Submitted by: Mahesh Singh Madai

#2. Model Question-2075

2) Explain operator overloading? Write a program that overloads insertion and extraction operators.

Ans

Operator overloading provides a flexible option for the creation of new definition for most of the c++ operators.

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

Program:

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

class Distance{
    int feet;
    int inches;
public:
    Distance(){
      feet = 0;
      inches = 0;
    }
    Distance(int f,int i)
    {   feet = f;
        inches = i;
    }
```

```
    friend ostream & operator<< (ostream & output, const Distance &
    {
      output << "F: " << D.feet << "I: " << D.inches;
      return output;
    }
    friend istream & operator >> (istream & input, Distance &D)
    {
      input >> D.feet >> D.inches;
      return input;
    }
};
int main()
{
    Distance D1 (11,10), D2 (5,11), D3;
    cout<<"Enter the value of object: "<<endl;
    cin>> D3;
    cout<<" First Distance: "<<D1 << endl;
    cout<<" Second Distance: " << D2 <<endl;
    cout<<" third distance: "<< D3 <<endl;

    return 0;
}
```

4. Explain the purpose of a namespace with suitable example.

AN

The namespace defines a scope for classes, functions, blocks etc to define a scope that could hold global identities.

eg: All classes, functions and templates are declared within the namespace named 'std'.

ie using namespace std;

Example:-

```
namespace Testspace
{
    int m;
    void display(int n)
    {
        cout<<n;
    }
}
```

Now, we can create a declarative and use the functions inside by using it

5) What is the principle reason for passing arguments by reference? Explain with suitable code.

Ans The pass by reference mechanism is useful in object-oriented programming because it permits manipulating of objects by reference and eliminates the copying of object parameters back and forth.

example:

```
#include<iostream>
using namespace std;
void swap(int &, int &);
void main()
{
    int a,b;
    cout<<"Enter the numbers:"<<endl;
    cin>>a>>b;
    cout<<"Before swap"<<endl<<a<<b;
    cout<<"After swap"<<endl;
    swap(a,b);
    cout<<"After swap"<<endl<<a<<b;
}
void swap(int &x, int &y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```

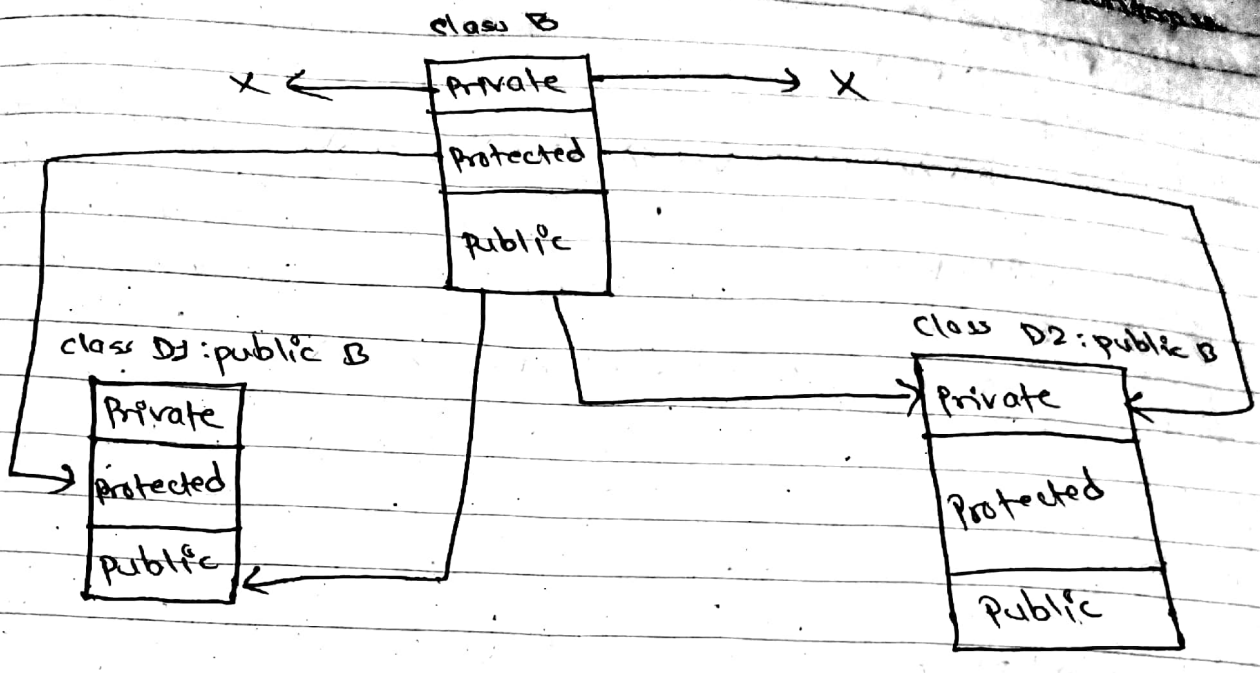
7) Write a program that

7) Write a program that illustrates the conversion between objects of different classes having conversion function in source object

Ans
```
#include <iostream>
#include <conio.h>
#define PI 3.142857
using namespace std;
class Radian
{
    float Rad;
public:
    Radian() {
        rad = 0.0; }
    }
    Radian(float r)  -(1)
    {   rad = r;
    }
    float getradian()
    {   return rad;
    }
    void display() -(2)
    {
        cout<<"Radian= "<< getradian();
    }
};
class Degree
{
    float degree;
```

```
public:
    Degree()
    {
        degree= 0.0;
    }
    operator Radian()
    {
        float radian;
        radian= degree * PI/180.0;
        return Radian(radian); ← (1)
    }
    void input()
    {
        cout<<"\n Enter the degree :"<<endl;
        cin>> degree;
    }
};
void main()
{
    Degree d1;
    Radian r1;
    d1.input();
    r1 =d1;
    r1. display(); ← (2)
    getch();
}
```

8. Explain the difference between public and private inheritance with suitable diagram.



9) ~~Write~~ Write a program that explains the late binding using virtual function.

⇒
```
class base
{
  public:
    virtual void print()
    {
      cout<<" Print base class "<<endl;
    }
    void show()
    {
      cout<<" show base class "<<endl;
    }
};
```

```
class derived: public base
{
    public:
        void print()
        {
            cout<<" Print derived class" <<endl;
        }
        void show()
        {
            cout<<" show derived class" <<endl;
        }
};
int main()
{
    base *bptr;
    derived d;
    bptr = &d;

    bptr -> print();
    bpt -> show();

}
```

y do we need exceptions? Explain 'exception with argument' with suitable program.

Exceptions provide a way to react to exceptional circumst in programs by transferring control to special functions called handlers.

Exception handling provides a type-safe, integrated approach for coping with unusual predictable problems that arises while executing a program.

Exception with arguments:

⇒ Program:

```
void divide(int x, int y, int z)
{
    cout<<" In we are inside the function";
    if((x-y)!=0)
    {
        int R= z/(x-y);
        cout<<R<<endl;
    }
    else
    {
        throw (x-y);
    }
}
int main()
{
    try
    {
        cout<<"We are inside the try blocks";
        divide (10,20,30);
        divide (10,10,20);
    }
```

```
catch (int i)
{
    cout<<" Caught the exception ";
}
return 0;
}
```

2) What are the advantages of using the stream classes for I/O? Write a program that writes object to a file.

Ans The advantages of using stream classes for I/O are:-

i) The input (>>) and output (<<) operator are 8 types of. These are easier to use.

ii) We can overload the input and output operators to define input and output for your own types and classes.

Program for writing object to a file :-

```
class student
{
    int roll;
    char name[30];
public:
```

```
void input()
{
    cout<<" Enter the roll no & name: " <<endl;
    cin>> roll >> name;
  }
void display()
{
    cout<<" Roll no" << roll << "Name :" <<name;
}
};
void main()
{
  ofstream outf("myfile.txt");
  student s1;
  s1 input();
  outf.write((char *)&s1, sizeof(s1));
  cout<<" write success";
  outf.close();
  student s2;
  ifstream infile("myfile.txt");
  infile.read ((char *) &s2, sizeof (s2));
  infile.close();
  s2.display();
```