8. [8 points] A complex number is any number $a + bi$, where a is the real part, and b is the imaginary part. Consider the following code representing a complex number class, implement the overloaded addition (+) and multiplication (*) operators to perform complex number addition and multiplication correctly.

Hint: $(a + bi) + (c + di) = (a + c) + (b + d)i$

$(a + bi) * (c + di) = (a * c - b * d) + (a * d + b * c)i$

```
class ComplexNumber {
 public:
    double re, im;
    ComplexNumber( double x, double y) { re = x; im = y;}
};

//Your overloaded + operator here:
```

```
ComplexNumber operator+ (const ComplexNumber& c1, const ComplexNumber& c2){
 return ComplexNumber(c1.re + c2.re, c1.im + c2.im);
}
```

```
//Your overloaded * operator here:
```

```
ComplexNumber operator*(const ComplexNumber& c1, const ComplexNumber& c2){
 return ComplexNumber(c1.re*c2.re - c1.im*c2.im, c1.re*c2.im + c1.im*c2.re);
}
```

9. [6 points] In your main program, LinkedList is used as a stack. Assume LinkedList is declared as bel initialized as an global object (LinkedList ***L***).

```
struct LinkedList {
    struct Link {
            void* data;
            Link* next;
            void initialize(void* dat, Link* nxt);
      }* head;

      LinkedList();
      ~LinkedList();

      void addFront(void* d);
      void addBack(void* d);
      void removeBack();
};
```

Based on the available member functions of ***L***, implement a function ***void push(double d)*** that a value to the stack.

```
void push(double d){

      Link* = new Link = new Link;
      new Link = initialize(d, head);
      head = new LInk;

}
```

Based on the available member functions of ***L***, implement a function ***void pop()*** that removes the last value inserted from the stack.

```
void pop(){

    if(head = 0){
       return 0;
    }

    void* result = head -> data;
    Link* oldHead = head;
    head = head -> next;
    return result;

}
```

10. [12 points] What is the following code fragment printing?

```
class Object{
public:
      static int count;

      Object(){
          cout << "Object()"<< endl;
            cout << ++count << endl;
      }
      ~Object(){
          cout << "~Object()"<< endl;
         cout << --count << endl;
    }
};

int Object::count = 0;

Object f(Object someObject){
    return someObject;
}

int main(int argc, const char * argv[])
{
    Object myObject;
    Object another = f(myObject);
    return 0;
}
```

Output:
```
Object
1
~Object()
0
~Object()
-1
```

Modify Object class so that the object counts will be correct.

```
class Object{
public:
      static int count;

      Object(){
          cout << "Object()"<< endl;
            cout << ++count << endl;
      }
      ~Object(){
          cout << "~Object()"<< endl;
         cout << --count << endl;
    }


        Object(const Object&){
           cout << "Object()"<< endl;
           cout << ++count << endl;
        }


};
```

1. [20 points] Indicate whether the following statements are **True** or **False**:

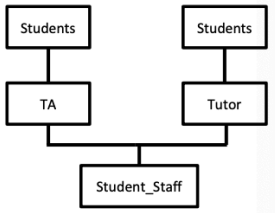
a.	Upcasting conversion is performed automatically by the compiler without any special operations (function calls).	(T) / F
b.	Downcasting conversion is performed automatically by the compiler without any special operations (function calls).	T / (F)
c.	We use virtual functions to store objects in virtual memory.	T / (F)
d.	With inline functions, the compiler creates compiled code for the functions to be linked from every function call.	T / (F)
e.	A constant member function prevents us from modifying the variable members of the class.	(T) / F
f.	Memory can be allocated dynamically from the "heap" memory by using **new** operation.	(T) / F
g.	In C++, a namespace allows us to use different variable names in the same memory location.	T / (F)
h.	If no exception is caught, **terminator()** is called automatically to end the program.	T / (F)
i.	Two main classes derived from the **exception** class are **runtime_error** and **compiletime_error**.	T / (F)
j.	The compiler allocates storage for a template when it is defined.	T / (F)
k.	References can be set to NULL.	T / (F)
l.	Each object stored in a container class must be derived from the same base class.	(T) / F
m.	Multiple inheritance allows us to add functionalities of a class to another class.	(T) / F
n.	A traits class in the standard library allows us to modify the content of a string.	T / (F)
o.	Vectors provide us faster insertions at the front than a linked list.	T / (F)
p.	Maps implemented using hash tables are called ordered maps.	T / (F)
q.	We can use an iterator to access contents in a linked list.	(T) / F
r.	An abstract base class is a class with its member functions declared as virtual functions.	T / (F)
s.	A virtual base class is a class with its member functions declared as virtual functions.	T / (F)
t.	Namespaces can be defined across multiple files.	(T) / F

2. [10 points] Add code to main and/or the CoolArray class to prevent this code from having a memory leak.

```
int main() {
  CoolArray* a = new CoolArray(100);
  //Add your code here
  delete a;
}
```

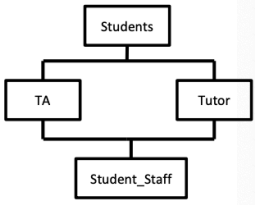
```
class CoolArray {
  int* arr;
 public:
  CoolArray(int size) {
    arr = new int[size];
  }
  //Add your code here
  //OR Add a destructor:
  ~CoolArray(){}




};
```

3. [8 points] The diagram below shows a multiple inheritance case where ***Student*** is duplicated. Write class definitions (with empty bodies) to create this hierarchy structure.




```
class Students{
};
class TA : public Students{
};
class Tutor : public Students{
};
class Student_Staff : public TA, public Tutor{
};
```

4. [8 points] The diagram below shows a multiple inheritance case where ***Student*** is shared. Write class definitions (with empty bodies) to create this hierarchy structure.




```
class Students{
};
class TA : virtual public Students{
};
class Tutor : virtual public Students{
};
class Student_Staff : public TA, public Tutor{
};
```

5. [9 points] What is the following code fragment printing?

```
class A{
public:
 int f ( int x ) { return (x + 5); }
};

class C : public A {
 public:
 int f ( int x ) { return (x - 5); }
};

int main() {
 C obj; cout << obj.f(15) << endl;
 C* c = new C; cout << c->f(15) << endl; delete c;
 A* a = new C; cout << a->f(15) << endl; delete a;
}
```

Output: 10
10
20

C obj -> obj.f() ->
15 - 5 = 10 print 10 ->
C* c = new C -> c -> f() ->
15 - 5 = 10 print 10 ->
A* a = new C -> a -> f() ->
15 + 5 = 20 print 20

6. [9 points] What is the following code fragment printing?

```
class A {
public:
 virtual int f ( int x ) { return (x + 5); }
};

class C : public A {
public:
 int f ( int x ) { return (x - 5); }
};

int main() {
 C o; cout << o.f(15) << endl;
 C* c = new C; cout << c->f(15) << endl; delete c;
 A* a = new C; cout << a->f(15) << endl; delete a;
}
```

Output: 10
10
10

Same as 5. except class A has a virtual function so class C's function is called instead

7. [10 points] Write a template function ***ArrayAve*** which takes in a pointer to an array of type ***T*** and the number of elements (***n***) in the array as input arguments. The function will return the average value of the array. Write your implementation of ArrayAve in the space below so that the program will run correctly.

```
//Implement your ArrayAve here:


        template <class T>
          class ArrayAve(T* arr, int size){
              float total = 0;
              for(int i = 0; i < size; i++){
                  total += T[i];
              }
              total = total / size;
              return total;
          };




int main()
{
    float ar[10];
    for(int i = 0; i < 10; i++)
        ar[i] = (float)i*1.1;
    float ave = ArrayAve<float>(ar, 10);
    cout << ave << endl;
}
```