

CS 14: Data Structures and Algorithms

Monday, October 20, 2003

Quiz 4, Form:

Name: _____

Login: _____

ID Number: _____

Signature: _____

This quiz is worth **25 points** and lasts **15 minutes**. You may use **ONLY** the test's last sheet (front and back pages) for **scratch work**, and your final answers must be **BRIEF, CLEAN, COHERENT, LEGIBLE**, and written **ONLY** in the boxed spaces provided.

Good luck!

1. [5 points] In a list of 10 elements, suppose that the elements with indices 2, 3, and 8 have reference counts of 1, the elements with indices 4 and 7 have reference counts of 2, and all other elements have reference counts of zero. Assuming that indices start at zero, answer the following questions:

(2 points) How many iterators are currently traversing the list? Why?

(3 points) Which elements can be safely deleted? Why?

2. [3 points] The array implementation of a List of characters has an array, `char a[]`, and an unsigned int size variable to keep track of the length of the list. Write a **simple** C++ implementation for the List member function `void List::insertLast(char c)`, which inserts a character at the **end** of the list. Don't worry about checking for list-full conditions.

3. [5 points] The simplest array implementation of a Queue of doubles has an array, `double a[]`, and an `int back` index variable to keep track of the back of the queue. Write a **simple** C++ implementation for the Queue member function `double Queue::dequeue()` which removes and returns the **double** located at the **front** of the queue. Don't worry about checking for a queue-empty error.

4. [12 points] What is the worst-case runtime complexity, in Big-Oh notation, of **deleting** the **last** element for each of the following data structures? **Explain** your answer.

(3 points) singly-linked list with a tail:

(3 points) singly-linked list without a tail:

(3 points) doubly-linked list with a tail:

(3 points) doubly-linked list without a tail:

Answer Key for Exam A

This quiz is worth **25 points** and lasts **15 minutes**. You may use **ONLY** the test's last sheet (front and back pages) for **scratch work**, and your final answers must be **BRIEF, CLEAN, COHERENT, LEGIBLE**, and written **ONLY** in the boxed spaces provided.

Good luck!

1. [5 points] In a list of 10 elements, suppose that the elements with indices 2, 3, and 8 have reference counts of 1, the elements with indices 4 and 7 have reference counts of 2, and all other elements have reference counts of zero. Assuming that indices start at zero, answer the following questions:

Answer: There are 7 iterators currently traversing the list, since each reference count value corresponds to how many iterators are *stationed* at each element: 3 elements have 1 iterator each and 2 elements have 2 each, for a total of $2 \times 2 + 3 \times 1 = 7$. Only the elements with reference count 1 can be safely deleted. You cannot delete elements which have no iterators positioned at them (reference count of zero), since you have no way of referring to those elements in the first place, and you cannot safely delete any element with more than one iterator positioned at it because you'll be invalidating the other iterators positioned at that element.

2. [3 points] The array implementation of a List of characters has an array, `char a[]`, and an `unsigned int size` variable to keep track of the length of the list. Write a **simple** C++ implementation for the List member function `void List::insertLast(char c)`, which inserts a character at the **end** of the list. Don't worry about checking for list-full conditions.

Answer:

```
void List::insertLast(char c)
{
    if ( isFull() )
        throw list_exception(" List full error.");
    else
        a[++size] = c;
}
```

3. [5 points] The simplest array implementation of a Queue of doubles has an array, `double a[]`, and an `int back` index variable to keep track of the back of the queue. Write a **simple** C++

implementation for the Queue member function **double Queue::dequeue()** which removes and returns the **double** located at the **front** of the queue. Don't worry about checking for a queue-empty error.

Answer:

```
double Queue::dequeue()
{
    if ( isEmpty() )
        throw queue_exception("Queue empty error.");
    else
    {
        // save the element at the front
        double front_value = a[0];

        // move every element one slot towards the front
        for (int i = 0; i < back; ++i)
            { a[i] = a[i+1]; }

        // update 'back' since we now have one less element
        --back;

        // return the element originally at the front
        return front_value;
    }
}
```

4. [12 points] What is the worst-case runtime complexity, in Big-Oh notation, of **deleting** the **last** element for each of the following data structures? **Explain** your answer.

Answer: singly-linked list with or without a tail: $\mathcal{O}[n]$, since we need to traverse the list from the beginning to be able to access both the last and next-to-last elements.
doubly-linked list without a tail: same as above.
doubly-linked list with a tail: $\mathcal{O}[1]$, since we can access the last element and, from it, the next-to-last element, both in constant time.