CMPSC24 Midterm-I
Spring 2018

This exam is closed book, closed notes. You may use a one-page A4 size paper containing your notes. Write your name on the your notes paper and all other sheets of the exam. No calculators or phones are allowed in the exam. **Write all your answers on the answer sheet. All exam sheets, notes paper and scratch paper must be turned in at the end of the exam.**

By signing your name below, you are asserting that all work on this exam is yours alone, and that you will not provide any information to anyone else taking the exam. In addition, you are agreeing that you will not discuss any part of this exam with anyone who is not currently taking the exam in this room until after the exam has been returned to you. This includes posting any information about this exam on Piazza or any other social media. Discussing any aspect of this exam with anyone outside of this room constitutes a violation of the academic integrity agreement for CMPSC24.

Signature: _____

Name (please print clearly):_____

Umail address: _____@umail.ucsb.edu

Perm number: _____

You have 75 minutes to complete this exam. Work to maximize points. A hint for allocating your time: if a question is worth 10 points, spend no more than 5 minutes on it if a question is worth 20 points, spend no more than 10 minutes on it etc. If you don't know the answer to a problem, move on and come back later. Most importantly, stay calm. You can do this.

WRITE ALL YOUR ANSWERS IN THE PROVIDED ANSWER SHEET IN PEN OR DARK PENCIL!

DO NOT OPEN THIS EXAM UNTIL YOU ARE INSTRUCTED TO DO SO.
GOOD LUCK!

Name: _____

Q1 [30 points] A complex number has the form ***a + b*i,*** where ***a*** and ***b*** are real numbers and ***i*** is the square root of -1(negative one). We refer to ***a*** as the real part and ***b*** as the imaginary part.

a. (10 pts) Provide the definition of a class called **`Complex`** that represents a complex number. Do not implement the methods yet. Your class definition should satisfy the following requirements:
   - (2 pts) declare two private data members to represent the real and imaginary parts of the complex number
   - (2 pts) declare a constructor that takes two arguments to initialize the data members. The constructor should additionally provide default values (zero) to both member variables.
   - (2 pts) declare the overloaded +operator that operates on two objects of **`Complex`** and creates a new **`Complex`** object whose real and imaginary parts are the sum of the corresponding parts of the two objects. For example if **`c1`** represents: ***2.0 + 5.0*i*** and **`c2`** represents: ***8.0 + 20.0*i,*** then **`c1 + c2`** should represent the quadratic expression: ***10.0 + 25.0*i*.*** You may provide the declaration of this operator as a public member function or a non-member function that may be a friend.
   - (2 pts) declare the overloaded the operator << as a **non-member** function to print the value of a complex number. For example, if **`c1`** represents the complex number: ***2.0+5.0*i*** , then **`cout<<c1;`** should print ***2.0 + 5.0*i*** on the terminal. Provide the declarations for any other accessor or modifier functions that you need in your implementation. You may alternatively make this function a friend function.
   - (2 pts) differentiate between accessors and modifiers by making appropriate use of the keyword **`const`**

b. (20 pts) Implement all the methods of your class (from part a), specifically:
   (i) the parameterized constructor,
   (ii) the overloaded +operator
   (iii) the overloaded << operator
   (iv) any other methods that you used in your implementations in (i) - (iii)

Q2 [20 points] Assume that you have implemented the **`Complex`** class from the previous question correctly. Read the code below and answer the questions that follow:

```
void complexFun(){
     Complex c1(1.0, 10.0), c2(2.0, 5.0), c3;
     Complex *p = new Complex(c1);
     Complex *q = &c2;
     Complex *r;
     c3 = c1 + c2;
     cout<<c3<< " "<<*p<<  " "<<*q<<endl;
     c2 = c1;
     r = q;
     cout<<c3<< " "<<*r<<  " "<<*q<<endl;
     return;
}
```

Name: _____

a. (6 pts) Suppose that the function **`complexFun()`** is called and the code within the function is executed until it reaches the **`return`** statement. Assume the **`return`** statement has not been executed yet. Draw a pointer diagram showing all the objects on the heap and stack and their relationship. Label your diagram for full credit.

b. (4 pts) What is the output of the function **`complexFun()?`**

c. (2 pts) The **copy-assignment** operator of **`Complex`** is called as a result of which of the following statements. The variables **`c1,c2, c3, p, q`** and **`r`** are defined in the function **`complexFun().`** *SELECT ALL THAT APPLY*

   A. `c3 = c1 + c2;`
   B. `c2 = c1;`
   C. `r = q;`
   D. **None of the above**

d. (2 pts) The **copy-constructor** of **`Complex`** is called as a result of which of the following statements, assuming **`x`** is an existing instance of **`Complex .`** *SELECT ALL THAT APPLY*

   A. `Complex y(x);`
   B. `Complex y = x;`
   C. `Complex* y = new Complex(1, 10);`
   D. `Complex* y = new Complex(x);`
   E. **None of the above**

e. (2 pts) Consider the following two declarations of the functions **`f()`** and **`g():`**

```
void f(Complex c);
void g(Complex& c);
```

The copy-constructor of **`Complex`** is called when which of the following functions is called? SELECT ALL THAT APPLY

   A. `f()`
   B. `g()`
   C. **None of the above**

f. (2 pts) Write two reasons for passing parameters to functions by reference.

g. (2pts) Does the following code have a memory leak? Justify your answer.

```
Complex* createAnotherComplexNumber(const Complex& c){
      Complex* nc = new Complex(c);
      return nc;
}
int main(){
      Complex a(10, 50);
      Complex* p = createAnotherComplexNumber(a);
      p = new Complex(a);
      delete p;
}
```

Name: _____

Q3 [50 points] Linked-list

Consider the definition of the class **`LinkedList`** used in the construction of a singly linked-list containing integer elements.

```
class LinkedList {

public:
    // ASSUME ALREADY IMPLEMENTED
    LinkedList(){head = 0; tail = 0;}     //Default constructor
    ~LinkedList();                        // De-constructor
    LinkedList(const LinkedList& source); //Copy constructor
    LinkedList& operator=(const LinkedList& source); //Copy assignment

    // YOU MUST IMPLEMENT THESE:
    void insertFront(int value);
    void append(int value);
    LinkedList(const int* arr, int len);  //Parameterized constructor
    friend LinkedList operator*(const LinkedList& ll, int k);

private:
    class Node{
       public:
            int data;          // data element
            Node* next;        // pointer to the next node in the list
            Node (const int& d) { data = d; next = 0;}
    };
    Node* head; // pointer to the first node in the linked-list
    Node* tail; // pointer to the last node in the linked-list
};
//NON-MEMBER OVERLOADED * OPERATOR
LinkedList operator*(const LinkedList& ll, int k);
```

Assume that the following methods have already been implemented

```
    LinkedList(){head = 0; tail = 0;}     //Default constructor
    ~LinkedList();                        // De-constructor
    LinkedList(const LinkedList& source); //Copy constructor
    LinkedList& operator=(const LinkedList& source); //Copy assignment
```

You may use any of the above in your implements for the following questions:

a. **(10 pts)** Implement the **`insertFront()`** method of the class. This method takes an integer value and adds a Node with that value to the front of the linked list. The method should handle all edge cases including insertion into an empty list.

b. **(10 pts)** Implement the **`append()`** method of the class. This method takes an integer value and adds a Node with that value to the end of the linked list. The method should handle all edge cases including insertion into an empty list.

c. **(15 pts)** Implement the parameterized constructor
`LinkedList(const int* arr, int len);`
The constructor should take as input a pointer to an integer array and the length of the array. It should create a linked list whose nodes have the same values as the elements of the array. See example usage below:

```
int array[]={10, 50, 70, 20, 90};
LinkedList ll(array, 5);
// ll should represent the following linked list
// 10->50->70->20->90->null
```

Element 10 is the first node and 90 is the last node in the linked list

Assume that you have correct implementations of **`insertFront()`** and **`append()`.** Use whichever is appropriate in your implementation. Recall that both **`insertFront()`** and **`append()`** detect an empty linked list based on the values of the **`head`** and **`tail`** pointers, and do not make any other assumptions about the state of the linked list.

d. **(15 pts)** Implement the non-member friend scaling operator **`*`**:
`LinkedList operator*(const LinkedList& source, int k);`

This operator should create a new linked list that contains the sequence of numbers in the linked list **`source`** scaled by a constant **`k`.** See the example usage below:

```
int array[]={10, 50, 70, 20, 90};
LinkedList list1(array, 5);
//list1 now contains the sequence 10->50->70->20->90->null

LinkedList list2;
list2 = list1*10;
//list2 now contains the sequence 100->500->700->200->900->null
//list1 should not be modified

list1 = list1*2;
//list1 should contain the sequence 20->100->140->40->180->null
//list2 should not be modified

list2 = list1*0;
//list2 should contain the sequence 0->0->0->0->0->null
//list1 should not be modified
```

Name: _____

Name: _____

Scratch Paper