copy of the code of that function at each point where the function is called at compile time.

To inline a function, place the keyword **inline** before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier in case defined function is more than a line.

```
inline int fun(int n)
{
return  (n > 0   ? 1   :0);
}
```

3.

(a) What is concept of reusability? How it is achieved in C++? Explain with the help of an example.

Ans:-software re-usability is primary attribute of software quality. C++ strongly supports the concept of reusability. C++ features such as classes, virtual function, and templates allow designs to be expressed so that re-use is made easier there are many advantage of reusability. They can be applied to reduce cost, effort and time of software development. It also increases the productivity, portability and reliability of the software product.

For example:-

```
#include<iostream.h>
#include<conio.h>
Class pixeles
{
public:
int c;
void sum(int x, int y)
{
c=x+y;
cout<<"PIXELES CLASSES FOR BCA & MCA \n RESULT="<<c;
}
};
Class pixelesindia: public pixeles
```

```
{
public:
int p;
void add(int x, int y)
{
p=x+y;
cout<<"\n www.pixelesindia.com  \n RESULT="<<p;
}
};
void main()
{
clrscr();
pixelesindia pc;
pc.sum(4,34);
pc.add(3,3);
getch();
}
```

(b) Write a C++ program to overload '+' operator to find the sum of two complex numbers

Ans :-

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

class add
{
public:
inta,b;
add(intr,int i)
{
a=r;
b=i;
}
add()
{
}
add operator +(add);
void disp();
};
add add::operator + (add t1)
{
add t2;
t2.a=a+t1.a;
t2.b=b+t2.b;
return t2;
}
void add :: disp()
{
cout<<" \n  RESULT=" <<a<<"and"<<b;
}
void main()
{
clrscr();
```

```
add pix1(12,4);
add pix2(82,4);
add pix3;
pix3=pix1+pix2;
pix3.disp();
getch();
}
```

(c) Explain different visibility modes in C++, in context of inheritance.

Ans:-There are three visibility modifier.
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 put data and function in a class and make them private.
Public :- public member 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 this class.
Protected :- the protected data and function 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:
Void Function2(void);
Int y;
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
```

4.
(a) What is function overloading? Explain how overloaded functions are called with the help of an example.

Function overloading:-more than one function is having same name but different argument signature.

Overloaded functions:-

- First c++ tries to find an exact match. This is the case where the actual argument exactly matches the parameter type of one of the overloaded functions. For example :

  Void print(char *szValue);
  Void print(intnValue);
  Print(10);//exact match with print(int)

  Although 10 could technically match print(char*), it exactly matches print(int). Thus print(int) is the best match available.

- If no exact match is found, C++ tries to find a match through promotion. In the lesson on type conversion and casting, we covered how certain types can be automatically promoted via internal type conversion to other types.
  For example:-

  Void print(char *szValue);
  Void print(intnValue);
  Print('a');//promoted to match print(int)

(b) What is dynamic binding? Explain with the help of an example.

Ans:- polymorphism is the ability to use an operator or function in different ways. Polymorphism gives different meanings to the operator or functions. Poly, referring to-many, signifies the many use of these operators and functions. A single function usage in many forms or an operator functioning in many ways can be called polymorphism.

For example:-

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

classpixeles
{
public:
virtual void disp()
{
cout<<"PIXELES CLASSES FOR BCA & MCA" ;
}
};
classindia : public pixeles
{
public:
voiddisp()
{
cout<<"\n     A SUCCESS KEY";
}
};
class success : public pixeles
{
public:
voiddisp()
```

```
{
cout<<"\n www.pixelesindia.com";
}
};
void main()
 {
clrscr();
pixeles op;
indiapp;
successll;
op.disp();
pp.disp();
ll.disp();

pixeles *h;
 h=&op;
h->disp();
 h=&pp;
h->disp();
 h=&ll;
h->disp();
getch();
 }
```

(c) What is an abstract class? Explain with an example. Also explain features of an abstract class.

Ans:- it is a base class that is create for the purpose derivation we need not create an object of his class.

For example:-

```
class student
{
private:
int roll;
char name[20];
public:
void accept()
{
cout<<"enter the number";
cin>>roll;
cout<<"enter the name";
}
void disp()
{
cout<<"roll="<<roll<<endl;
cout<<"name="<<name<<endl;
}
};
class mark:public student
{
private:
int m;
```

```
public:
void accept()
{
student::accept();
cout<<"enter the marks";
cin>>m;
}
void disp()
{
student::disp();
cout<<"marks="<<m;
}
};
void main()
{
mark t;
t.accept();
t.disp();
getch();
}
```

4.

(a) What is template? Write appropriate statements to create a template class for Stack data structure in C++

Ans :-function template 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.

For example:-

```
#define max 10
#include<iostream.h>
template <class t>
class stack
{
private:
t  arr[max]
t  top;
public:
void push(t n)
{
if(top==max-1) // check max position is full.
{
cout<<"overflow";
return;
}
else
{
top++; // increase top position
stack[top]=n;     // store new value at top of stack
}}
void  pop()
{
```

```
t  x;
if(top == -1)  // check if element is not avail in stack
{
cout<<"underflow!! element is not avail to delete";
}
else
{
x=stack[top];
top--;
cout<<"\deleted element is %d\n"<<x;
}
}
void main()
{
stack <int> obj
obj.push(10);
obj.pop();
getch();
}
```

(b) What are containers? Explain use of List container class with the help of an example

Ans :- Containers are objects that hold data. These container classes are defined as class templates that can be customized to hold different kinds of data. The container classes contain definitions for commonly used data structures such as vector, list, queue, stack, set, map etc. Each container class also defines a set of functions that can be used to manipulate its contents. The STL defines a number of containers which can be grouped into mainly three types: sequence containers, associative containers and derived containers. The derived containers are sometime also referred to as container adapters.

Container	Description	Header File	Iterator
vector	A dynamic array. Allows insertion and deletions at rear. Permits direct access.	<vector>	Random access
list	A bidirectional linear list. Allows insertion and deletions anywhere.	<list>	Bidirectional
stack	A standard stack, Last in First out operation.	<stack>	No iterator
queue	A standard Queue, First in First out operation.	<queue>	No iterator
priority queue	A priority queue, highest priority element as first out.	<queue>	No iterator
deque	A double ended queue, allows	<deque>	Random

Example:

```
int main()
 {
string name; int number; phonemap phone;
cout << "Enter three sets of name and numbers \n";
for (int i=0, i<3; i++)
{
cin >> name;
cin >> number;
phone[name] = number;
}
phone["Ramesh"]= 621345;
phone.insert(pair<string, int> ("ajay", 234432));
int n = phone.size();
cout << "\n size of map:" << n;
cout << "\n List of telephone numbers \n";
phonemap::iterator p;
for (p=phone.begin(); p!=phone.end(); p++)
cout << (*p).first << " " << (*p).second << "\n"; cout << "\n";
cout << "Enter name:";
cin >> name;
number = phone[name];
cout << "Number:" << number << "\n";
 return(0);
};
```

(c) What is exception handling? Write C++ program to demonstrate exception handling with multiple catch statements.

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 behaviour 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 behaviour is called exception handling. C++ provides three keywords to handle an exception.

- **Try** :- It encloses the programming statements that may have an abnormal condition.
- **Catch:-** it is used to recognize the exception that is identified in try block. It has an argument the specify the types of exception. If there is no argument to pass, the **catch** must at least take a three-period argument as in **catch(…)**.

- **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”;
}
}
```