Q1- **On Friend Function -**

Create these three classes - student, lib, degree.

variables:

student will have these variables : rn, m1,m2

lib will have these variables: numofbooks

degree will have these variables: char award_degree;

Functions:

Create set fun in all these classes.

Create a fun get in all these classes.

objects:

Create student_obj and assign value 1,45,60

Create lib_obj and assign value 2

Create degree_obj and do not assign any value.

Friend function:

create a friend fun "check". This friend fun will be called like this

degree_obj = check(student_obj, lib_obj);

degree_obj.get(); // this will print ‘y’ or ‘n’

It will check if total marks >50 in student_obj and numofbooks == 0 in lib_obj. If yes, then it will set award_degree =’y’ in degree_obj.

Answer-1

```
#include<iostream>
using namespace std;
//write forward declaration of all classes except the first class
class lib;
class degree;

//define class-1
class student
{
   int rn,m1,m2;
   public:
   void set(int x, int y, int z);
   void get();
   //declare friend fun in each class
   friend degree check(student po1, lib po2);


};
//define fun of class-1
void student::set(int x, int y, int z)
{
   rn=x;
   m1=y;
```

```
   m2=z;
}
void student::get()
{
   cout<<rn<<endl<<m1<<endl<<m2<<endl;
}
//Define class-2
class lib
{
   int nob;
   public:
      void set(int x);
      //write FF in class-2
      friend degree check(student po1, lib po2);


};
//Define fun of class-2
void lib::set(int x)
{

   nob=x;
}

//Define class-3
class degree
{
   private:
      char award_degree;
   public:
      void get();
      //Write FF in class-3
      friend degree check(student po1, lib po2);

};
//Write fun of class-3
void degree::get()
{
   cout<<"Check if degree awarded: "<<award_degree<<endl;
}
//Write the FF after all classes
//Write FF only once.

//Return type of FF should be the object returned by fun.
degree check(student po1, lib po2)
{
   //create a temp result object of class-3
   degree result;
   if(po1.m1+po1.m2 > 50 &&  po2.nob==0)
```

```
    {
        //store result in temp result object
        result.award_degree='y';
    }
    else result.award_degree='n';
    //This temp result object will be the returning object
    return result;
}

int main()
{
    //create objects of all three classes
    student obj1;
    lib obj2;
    degree obj3;
    //assign values
    obj1.set(1,45,60);
    obj2.set(2);
    //call FF
    obj3=check(obj1,obj2);
    //Check the value of obj3
    obj3.get();

}
```

Q2- What is operator overloading? Which function is used for operator overloading? Why is operator overloading done ?

Answer-2

Overloading means to create new uses for something. Or to write user definitions for something. OO means to write user defined fun for these operators so that they can be used to perform operations on objects.

For eg. Normally, we can't add two objects using + , (obj1+obj2 is invalid)

But, with OO, we can add two objects using + operator.

Operator fun is used for overloading operators. This operator fun will have to be defined by the user.

OO is done so that we can perform operations on objects using operators.

For eg. we can do these operations easily

obj3=obj1+obj2; // By overloading + operator

obj3=obj1=obj2; // By overloading = operators

obj3=++obj1; // By overloading pre-inc operators

Q3-**On operator overloading** using member operator function
Create a class student. Write a program to overload + , = , pre-increment, postincrement operators for this class using member operator function.
Inside main create three objects and do these operations.
Obj1.set(1,20,30);
obj2.set(2, 5, 6);
Create obj3;
obj3= obj1+obj2;
obj3.get();
obj3=obj2=obj1;
obj3.get();
obj3 = ++obj2;
obj3.get()
obj2.get()
obj3 = obj2++;
obj3.get();
obj2.get();

Answer-3

```
#include<iostream>
using namespace std;

class student
{
        private:
        int rn, m1,m2;
        public:
        void set(int , int , int);
        void get();
        void add();
        //Declaring all the operator overloading functions.
        //**THESE FUNCTIONS ARE MEMBER FUNCTIONS *******
        student operator+(student po1);//overloading + operator
        student operator=(student po1);//overloading assignment operator =
```

```
        student operator++();//overloading pre-increment operator
        student operator++(int x);//overloading post-inc operator


};
void student::set(int x, int y, int z)
{
        rn=x;
        m1=y;
        m2=z;
}
void student::get()
{
        cout<<rn<<endl<<m1<<endl<<m2<<endl;
}
void student::add()
{
        cout<<m1+m2<<endl;
}
//Overloading + operator: add calling obj and po to a temp obj
//return this temp obj
//rn,m1,m2 -> calling obj
//po1.m1, po2.m2 -> passing obj
student student::operator+(student po1)
{
   student temp;
   temp.rn=3;
   temp.m1=m1+po1.m1;
   temp.m2=m2+po1.m2;
   return temp;
}
//overloading the assignment operator.
//rn,m1,m2 -> calling obj
//po1.m1, po2.m2 -> passing obj
student student::operator=(student po1)
{
   //rn is not copied, only marks are copied.
   m1=po1.m1;
   m2=po1.m2;
```

```
   //*this will return the calling object.
   return *this;
}
//overloading pre-inc operator
//m1,m2 -> for calling object
student student::operator++()
{

   ++m1;
   ++m2;
   //*this will return the calling obj
   return *this;
}
//overloading the post-inc operator
//To call post inc operator int x must be specified as parameter
//This int x parameter will not be used in the definition.
student student::operator++(int x)
{
   //First create a temp obj
   student temp;
   //copy the calling object to this temp obj
   temp.m1=m1;
   temp.m2=m2;
   //Then inc the calling object
   m1++;
   m2++;
   //Return the temp object;
   return temp;
}

int main()
{
        student obj1, obj2, obj3;
        obj1.set(1,3,4);
        obj2.set(2,97,96);
        obj3.set(3,-1,-1);
        cout<<"Add obj1 and obj2 and copy to obj3"<<endl;
        //obj1 is calling obj
        //obj2 is passing obj
```

```
        //obj3 is assigning obj
        obj3=obj1+obj2;//overloaded + operator fun is called
        obj3.get();

        cout<<"Assign obj1 to obj2 and obj3"<<endl;
        obj1.set(1,30,40);
        //obj2 is calling obj
        //obj1 is passing obj
        //obj3 is assigning obj
        obj3=obj2=obj1;//overloaded (=) assignment operator fun is called
        obj3.get();

    cout<<"Pre increment obj2 and assign to obj3"<<endl;
    obj2.set(2,9,19);
    //obj2 is calling obj
    //No passing obj
    //obj3 is assigning obj
    obj3=++obj2; //overloaded pre-inc operator fun is called
    obj3.get();


    cout<<"Post increment obj2 and assign to obj3"<<endl;
    obj2.set(2,500,600);
    //obj2 is calling obj
    //No passing obj
    //obj3 is assigning obj
    obj3=obj2++;//overloaded post-inc operator fun is called.
    obj3.get();
    obj2.get();


}
```

Q4- **On operator overloading** using friend operator function

Create a class student. Write a program to overload +, pre-increment, post-increment operators for this class using a **friend operator function.**

Note: In this question a friend operator function is used for a single class. But it can

be used for more than one class.

Inside main create three objects and do these operations.

Obj1.set(1,20,30);
obj2.set(2, 5, 6);
Create obj3;
obj3= obj1+obj2;
obj3.get();
obj3=obj2=obj1;
obj3.get();
obj3 = ++obj2;
obj3.get()
obj2.get()
obj3 = obj2++;
obj3.get();
obj2.get();

Answer-4:

```
#include<iostream>
using namespace std;

class student
{
        private:
        int rn, m1,m2;
        public:
        void set(int , int , int);
        void get();
        void add();
        /*Overloaded operator functions using Member Functions are declared like this:
        student operator+(student po1);//overloading + operator
        student operator=(student po1);//overloading assignment operator =
        student operator++();//overloading pre-increment operator
        student operator++(int x);//overloading post-inc operator
        */

        //overloaded operator functions using Friend Function are declared like this:
        //friend keyword is used and there is an extra passing object
```

```
        //There is no calling obj, so there will be an extra passing object.
        friend student operator+(student po1,student po2);//overloading + operator
using FF
        //assignment operator can't be overloaded using friend fun.
        //friend student operator=(student po1, student po2);
        friend student operator++(student &po1);//overloading pre-increment operator
using FF
        friend student operator++(student &po1,int x);//overloading post-inc operator
using FF

};
void student::set(int x, int y, int z)
{
        rn=x;
        m1=y;
        m2=z;
}
void student::get()
{
        cout<<rn<<endl<<m1<<endl<<m2<<endl;
}
void student::add()
{
        cout<<m1+m2<<endl;
}
//Overloading + operator:
//There is no calling obj.
//There are two po
//add both po to a temp obj and return this temp obj
//po1.rn,po1.m1,po1.m2 -> passing obj1
//po2.rn, po2.m1, po2.m2 -> passing obj2
student operator+(student po1,student po2)
{
   //create a temp obj
   student temp;
   temp.rn=3;
   //add po1 and po2 to this temp obj
   temp.m1=po1.m1+po2.m1;
   temp.m2=po1.m2+po2.m2;
```

```
   //return this temp obj
   return temp;
}

//overloading pre-inc operator
//There is no calling obj
//There is one po and ***it is passed by reference***
//This is done to make permanent changes to this po
student operator++(student &po1)
{
   //create a temp obj
   student temp;
   //assign rn to this temp obj.
   temp.rn=3;
   //inc the po
   ++po1.m1;
   ++po1.m2;
   //copy the po to temp
   temp.m1=po1.m1;
   temp.m2=po1.m2;
   //return the temp obj
   return temp;
}

//overloading the post-inc operator
//There is no calling obj
//There are two parameters: po and int x
//To call post inc operator int x must be specified as parameter
//This int x parameter will not be used in the definition.
student operator++(student &po1,int x)
{
   //First create a temp obj
   student temp;
   //copy the passing object to this temp obj
   temp.rn=3;
   temp.m1=po1.m1;
   temp.m2=po1.m2;
   //Then inc the passing object
   po1.m1++;
```

```
   po1.m2++;
   //Return the temp object;
   return temp;
}

int main()
{
        student obj1, obj2, obj3;
        obj1.set(1,3,4);
        obj2.set(2,97,96);
        obj3.set(3,-1,-1);
        cout<<"Add obj1 and obj2 and copy to obj3"<<endl;
        //There is no calling obj because it is a FF
        //obj1 is passing obj or po1
        //obj2 is passing obj or po2
        //obj3 is assigning obj
        obj3=obj1+obj2;//overloaded + operator fun is called
        obj3.get();


   cout<<"Pre increment obj2 and assign to obj3"<<endl;
   obj2.set(2,9,19);
   //There is no calling obj
   //obj2 is passing obj or po1
   //obj3 is assigning obj
   obj3=++obj2; //overloaded pre-inc operator fun is called
   obj3.get();
   obj2.get();

   cout<<"Post increment obj2 and assign to obj3"<<endl;
   obj2.set(2,500,600);
   //There is no calling obj because it is a FF
   //obj2 is passing obj or po1
   //obj3 is assigning obj
   obj3=obj2++;//overloaded post-inc operator fun is called.
   obj3.get();
   obj2.get();

}
```

Q5- What are **references** in C++ ? What is the difference between a reference and a pointer ? Give an example of how you will create a reference for a variable in c++ ?

Answer-5
References are a feature of c++. They can be an alternative to pointers.
References are implicit pointers. When you create a reference for a variable , you are using that as another name for that variable. Now whatever, changes you make to a reference, it will be reflected in the variable.

Creating a reference:

```
int a=10;
int &b=a; //b is now a reference to a
//now b is another name for a. Whatever change you make in b will be reflected in a

b=20;
cout<<a<<b;

//output will be 20, 20
//Remember, whatever change you make to b will be reflected in a and vice versa.
```

Q6- WAP to swap two variables using **Call by value, Call by address, and Call by Reference ?**

Answer-6

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

```
    *x=*y;
    *y=temp;
}
void cbr(int &x, int &y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
int main()
{
    int a=10, b=20;
    cout<<"a and b before CallByValue "<<a<<" "<<b<<endl;
    cbv(a,b);
    cout<<"a and b after CallByValue "<<a<<" "<<b<<endl;

    a=50;
    b=60;
    cout<<"a and b before CallByAddress "<<a<<" "<<b<<endl;
    cba(&a,&b);
    cout<<"a and b after CallByAddress "<<a<<" "<<b<<endl;

    a=45;
    b=54;
    cout<<"a and b before CallByReference "<<a<<" "<<b<<endl;
    cbr(a,b);
    cout<<"a and b after CallByReference "<<a<<" "<<b<<endl;

}
```

Q7- What are dynamic objects? In how many ways can you create Dynamic objects in C++ ?

Answer-7

Those obj for which you allocate memory during run time are called Dynamic objects. There are three ways to create a dynamic obj – using malloc fun, calloc fun or new operator.

You can create dynamically allocated arrays of object using these operators

Q8- On **Dynamic Objects using new operator.**

WAP to create a class student. Create an array of objects dynamically using new operator. The number of objects in array should be input by user during runtime. During runtime , input the value 3 and call set , get, add fun for these 3 dynamic objects.

Answer-8

```
#include<iostream>
using namespace std;
 class student
 {
   private:
   int rn,m1,m2;
   public:
   void set(int x , int y,int z);
   void get();
   void add();


 };

 void student::set(int x , int y,int z)
 {
   rn=x;
   m1=y;
   m2=z;
 }

 void student::get()
 {
   cout<<rn<<endl<<m1<<endl<<m2<<endl;
 }
 void student::add()
 {
   cout<<m1+m2<<endl;
 }
```

```
int main()
{
  int n;
  cout<<"How many elements do you want in array of objects "<<endl;
  cin>>n;
  //create dynamic objects of n elements using new operator:
  //First create a pointer
  student *arr_obj;
  //Use new operator to allocate memory for three objects
  arr_obj=new student[n];

  //Calling set function for these objects using loop
  for(int i=0; i<n;i++)
  {
     int a,b,c;
     cout<<"input values for object: "<<i<<endl;
     cin>>a>>b>>c;
     arr_obj[i].set(a,b,c);
  }
  //calling get function for these objects using loop
  for(int i=0;i<n;i++)
  {
     cout<<"print values for object: "<<i<<endl;
     arr_obj[i].get();
  }
  //calling add fun for these obj using loops
  for(int i=0;i<n;i++)
  {
     cout<<"print sum for object: "<<i<<endl;
     arr_obj[i].add();
  }

}
```

Q9- What is a **Scope resolution operator**? List three uses of scope resolution operator.

Answer-9
Scope resolution operator is ::
1-It can be used while defining member fun of a class
void student::set(int x, int y, int z);

2-It can be used while accessing a namespace
std::cout<<"hello world"<<endl;

3-It can be used to access a global variable
eg. if there are local and global var of same name then , global var becomes hidden. In that case SRO is used to access the global variable

```
int a;
int main()
{
        int a;
        cout<<a; // local a is printed;
        cout<<::a //global a is printed;
}
```

Q10- What is function overloading? Why is it done? What rule must be followed for function overloading?

Answer-10
Fun overloading means to write several fun with same names. It is a feature of oops and only allowed in c but not in c++.

FO allows us to use same name for different fun. It makes it easier as we do not have to use different names for fun.

Rule:
All the fun shud have different parameters.

Q11- On **Function Overloading**
Do not use any class. Overload a simple sum function three times without any class.

Answer-11

```
#include<iostream>
using namespace std;
//Three fun are created.
//All have same name i.e add. So this is called fun overloading
//But the parameters in all three fun are different.

void add(int x, int y)
{
   cout<<"fun with two parameters"<<endl;
   cout<<x+y<<endl;

}
void add(int x, int y, int z)
{
   cout<<"fun with three parameters"<<endl;
   cout<<x+y+z<<endl;
}
void add(char x, char y)
{
   cout<<"fun with character parameters"<<endl;
   cout<<x+y<<endl;
}
int main()
{
   //call the add fun with different parameters.
   add(1,19);//output will be 20
   add(10,10,10);//output will be 30
   add('a', 'b');//output will be ascii value of a and b = (97+98=195)
}
```

Q12- On **Function overloading with class**

Overload the set function two time in student class

Answer-12

```
#include<iostream>
using namespace std;
```

```
class student
{
        private:
        int rn, m1,m2;
        public:
           //Declare two set functions here, with different parameters
        void set(int , int , int);
        void set(int);
        void get();
        void add();


};
void student::set(int x, int y, int z)
{
   //This set will assign values to all variables
        rn=x;
        m1=y;
        m2=z;
}
void student::set(int x)
{
   //This set will assign values to only rn.
   rn=x;
}
void student::get()
{
        cout<<rn<<endl<<m1<<endl<<m2<<endl;
}
void student::add()
{
        cout<<m1+m2<<endl;
}

int main()
{
        student obj1, obj2;
        //obj1 is assigned rn=5;
```

```
        obj1.set(5,10,20);
        obj1.get();
        //Now change rn of obj1 to 1;
        obj1.set(1);
        obj1.get();

        obj2.set(2,15,15);
        obj2.get();
        obj2.add();



}
```

Q13- What is constructor overloading ? Why is it done ?

Answer-13
CO means to have more than one constructor. But there is one condition, these const must have different parameters.

It is done to have flexibility in creating objects.
We can create obj two ways

student obj1(1,11,11); // by passing values. Parameterized const will be called
student obj2; // without passing values. Blank const willl be called

Q14- on **Constructor overloading.**
Create the student class. Create two constructors for this class. One with parameters and other without parameters. Create two objects using different constructors.

Answer-14

```
#include<iostream>
using namespace std;

class student
{
        private:
```

```
        int rn, m1,m2;
        public:
        void set(int , int , int);
        void get();
        void add();
        //There will be two constructors

        //This constructor is with parameters
        student(int , int , int);

        //This constructor is without parameters
        //It can be used to create obj without passing any values
        student();


};
void student::set(int x, int y, int z)
{
        rn=x;
        m1=y;
        m2=z;
}

void student::get()
{
        cout<<rn<<endl<<m1<<endl<<m2<<endl;
}
void student::add()
{
        cout<<m1+m2<<endl;
}
student::student(int x , int y, int z)
{
   //This is parameterized constructor
        rn=x;
        m1=y;
        m2=z;
        cout<<"constructor called"<<endl;
}
```

```
student::student()
{
   //This is non parameterized constructor
   cout<<"constructor called"<<endl;

}

int main()
{
   //You can create obj in two ways
   //obj1 is created by passing values, obj2 is created without passing values
         student obj1(1,11,11), obj2;
         obj1.get();
         obj1.add();

   //Call set fun for obj2 to assign values.
         obj2.set(2,22,22);
         obj2.get();
         obj2.add();


}
```

Q15- What is a copy constructor? How many copy constructors are there ? What are the situations in which a copy constructor is called?

Answer-15
The CC is a const which is called when one obj is being copied into another.
There are two CC
1- Default cc – It is provided by the compiler. It will be called automatically when one obj is copied into other
2- User defined cc / overloaded cc - It is will be defined/written by the user.

When one obj is copied to another then,
-def cc is called if there is no user defined cc
-def cc is not called if there is a user defined cc

cc will be called in three situations
1-when one obj is copied to another using = operator
eg. obj3=obj1; // here def cc or user defined will copy obj1 to obj3.
//note that = operator doesn't copy obj1 to obj2. Rather cc will be called and it will copy obj1 to obj3

2- When you call the cc explicitly as below
eg. obj3(obj1); // here obj1 is copied to obj3.
//note that , cc is called explicitly here.

3- when you pass obj as parameters to a fun.
In this case, cc will be called for the obj which are in the fun parameters.
eg. obj3.add_obj(obj1, obj2); // obj1, obj2 are passed to this fun.

Void student::add_obj(student po1, student po2); // this is the fun declaration.

When the fun call is made obj1 is copied to po1 and obj2 is copied to po2. So cc is called automatically to copy these objects.

Q16- What is a default copy constructor? What is the need for overloading default copy constructor?

Answer-16
A default cc is provided by the compiler. This const is called when one obj is copied to another.
Some issue may arise when we copy pointer members from one obj to another. So to avoid these issues with pointers, the def cc is overloaded and a user defined cc is provided.

Q17- On **overloading copy constructor**
create a class student. Overload the default copy constructor for this class.
Create two objects obj1, obj2.
Copy obj1 to obj2 like this -
obj2=obj1; //overloaded copy const will be called.

Answer-17

#include<iostream>

```
using namespace std;
class student
{
	private:
	int rn, m1,m2;
	public:
	void set(int , int , int);
	void get();
	void add();
	//There are two constructors

	//This is overloaded CC or user Defined CC
	student(student &po);

	//This is a blank constructor for creating obj
	student();


};
void student::set(int x, int y, int z)
{
	rn=x;
	m1=y;
	m2=z;
}
void student::get()
{
	cout<<rn<<endl<<m1<<endl<<m2<<endl;
}
void student::add()
{
	cout<<m1+m2<<endl;
}
student::student(student &po)
{
	rn=po.rn;
	m1=po.m1;
	m2=po.m2;
	cout<<"Overloaded Copy Constructor called"<<endl;
```

```
}
student::student()
{

}
int main()
{
        //These obj are created by using blank constructor.
        student obj1, obj2;
        obj1.set(1,11,11);
        obj1.get();
        obj1.add();

        obj2.set(2,22,22);
        obj2.get();
        obj2.add();

        //Two ways to call CC

        //By using assignment operator (=)
        //Here copy constructor is called
        student obj3=obj1;//Copy constructor called. obj1 copied to obj3


        //By calling the CC explicitly
        //Pass the second obj as parameter to first obj
        student obj4(obj1);//Copy constructor called. obj1 copied to obj4

   obj3.get();
   obj4.get();

}
```

Q18- Write some differences between calloc and malloc.