Q.1 Write a C++ Program to implement:

(1) Multiple Inheritance (2) Constructor and destructor 4

Ans: **Multiple Inheritance**

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

class student {
protected:
    int rno, m1, m2;
public:

    void get() {
        cout << "Enter the Roll no :";
        cin>>rno;
        cout << "Enter the two marks   :";
        cin >> m1>>m2;
    }
};

class sports {
protected:
    int sm; // sm = Sports mark
public:

    void getsm() {
        cout << "\nEnter the sports mark :";
        cin>>sm;

    }
};

class statement : public student, public sports {
    int tot, avg;
public:

    void display() {
        tot = (m1 + m2 + sm);
        avg = tot / 3;
        cout << "\n\n\tRoll No    : " << rno << "\n\tTotal      : " << tot;
        cout << "\n\tAverage    : " << avg;
    }
};

void main() {
    clrscr();
    statement obj;
    obj.get();
    obj.getsm();
    obj.display();
    getch();
}
```

Constructor and destructor

Ans: Constructor has the same name as the name of the class. Constructors are automatically called as soon as the object of the class is created. Constructor is also used to initialize the object of that class.

Destructor is the opposite of constructor. Destructor is used to destroy the objects created by constructor. Destructor also has the same name as of the class. There will be only one destructor in a class.

```
#include<iostream.h>

#include<conio.h>

Class student

{

Private:

Char name [25];

Int roll;

Float height, weight;

Public:

Student ()

{

Strcpy (name, “ram”);

roll=0;

height=0;

weight=0;

}

Void display ()

{

Cout<<”\n name :”<< name;

Cout<<”\n roll no”<<roll;
```

```
Cout<<"\n Height"<<height;

Cout<<"\n weight"<<weight;

}

~ Student ()

{

Cout<<"\n destroying object";

}

};

void main ()

{

Student obj;

Obj. Student ();

getch ();

}
```

The above program is designed in-order to store the data's of student. The class named 'student' has been declared in the program. This program uses constructor and destructor.

Ans: Object-oriented programming (OOP) languages are designed to overcome these problems.

1. The basic unit of OOP is a *class*, which encapsulates both the *static attributes* and *dynamic behaviors* within a "box", and specifies the public interface for using these boxes. Since the class is well-encapsulated (compared with the function), it is easier to reuse these classes. In other words, OOP combines the data structures and algorithms of a software entity inside the same box.
2. OOP languages permit *higher level of abstraction* for solving real-life problems. The traditional procedural language (such as C and Pascal) forces you to think in terms of the structure of the computer (e.g. memory bits and bytes, array, decision, loop) rather than thinking in terms of the problem you are trying to solve. The OOP languages (such as Java, C++, C#) let you think in the problem space, and use software objects to represent and abstract entities of the problem space to solve the problem.

Benefits of OOP

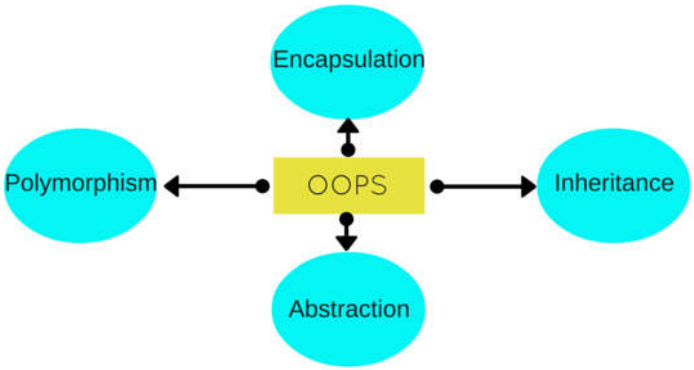
The procedural-oriented languages focus on procedures, with function as the basic unit. You need to first figure out all the functions and then think about how to represent data.

The object-oriented languages focus on components that the user perceives, with objects as the basic unit. You figure out all the objects by putting all the data and operations that describe the user's interaction with the data.

Object-Oriented technology has many benefits:

- *Ease in software design* as you could think in the problem space rather than the machine's bits and bytes. You are dealing with high-level concepts and abstractions. Ease in design leads to more productive software development.
- *Ease in software maintenance*: object-oriented software are easier to understand, therefore easier to test, debug, and maintain.
- *Reusable software*: you don't need to keep re-inventing the wheels and re-write the same functions for different situations. The fastest and safest way of developing a new application is to reuse existing codes - fully tested and proven codes.

Object Oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc.



Ans:

(1)

C Structure :-
1. Only variables of different data types can be declared, functions are not allowed

2. Direct access to data members is possible

3. 'struct' data type is not treated as built in type – use of 'struct' necessary to declare objects

4. Member variables cannot be initialized inside a structure

C++ Structure :-
1. In C++ structure declaration functions can also be declared

2. The members declared in a C++ structure is public by default

3. While declaring an object the keyword 'struct' is omitted in C++

(2) Pointer

A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is

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

Here, **type** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer variable. The asterisk ***** used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer declarations

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

There are a few important operations, which we will do with the help of pointers very frequently. **(a)** We define a pointer variable, **(b)** assign the address of a variable to a pointer and **(c)** finally access the value at the address available in the pointer variable. This is done by using unary operator ***** that returns the value of the variable located at the address specified by its operand.

(3) Access Specifier

Access specifiers in C++ class defines the access control rules. C++ has 3 new keywords introduced, namely,

1. public
2. private
3. protected

These access specifiers are used to set boundaries for availability of members of class be it data members or member functions

Access specifiers in the program, are followed by a colon. You can use either one, two or all 3 specifiers in the same class to set different boundaries for different class members. They change the boundary for all the declarations that follow them.

Public

Public, means all the class members declared under public will be available to everyone. The data members and member functions declared public can be accessed by other classes too. Hence there are chances that they might change them. So the key members must not be declared public.

Private

Private keyword, means that no one can access the class members declared private outside that class. If someone tries to access the private member, they will get a compile time error. By default class variables and member functions are private.

Protected

Protected, is the last access specifier, and it is similar to private, it makes class member inaccessible outside the class. But they can be accessed by any subclass of that class. (If class A is inherited by class B, then class B is subclass of class A.

II Mid Term Exam 2017-18 III Sem IT Object Oriented Programming Marks :10

Q.1 Explain Template Class and Function with a C++ Program. 5

Ans: Templates allow programmer to create a common class or function that can be used for a variety of data types. The parameters used during its definition are of generic type and can be replaced later by actual parameters.

Types of templates

Templates in C++ can be divided into major two types, they are

- Function Template
- Class Template

As of C++ 11, Variable Template has also been added.

Function Template

A generic function that represents several functions performing same task but on different data types is called function template. **For example**, a function to add two integer and float numbers requires two functions. One function accept integer types and the other accept float types as parameters even though the functionality is the same. Using a function template, a single function can be used to perform both additions. It avoids unnecessary repetition of code for doing same task on various data types.

Example of Function Template

1. C++ program to add two numbers using function template.

```
#include <iostream>
#include <conio.h>
using namespace std;

template<class t1,class t2>
void sum(t1 a,t2 b) // defining template function
{
    cout<<"Sum="<<a+b<<endl;
}

int main()
{
    int a,b;
    float x,y;
    cout<<"Enter two integer data: ";
    cin>>a>>b;
    cout<<"Enter two float data: ";
    cin>>x>>y;
    sum(a,b); // adding two integer type data
    sum(x,y); // adding two float type data
    sum(a,x); // adding a float and integer type data
    getch();
    return 0;
}
```

This program illustrates the use of template function in C++. A template function *sum()* is created which accepts two arguments and add them. The type of argument is not defined until the function is called. This single function is used to add two data of integer type, float type and, integer and float type. We don't need to write separate functions for different data types. In this way, a single function can be used to process data of various type using function template.

Class Template

Like function template, a class template is a common class that can represent various similar classes operating on data of different types. Once a class template is defined, we can create an object of that class using a specific basic or user-defined data types to replace the generic data types used during class definition.

2. C++ program to use class template

```
#include <iostream>
#include <conio.h>
using namespace std;

template<class t1,class t2>
class sample
{
    t1 a;
    t2 b;
    public:
        void getdata()
        {
            cout<<"Enter a and b: ";
            cin>>a>>b;
        }
        void display()
        {
            cout<<"Displaying values"<<endl;
            cout<<"a="<<a<<endl;
            cout<<"b="<<b<<endl;
        }
};

int main()
{
    sample<int,int> s1;
    sample<int,char> s2;
    sample<int,float> s3;
    cout <<"Two Integer data"<<endl;
    s1.getdata();
    s1.display();
    cout <<"Integer and Character data"<<endl;
    s2.getdata();
    s2.display();
    cout <<"Integer and Float data"<<endl;
    s3.getdata();
    s3.display();
    getch();
    return 0;
}
```

Q.2 What is virtualization? How virtual function and Dynamic binding are important explain. 3

Ans:

A virtual function is a member function that is declared within a base class and redefined by a derived class. To create virtual function, precede the function's declaration in the base class with the keyword virtual. When a class containing virtual function is inherited, the derived class redefines the virtual function to suit its own needs.

- Base class pointer can point to derived class object. In this case, using base class pointer if we call some function which is in both classes, then base class function is invoked. But if we want to invoke derived class function using base class pointer, it can be achieved by defining the function as virtual in base class, this is how virtual functions support runtime polymorphism.
- Consider the following program code :

```
Class A
{
  int a;
  public:
     A()
     {
       a = 1;
     }
     virtual void show()
     {
       cout <<a;
     }
};

Class B: public A
{
  int b;
  public:
     B()
     {
       b = 2;
     }
     virtual void show()
     {
       cout <<b;
     }
};

int main()
{
  A *pA;
  B oB;
  pA = &oB;
  pA→show();
  return 0;
}
```

Polymorphism is also achieved in C++ using virtual functions. If a function with same name exists in base as well as parent class, then the pointer to the base class would call the functions associated only with the base class. However, if the function is made virtual and the base pointer

is initialized with the address of the derived class, then the function in the child class would be called.

Virtual function is the member function of a class that can be overriden in its derived class. It is declared with virtual keyword. Virtual function call is resolved at run-time (dynamic binding) whereas the non-virtual member functions are resolved at compile time (static binding).

A virtual function in C++ is :
- A simple member function of a class which is declared with “virtual” keyword
- It usually performs different functionality in its derived classes.
- The resolving of the function call is done at run-time.

- A virtual table is a mechanism to perform dynamic polymorphism i.e., run time binging. Virtual table is used to resolve the function calls at runtime. Every class that uses virtual functions is provided with its own virtual functions.
- Every entry in the virtual table is a pointer that points to the derived function that is accessible by that class. A hidden pointer is added by a compiler to the base class which in turn calls *_vptr which is automatically set when an instance of the class is created and it points to the virtual table for that class.

Q.3 Write a C++ program to explain friend function. 2
Ans:

Friend Class A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class. For example a LinkedList class may be allowed to access private members of Node.

```
class Node
{
private:
  int key;
  Node *next;
  /* Other members of Node Class */

  friend class LinkedList; // Now class  LinkedList can
                           // access private members of Node
};
```

Friend Function Like friend class, a friend function can be given special grant to access private and protected members. A friend function can be:
a) A method of another class
b) A global function

```
class Node
{
private:
  int key;
  Node *next;

    friend int LinkedList::search(); // Only search() of linkedList
                                     // can access internal members
};
```

Following are some important points about friend functions and classes:

1) Friends should be used only for limited purpose. too many functions or external classes are declared as friends of a class with protected or private data, it lessens the value of encapsulation of separate classes in object-oriented programming.

2) Friendship is not mutual. If a class A is friend of B, then B doesn't become friend of A automatically.

3) Friendship is not inherited

4) The concept of friends is not there in Java.

```
#include <iostream>
class A {
private:
    int a;
public:
    A() { a=0; }
    friend class B;     // Friend Class
};

class B {
private:
    int b;
public:
    void showA(A& x) {
        // Since B is friend of A, it can access
        // private members of A
        std::cout << "A::a=" << x.a;
    }
};

int main() {
   A a;
   B b;
   b.showA(a);
   return 0;
}
```