Simon Fraser University

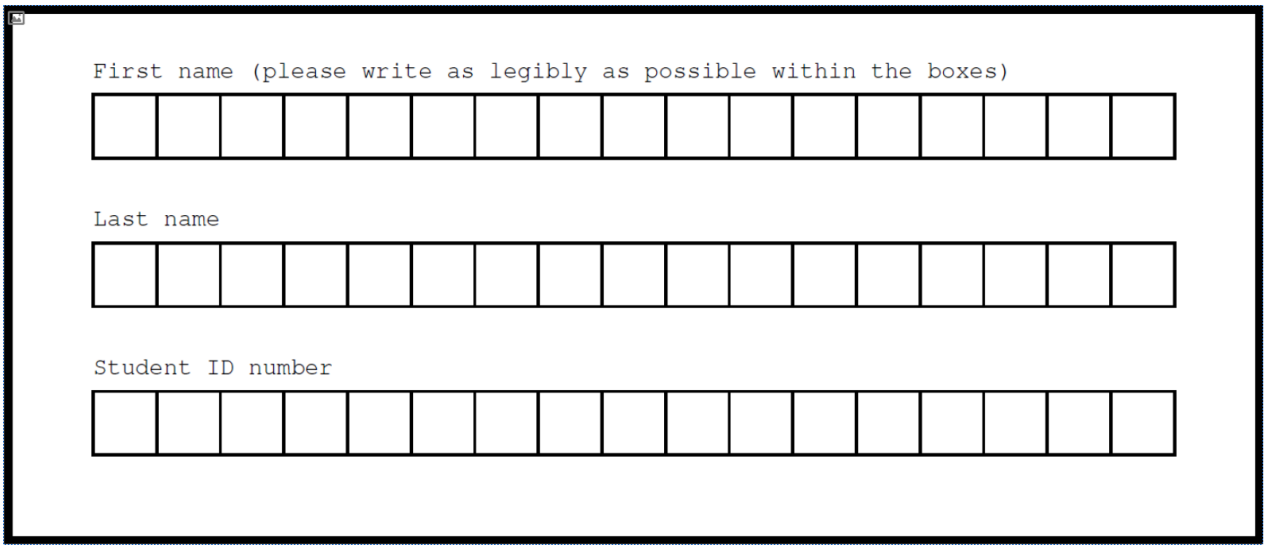
CMPT 225 D200

Spring 2024

Wednesday January 31

Practice Quiz 1

Time: 45 minutes – out of 45 marks



NOTE: The instructions below are the same instructions you will see on our Quiz 1.

This examination has 6 pages.

Read each question carefully before answering it.

- No textbooks, cheat sheets, calculators, computers, cell phones or other materials/devices may be used.
- Since this quiz is examining what you have learnt in this course, always use the algorithms, the definitions, i.e., the material you have seen in this course (in the lecture notes etc.) when answering the questions in this quiz.
- *ADT* means Abstract Data Type.
- List any assumptions you make when answering a question.
- **Always comment your code.**
- When ask to, always write as syntactically correct C++ code as you can. Use C++ code and syntax that can compile and execute on our *target machine*.
- Hand in your scrap sheet(s) of paper (if any) along with your quiz. The scrap sheet(s) will not be marked.
- The marks for each question are given in [ ]. Use this to manage your time.

Good luck!

Part 1 – Questions that require short answers. The weight of each question is indicated in [ ] – There are no part marks given!

1. [1 mark each] Express the time efficiency of the following operations using the Big O notation:

a. Retrieving a member (i.e., an object of the Profile class) from the FriendsBook application of our Assignment 1.	**Linear search: O(n)** **Binary Search: O($\log_2$ n)**
b. Popping an element from a Stack.	**O(1)**
c. Displaying the elements stored in a List designed as an abstract data type (ADT).	**O(n)**
d. Determining whether a value-oriented List is empty.	**O(1)**
e. Cloning an object of the Queue ADT class.	**O(n)**
f. Deleting the first element in a sorted array-based List.	**O(n)**

2. [1 mark] In order to answer the question d. in the above question 1, which assumption did you make? Answer: The List ADT class has a data member that keeps track of the number of elements currently held in the List such as `elementCount` .

3. [1 mark] See question e. in the above question 1. In order to allow a client code to clone an object of the Queue ADT class (assume the elements of this Queue are kept on the memory segment called the *heap*), which Queue method must we be offering as part of the public interface of our Queue ADT class? Answer: a Copy Constructor and we must overload the assignment operator.

4. [1 mark] Fill in the blank with the most appropriate set of words:

   The goal of Step 1 of the software development process is to _____

   _____ .

The goal of Step 1 of the software development process is to **understand (clarify, detail, disambiguate) the problem statement and requirements.**

5. [1 mark] State a possible class invariant for the Stack ADT class: **Possible answer: Lifo or Filo**

Part 2 – The weight of each question is indicated in [ ] – Write your answer below each question.

1. A credit card company has asked you to develop a computerized business system that would help them keep track of their large collection of customer data. The company would like to keep the information sorted by descending sort order of customer income, to perform insertions and deletions 6% of the time, to search by customer income 83% of the time and to print all the elements by descending sort order of customer income 11% of the time.

   To implement this system, we decide to use a List ADT class to store the large collection of customer data. Which implementation of the List ADT class would be the most efficient?

   a. [1 mark] Most efficient implementation of the List ADT class is:

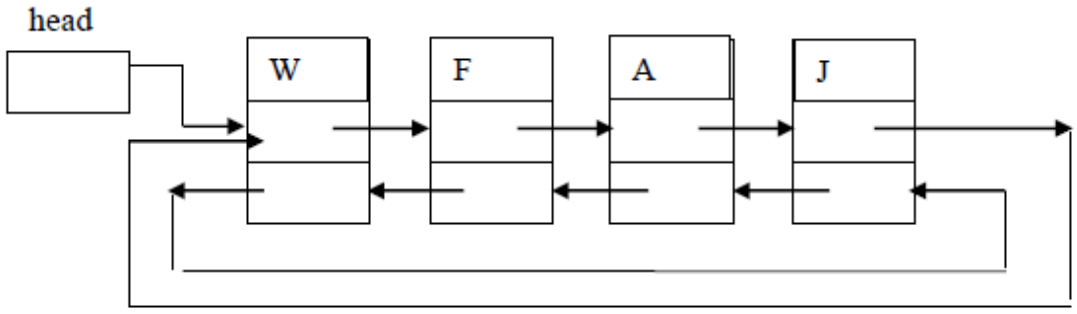
      Answer: array-based implementation

   b. [3 marks] Why (be as convincing as possible):

      Answer: Since 83% of the time is spent searching sorted data, we want to implement our List ADT class such that it searches very efficiently. Since our data is sorted, we can use either binary search or linear search. Binary search $O(\log_2 n)$ is more time efficient than linear search $O(n)$ and since binary search is easier to implement using an array than using a linked list, we shall use an array to implement our List ADT class.

2. [6 marks] Draw a circular singly-headed doubly-linked list containing the following 4 elements "W", "F", "A", and "J" in this order ("head" refers to "W"). Make sure your drawing is clear.

Answer:



3. [9 marks] A word **palindrome** is a word which reads the same backward or forward. Examples of palindromes are: *kayak*, *Anna*, and *rotator*. Write a function in C++ that makes use of one Queue object (instantiated from a Queue ADT class) and one Stack object (instantiated from a Stack ADT class) to figure out if a string (`word`) is a palindrome. Your function must return `true` when `word` is a palindrome or `false` otherwise.

```
bool palindrome( string word ) {
```

Possible Solution:

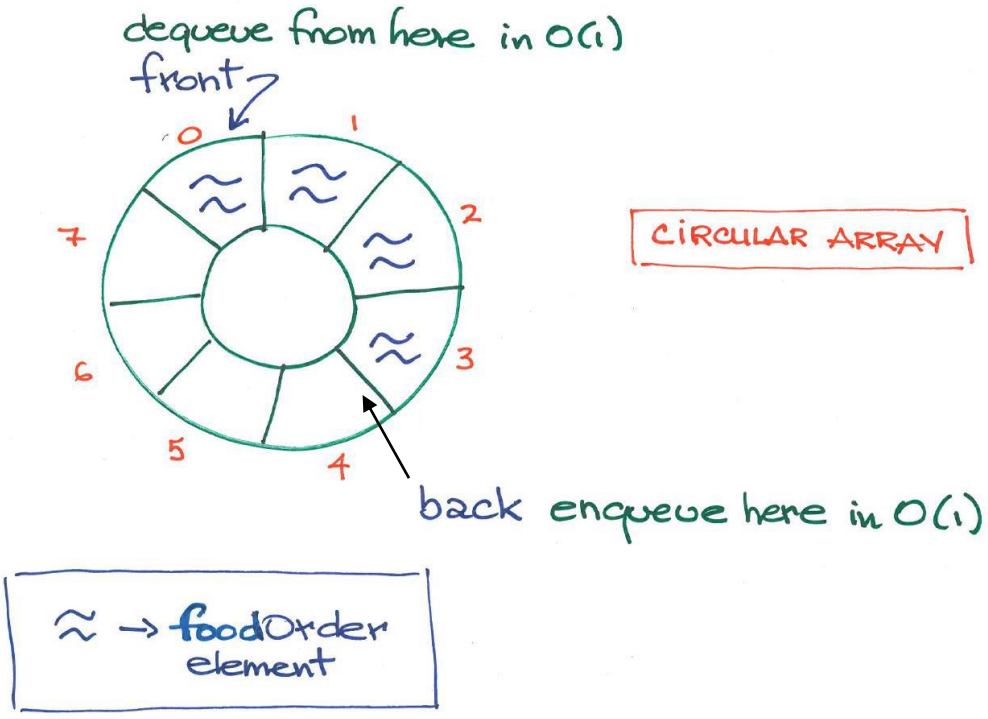
```
bool palindrome( string word ) {

  bool result = false;
  size_t wordLen = word.size();
  Stack<char> aStack;
  Queue<char> aQueue;


  // insert whole word onto the Stack and the Queue
  for ( unsigned int index = 0; index < wordLen; index++ ) {
    aStack.push(word[index]);
    aQueue.enqueue(word[index]);
  }

  // aStack.print(); // For testing purposes!
  // aQueue.print(); // For testing purposes!

  while ( !aStack.isEmpty() ) {  //  or aQueue.isEmpty()
    if ( aStack.peek() != aQueue.peek() ) {
      // the word is not a palindrome
      break;
    }
    aStack.pop();
    aQueue.dequeue();
  }

  if ( aStack.isEmpty() ) // the word is a palindrome
    result = true;

  return result;
}
```

4. You have been asked by the owner of a fast food restaurant chain to develop a fully automated "drive-through" application. This application is to allow drive-through customers to order their food from their car at a menu posted at the entrance of the drive-through (you know the kind!). As the food is being ordered by the car driver (who is ordering food for all occupants in the car), your application is to transform the verbal food order into an electronic food order, i.e., an object of foodOrder class, and quickly put this foodOrder element into a data collection ADT. When the kitchen is ready to prepare the next food order, your application is to quickly take this foodOrder element out of the data collection ADT and display it on the screen in the kitchen so that the short-order cooks can prepare the food.

   a. [1 mark] Which **data collection ADT**, seen so far in this course, would be the most appropriate to use in this "drive-through" application?

      Answer: ___Queue_____

   b. [6 marks] You decide to develop an **array-based implementation** of this data collection ADT you named above in a. Below, **draw its underlying data structure**. Indicate where, in this data structure, would a new foodOrder element be put and from where would a foodOrder element be taken out. Add any details you deem necessary for a reader to understand the drawing of your underlying data structure.



5. [9 marks] Consider the List ADT class from our Lab 3, write its method `removeLast()` as defined in the contract below. You cannot assume that this List ADT class has any other methods than the ones already defined in this class.

   Make sure to fill in the blank specifying the time efficiency of this method.

```
// Description: Removes the last node (i.e., element) in List
//              and returns the number of nodes left in the
//              List once the removal has been performed.
// Precondition: The List is not empty. If this List is empty,
//               this method returns 0.
// Time Efficiency: _____________________
unsigned int List::removeLast(){
```

   Possible Solution:

```
// Description: Removes the last node (i.e., element) in List
//              and returns the number of nodes left in the
//              List once the removal has been performed.
// Precondition: The List is not empty. If this List is empty,
//               this method returns 0.
// Time Efficiency: O(n)
unsigned int List::removeLast(){
  unsigned int result = 0;

  // Check to see if list is empty
  if (head != nullptr){
    // Move to the end of the list
    Node * previous = nullptr;
    Node * current = head; // Anchor

    while (current->next != nullptr){
      previous = current;
      current = current->next;
    }

    // Return node to the system
    Node * nodeToRemove = current;
    // nodeToRemove->next = nullptr;
    delete nodeToRemove;
    nodeToRemove = nullptr;
    // Remove the last node from the linked list
    if (previous != nullptr) previous->next = nullptr;
    elementCount--;
    result = elementCount;
  }

  return result;
}
```