CMPSC24 Midterm-II
Winter 2018
2/26/2018

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!

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

Name: _____

Q1 [20 points] A one variable quadratic expression is an arithmetic expression of the form ***a*x^2 + b*x + c,*** where **a, b** and **c** are some fixed numbers (called the coefficients) and **x** is a variable that can take on different values.

a. (8 pts) Provide the definition of a class called **quadratic** to store information about a quadratic expression. Do not implement the methods yet. Your class should satisfy the following requirements:
   - (2 pts) include a parameterized constructor that sets all three coefficients. If no coefficient values are provided to the constructor, it should set all coefficients to zero by default.
   - (2 pts) include a member function called **evaluate()** that takes a specific value of `x` as a parameter and returns the value of the expression: ***a*x^2 + b*x + c*** for the given value of **x**
   - (2 pts) include an overloaded +operator that takes two objects of **quadratic** as arguments and creates a new **quadratic** whose coefficients are the sum of the corresponding coefficients of the inputs. For example if `q1` represents: ***2.0*x^2 + 5.0*x + 100*** and `q2` represents: ***8.0*x^2 + 20.0*x + 200,*** then **q1 + q2** should represent the quadratic expression: ***10.0*x^2 + 25.0*x + 300***
   - (2 pts) differentiate between accessors and modifiers by making appropriate use of the keyword **const**

b. (12 pts) Implement all the methods of your class (from part a), specifically: (i) the parameterized constructor, (ii) the **evaluate** function, and (iii) the overloaded +operator. You may use **pow(x,2)** to compute x^2

Q2 [10 points] Assume that you have implemented the **quadratic** class from the previous question correctly. Read the code below and answer the questions that follow:

```
bool foo(){
     quadratic m, s(2.0, 5.0, 10.0);
     quadratic *p = new quadratic(s);
     quadratic *q = &s;
     quadratic *w;
     m = s;      //Referred to in part b below
     w = p;      //Referred to in part b below
     return q->evaluate(10)== p->evaluate(10);
}
```

a. (6 pts) Suppose that the function **foo()** is called and the code within the function is executed until it reaches the return statement (not yet executed). Complete each row of the table provided to answer the following questions about **the objects in memory that the variable/expression in column 1 of the table refers to**.

b. (2 pts) The **copy-assignment** operator of **`quadratic`** is invoked as a result of which of the following statements. The variables **`m, s, p`** and **`w`** are defined in the function **`foo().`** *SELECT THE SINGLE BEST ANSWER*

A. `m = s;`
B. `w = p;`
C. Both A and B
D. None of the above

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

A. `quadratic y(x);`
B. `quadratic y = x;`
C. `quadratic* y = new quadratic(1, 10, 50);`
D. `quadratic* y = new quadratic(x);`
E. None of the above

Q3. [8 pts] Big-O Multiple choice.

a. (2 pts) What is the goal of Big-O analysis? *SELECT THE SINGLE BEST ANSWER*

A. To determine the running time of an algorithm in some unit of absolute time (seconds/minutes/years) assuming it was implemented on the fastest supercomputer in the world.
B. To analyze the running time of an algorithm on "large" inputs ignoring details related to implementation, language, compiler or underlying hardware.
C. Both A and B
D. None of the above

b. (6pts) For the following questions select the operation that is "faster" based on its Big-O running time? *Write A, B or C in your answer sheet in each case*

i. (2 pts) **Inserting** a value:

A. **Inserting** a value at the head of a linked-list
B. **Inserting** a value into a sorted array
C. Both are equally fast

ii. (2 pts) **Searching** for an element:

A. **Searching** for an element in a linked-list
B. **Searching** for an element in a sorted array using binary search
C. Both are equally fast

iii. (2 pts) **Finding the minimum** element:

A. **Finding the minimum** element in a sorted array
B. **Finding the minimum** element in a sorted linked-list with elements in ascending order
C. Both are equally fast

Name: _____

Q4. [6 pts] The number of primitive operations needed for two algorithms in terms of the size of the input `N` is given below.

a. (4pts) What is the Big-O run time in each case?
   (i) Algorithm A: $500*N*\log(N) + 10000*N^2 + 250$
   (ii) Algorithm B: $0.5* 2^N + 500*N^2$

b. (2 pts) According to your Big-O analysis in part a, which algorithm is faster? (Write Algorithm A, Algorithm B or both are equally fast)

Q5 [6 pts] The functions given below solve the same problem in two different ways. Each function takes as input an array of integers of size N, where N>0 and checks if the elements of the array form a palindrome. The function returns true if the array has the same sequence of elements when traversed forward or backwards, and false otherwise. What is the worst case Big-O running time for each of the provided implementations ? Give the tightest bound. Explain your reasoning to receive partial credit.

(i)

```
bool palindrome(int arr[], int N){
      for(int i=0; i< N/2; i++){
            if(arr[i]!=arr[N-1-i])
                  return false;
      }
      return true;
}
```

(ii)

```
bool palindrome(int arr[], int N){
      int reverse[N];
      for(int i=0; i< N; i++){
            reverse[i] = arr[N-1-i];
      }
      for(int i=0; i< N; i++){
            if(arr[i]!=reverse[i])
                  return false;
      }
      return true;
}
```

Name: _____

Q6 [20 points] Sorted Linked-list

Consider the definition of the class **`Node`** and class **`LinkedList`** used in **the construction of a singly linked-list containing integer elements (with no duplicates) organized in ascending order. Assume that the `insert()` method of the `LinkedList` has been correctly implemented although the implementation is not shown here.**

```
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;}
};

class LinkedList {
  public:
    LinkedList(){head = 0; tail = 0;}
    void insert(int value);
// adds a node to the list with the given value if the value does not
// exist already. Maintains the nodes in the list in ascending order
// when inserting a new node

    friend LinkedList operator+(const LinkedList& s1,
                                const LinkedList& s2);

  private:
    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& s1, const LinkedList& s2);
```

Name: _____

(a) **(10 pts)** Assume that a linked-list is already created in memory as represented by the following pointer diagram: **`mylist`** is a pointer to a **`LinkedList`** object. Note that all items in the list are in sorted (ascending) order.

Redraw the above diagram in your answer sheet. Starting with this diagram, draw a pointer diagram that shows all the the objects in memory as a result of executing the following statements.

```
LinkedList otherList(*mylist);
otherList.insert(15);
otherList.insert(25);
LinkedList thirdList;
thirdList = otherList;
```

- Label your diagram clearly, showing all pointers, **`LinkedList`** objects and nodes in the linked-list. Points will be deducted if your drawing is not legible.
- Your diagram should be based on the current implementation of **`LinkedList`** class in which the copy-constructor and copy-assignment operators have not been overloaded.
- Assume that the **`insert()`** method has been implemented. This method inserts a new node in the list with a given value if the value does not exist in the list. Nodes are maintained in sorted (ascending) order during insertion.

(b) **(10 pts)** Implement the overloaded operator+, which takes two source **`LinkedList`** objects as arguments. The operator creates a new linked list that contains a copy of all the nodes in the input lists in sorted (ascending) order . For example if the first list **`s1`** has nodes with data values **10, 20, 30, 70** and the second list **`s2`** has nodes with data values **20, 25, 40, 50, 80** , then the function returns a new list with data values **10, 20, 25, 30, 40, 50, 70, 80**. Your code should work on source linked-lists of any size including an empty list.

- For this question alone assume that the the copy-constructor, copy-assignment operator, and de-constructor have been overloaded to perform deep copy/assignment/deletions of the linked list
- Hint: Use the **`insert()`** method in your implementation to save time.

Name: _____

Scratch Paper