CMPSC24 Midterm-II
Spring 2017
5/24/2017

This exam is closed book, closed notes. You may use a one-page A4 size paper containing your nodes. 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: _____

Part 1 [10 points] Multiple Choice: ***Select the SINGLE best answer by filling in the circles provided in your answer sheet. You must shade your selection completely as shown below:***

○ Not selected ● Selected

1. **[2 pts]** Dereferencing a null pointer results in which of the following outcomes. *CHOOSE THE SINGLE BEST OPTION.*

A. No error
B. Compile-time warning
C. Compile-time error
D. Run-time error, specifically a segmentation fault.

2. **[2 pts]** What is the Big-O running time of the `reverseString()` function given below:. The function uses the stack container class of the C++ Standard Template Library to reverse the characters of the input string. Assume that N is the number of characters of the string `input`. *CHOOSE THE SINGLE BEST OPTION: the tightest bound.*

```
string reverse(string input){
    string result="";
    stack<char> s;
    for(int i=0; i< input.length(); i++){
        s.push(input[i]);
    }
    while(!s.empty()){
        result+= s.top();
        s.pop();
    }
    return result;
}
```

A. O(1)
B. O(N)
C. O(logN)
D. $O(N^2)$

Name: _____

3. **[2 pts]** In your implementation of the polynomial class in PA04 that used a doubly linked-list, why did you implement your own copy constructor instead of using the default copy constructor? *SELECT THE SINGLE BEST ANSWER*

A. There was no particular reason. The default copy constructor would have worked as well.
B. The assignment operator would not work if we used the default copy constructor
C. The copy constructor was implemented to avoid copying nodes with zero coefficient from the source polynomial.
D. The default copy constructor only does a shallow copy of the linked-list, while the copy-constructor implemented in PA04 performs a deep copy.

4. **[2 pts]** Which of the following versions of the polynomial class implementation provides the most **space efficient** representation for polynomials with a large degree and very few terms, for example: $10x+x^{200}$. Recall that the degree of the polynomial is the largest exponent of the term with a non-zero coefficient. *SELECT THE SINGLE BEST ANSWER*

A. A static array-based implementation, where only the coefficients are stored in the array, and the exponent of each term is the index of the coefficient in the array. This version was implemented in PA03 part 1.
B. A dynamic array-based implementation, where only the coefficients are stored in the array, and the exponent of each term is the index of the coefficient in the array. The size of the array is changed dynamically at run-time. This version was implemented in PA03 part 2
C. A linked-list based implementation where the coefficient and exponent of each term in the polynomial was explicitly stored in each node of the linked-list. Terms with zero coefficients were not stored. This version was implemented in PA04
D. All the above have the same space efficiency.

5. **[2 pts]** Consider the following definition of a Node class used in the construction of a doubly linked-list:

```
template class <Item>
class Node{
      public:
         Item data;   // data element
          Node* next; //pointer to the previous node in the list
          Node* prev; //pointer to the next node in the list
          Node (const Item & d) { data = d; next = 0; prev =0;}
};
```

Which of the following statements correctly creates a Node object on the **runtime stack** with data value set to the value of the integer **`myInt`**. Assume **`myInt`** has been declared and initialized. *SELECT THE SINGLE BEST ANSWER*

```
A. Node n(myInt);
B. Node<int> n;
C. Node<int> n(myInt);
D. Node <int> * p = new Node<int>(myInt)
```

Name: _____

Part 2 [30 points] Sorted Linked-lists (25 minutes)

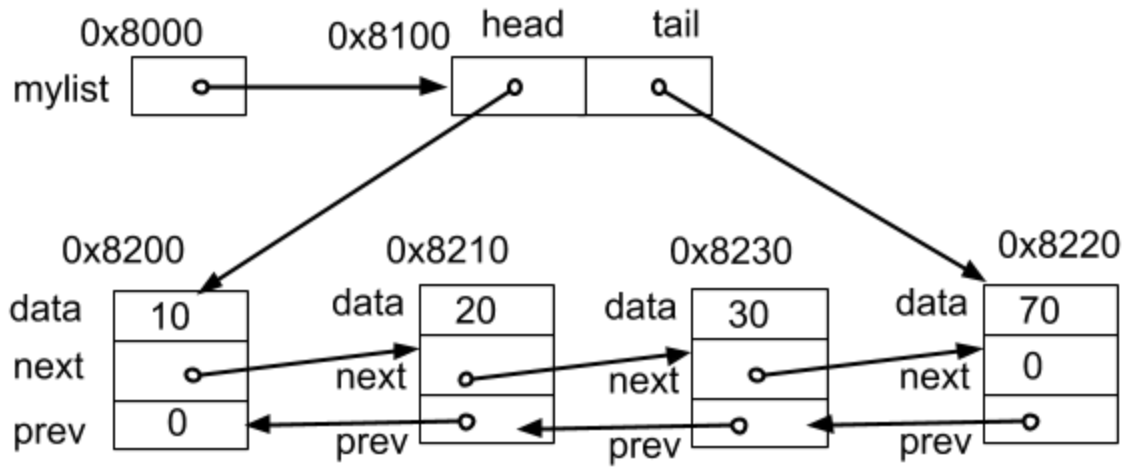
Consider the definition of the class `Node` and class `LinkedList` which were used in **the construction of a doubly linked-list containing integer elements (with no duplicates) in ascending order.**

```
class Node{
  public:
    int data;   // data element
    Node* next; //pointer to the next node in the list
    Node* prev; //pointer to the previous node in the list
    Node (const int& d) { data = d; next = 0; prev =0;}
};

class LinkedList {
  public:
    LinkedList(){head = 0; tail = 0;}     // constructor
    LinkedList(const LinkedList &source);     // copy-constructor
    ~LinkedList();                        // destructor
    void insert(int value);        // adds value to the list

  private:
    Node* head; // pointer to the first node in the list
    Node* tail; // pointer to the last node in the list
};
```

Assume that a linked-list is already created in memory as represented by the following pointer diagram: The hex numbers outside each box is the starting memory location of the corresponding data object. **`mylist`** is a pointer to a **`LinkedList`** object. Note that all items in the list are in sorted (ascending) order



Name: _____

1. **[5 pts]** Given the diagram on the previous page, write what each of the following C++ expressions evaluates to. Express your answer in hex or decimal as appropriate.

a. [1 pt] **`mylist`**
b. [1 pt] **`mylist->head`**
c. [1 pt] **`mylist->head->next`**
d. [1 pt] **`mylist->head->next->data`**
e. [1 pt] **`mylist->tail->prev->prev->data`**

2. **[10 pts]** Implement the copy constructor, which takes as argument another linked list and creates a duplicate of that list. Your copy constructor must do a deep copy of the source list. You may assume that the source list only has unique elements in sorted order, with the smallest element at the head of the list. You may use **`insert()`** in the implementation of your copy constructor. See the next question for a description of **`insert()`**. Your code should work on a source linked-list of any size including an empty list. If you prefer, attempt the next question first and then return to this question.

3. **[15 pts]** Implement the **`insert()`** method that takes as argument an integer value. If the value is already present in the list, the method should simply return. Otherwise it should insert a new node with the given value at the appropriate location in the list while maintaining the sorted order of elements. The smallest element should always be at the head of the list. Your code should work on a linked-list of any size including an empty list. Comment your code!!

Name: _____

Name: _____

Scratch Paper