

1. Given the class declaration:

```
class Rectangle{
private:
    int length;
    int width;
public:
    Rectangle(int l=0, int w=0) {length = l; width = w; }
    int getL() { return length;}
    int getW() { return width; }
};
```

a) Assume the following code fragment:

```
//a, b, c, d are initialized
Rectangle r1(a, b), r2(c, d), r3;
r3 = r1 + r2;
```

Overload + operator with the help of a member function so that the **length** member of object **r3** after this fragment is **a+c**, and the **width** member of **r3** is **b+d** (values **a**, **b**, **c**, **d** must come as private members of corresponding objects!). Show the following:
Function prototype inside the class declaration:

Function definition how it will appear outside the class declaration:

b) Overload the same operator with the help of a **friend** top-level function. Show the following:

Function prototype inside the class declaration:

Function definition outside the class declaration:

In parts c) - e) show only **definitions** of the functions how they are written **outside** the class declaration:

c) For the same class **Rectangle**, overload postfix version of **--** operator as a member function so that when it is executed both the **length** and **width** members of the **rectangle** object will be decremented by 1.

d) Overload the output operator **<<** for the class **Rectangle**. The result of the overloading is that the fragment

```
Rectangle r(m, n);
cout << r;
```

produces the rectangle of **m** lines, each line containing **n** symbols of **#** 's and a new line starts after that.

e) Overload the input operator **>>** for the class **Rectangle** so that when the following fragment is executed, it prompts the user to enter the length and the width of the rectangle, and these values become private members of an object **r**:

```
Rectangle r;
cin >> r;
```

2. Given the class declaration:

```
class FloatArr{
private:
    float * b;
    int size;
    float dummy_val;
public:
    FloatArr(int s) {size = s; b = new float [size];}
    float & operator[](int );
// the prototype of an overloaded operator= function must be shown
// in part (b) of this problem
};
```

(a) Overload = so that elements of one array will be assigned to the corresponding elements of another array, and assignment will be performed only for arrays of the same size. Show the function definition *outside* the given class.

(b) Write the prototype of the function from part (a) how it will appear in the class declaration:

(c) Given the definition of the overloaded subscript operator:

```
float & FloatArr::operator[](int i)
{
    if(i<0||i>size-1)
    {
        cout << "Index value of " << i;
        cout << " is out of bounds.\n";
        return dummy_val;
    }
    float min = b[0];
    int position = 0;
    for(int k=1; k<i; k++)
        if(min>b[k])
            {min=b[k]; position=k;}
    return b[position];
}
```

Show what will be displayed when the following code fragment is executed:

```
FloatArr ob1(7);
// elements of ob1.b are initialized to:
// 5, 7, 2, 6, 4, 1, 9 (in this particular order)
cout << " ob1[3] is: " << ob1[3] << endl;
cout << " now ob1[15]: " << ob1[15] << endl;
```

3. Mark the following statements as *TRUE* or *FALSE*:

(a) It's impossible to use the = operator to assign one object's value to another object unless you overload this operator. | *TRUE* *FALSE*

(b) All binary operators must always be overloaded as top-level friend functions.	<i>TRUE FALSE</i>
(c) Inheritance is a "has-a" relation between classes.	<i>TRUE FALSE</i>
(d) A derived class must always have a constructor.	<i>TRUE FALSE</i>
(e) It is possible for a base class to have more than one constructor.	<i>TRUE FALSE</i>
(f) A member function of a derived class is not allowed to have the same name as a member function of the base class.	<i>TRUE FALSE</i>
(g) An abstract base class cannot be instantiated.	<i>TRUE FALSE</i>
(h) Under public inheritance, protected members of a base class become public members of a derived class.	<i>TRUE FALSE</i>

4. Mark the lines in `main()` that will not compile. What values will be printed by the other `cout` statements of this program?

```

#include <iostream>
using namespace std;

class Base{
private:
    int i;
protected:
    int j;
    int get_i()
    { return i;}
public:
    int k;
    Base()
    {i=2; j=4; k=6;}
};

class Der1: public Base{
public:
    Der1(){k = 17;}
    int getI()
    {return get_i();}
};

class Der2: public Der1{
public:
    Der2(){k = 15;}
};

int main()
{
    Base b;
    Der1 d1;
    Der2 d2;

    cout << b.i << endl;
    cout << b.j << endl;
    cout << b.k << endl;
    cout << b.get_i();
    cout << endl;

    cout << d1.i << endl;
    cout << d1.j << endl;
    cout << d1.k << endl;
    cout << d1.get_i();
    cout << endl;
    cout << d1.getI();
    cout << endl;

    cout << d2.i << endl;
    cout << d2.j << endl;
    cout << d2.k << endl;
    cout << d2.getI();
    cout << endl;
    return 0;
}

```

5. Show the output of the following programs:

```
a)
#include <iostream>
using namespace std;
class One{
public:
    int F(){return 2;}
    virtual int G(){return 3;}
};
class Two: public One{
public:
    int F(){return 4;}
    int G(){return 5;}
};
class Three: public Two{
public:
    int G(){return 7;}
};
int main()
{ One * p1;
  One ob1;
  Two ob2;
  Three ob3;
  p1 = &ob1;
  cout << p1->F() * p1->G();
  cout << endl;
  p1 = &ob2;
  cout << p1->F() * p1->G();
  cout << endl;
  p1 = &ob3;
  cout << p1->F() * p1->G();
  cout << endl;
  Two * p2;
  p2 = &ob3;
  cout << p2->F() * p2->G();
  cout << endl;
  return 0;
}
```

```
b)
#include<iostream>
using namespace std;
class BC{
public:
    void show_class()
    { show1();
      show2();}
private:
    void show1()
    { cout << "#1 BC\n"; }
    virtual void show2()
    { cout << "#2 BC\n"; }
};
class DC : public BC{
private:
    void show1()
    { cout << "#1 DC\n"; }
    virtual void show2()
    { cout << "#2 DC\n"; }
};
class DDC : public DC{
private:
    void show1()
    { cout << "#1 DDC\n"; }
    virtual void show2()
    { cout << "#2 DDC\n"; }
};
int main()
{ BC b;
  b.show_class();
  cout << endl;
  DC d;
  d.show_class();
  cout << endl;
  DDC dd;
  dd.show_class();
  cout << endl;
  return 0;
}
```

6. Fill in the blank line so that a static variable `m` is assigned the value 3, and show the output of the following program:

```
class Demo{
protected:
    static int m;
public:
    Demo()
    { cout << "Constructor\n";
      m++;
    }
    ~Demo()
    { cout << "Destructor\n";
      m-=3;
    }
    void show_m()
    { cout << "m now is " << m << endl;}
};
class Der:public Demo{
public:
    Der()
    { cout << "Der Constructor\n";
      m+=2;
    }
    ~Der()
    { cout << "Der Destructor\n";
      m-=1;
    }
};
// in the line below define a static variable M
```

```
int main()
{ Demo a, b;
  a.show_m();
  b.show_m();
  { Demo c;
    c.show_m();
  }
  { Der d;
    d.show_m();
  }
  a.show_m();
  return 0;
}
```

OUTPUT:

7. (a) Fill in the blank line according to the instruction;

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

class First{
protected:
    int a;
public:
    First(int x=1){ a=x;}
    virtual void Twist(){ a*=2; cout << a << endl;}
    int getValue() { Twist(); return a;}
};

class Second : public First{
private:
    int b;
public:
    Second(int y=5){ b=y;}
    // in the space below override the function Twist, so
    // that it multiplies the current value of b by 10
    // and displays this new value:
    -----
};
```

(b) Write a test driver to demonstrate polymorphism in this class hierarchy.

(c) Show the output of your test driver:

8. Assume that the following code is saved in the file "Base.h":

```
using namespace std;
class base{
public:
    base(){ cout << " Base" << endl;}
    base(int i){ cout << " Base " << i << endl;}
    ~base(){ cout << " Base destr." << endl;}
};
class der : public base{
public:
    der(){ cout << "Der " << endl;}
    der(int j):base(j+1){cout << " Der " << j << endl;}
    ~der(){ cout << "Der destr." << endl;}
};
```

Show the output of the following test drivers:

(a)
`#include<iostream>`
`#include "Base.h"`

```
int main()
{
    {
        der d;
    }
    der d(5);
    return 0;
}
```

(b)
`#include<iostream>`
`#include "Base.h"`

```
int main()
{
    {
        der d;
    }
    {
        base * bp;
        bp = new der(5);
        delete bp;
    }
    return 0;
}
```

OUTPUT: