Com S 228 Introduction to Data Structures

Exam 2

6:30-7:30pm
Thursday, Mar 24, 2005

Name: ________________________________

ID (4-digit): ___ ___ ___ ___

Section: _____

No calculator is allowed.
To ensure fairness, everyone will be given the same amount of time, so you must stop writing when time is called. If you do not stop writing when instructed, you will receive a 25% deduction in your score.
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>12</td>
<td>15</td>
<td>20</td>
<td>27</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
1. [26 pts] Short Questions

(a) [10 pts] Determine if the following statements are true or false. For each statement, mark only the answer you think is correct.

(i) A stack is a data structure that implements first-in/last-out.

   true ______  false ______

(ii) An object of the derived class can always be constructed without calling some constructor of its base class.

   true ______  false ______

(iii) A pointer to the base class can point to objects of the derived class only if the base class has a pure virtual function.

   true ______  false ______

(iv) A class that has one or more pure virtual functions cannot be used to define objects.

   true ______  false ______

(v) A virtual function of the base class is overridden by the versions of the same function in its derived classes.

   true ______  false ______

(b) [3 pts] What does it mean when some data members of the base class are declared as protected in public inheritance?
(c) [3 pts] What is a pure virtual function?

(d) [5 pts] Convert the following infix expression into the postfix form.

\[
(a*b/4+3)*(5-b/(c+10))-(a*a/3)
\]

(e) [5 pts] Write a template function `void swap(T a[], T b[], int n);` that swaps the contents of two arrays `a[]` and `b[]` of size `n`. Here `T` can be any predefined data type.
2. [12 pts] Stacks and Queues

The class Stack implements a stack of integers. Its member functions Push(int item) and Pop() carry out push and pop operations, respectively. The member function StackEmpty() returns true if the stack is empty and false otherwise.

The class Queue implements a queue of integers. The member functions QInsert(int item) and QDelete() perform insertion and deletion for the queue, respectively. The member function QEmpty() returns true if the queue is empty and false otherwise.

You are given a stack S and a queue Q declared below:

```
Stack S;
Queue Q;
```

Their contents are shown in the following figure.

```
S:

<table>
<thead>
<tr>
<th>top \arrow{left}</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

Q:

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>front</td>
</tr>
<tr>
<td>7 9 2 4 5</td>
</tr>
<tr>
<td>rear</td>
</tr>
</tbody>
</table>
```

Draw the stack and the queue after executing the following code.

```
while (!Q.QEmpty())
    S.Push(Q.QDelete());
while (!S.StackEmpty())
    Q.QInsert(S.Pop());
```
3. [15 pts] Inheritance and Virtual Functions

Execute the following program and write the output as comments to the right of the corresponding statements inside main().

```cpp
#include <iostream.h>

class A
{
public:
    A(int k = 0) : data(k) {}
    virtual void display() const
    { cout << data << endl; }
    void display(int k) const
    { cout << data + k << endl; }

private:
    int data;
};

class B : public A
{
public:
    B(int k = 0) : A(k-1), data(k) {}
    virtual void display() const
    { cout << data << endl; }
    void display(int k) const
    { cout << data + 2 * k << endl; }

private:
    int data;
};

class C : public B
{
public:
    C(int k = 0) : B(k-1), data(k) {}
    virtual void display() const
    { cout << data << endl; }
    void display(int k) const
```
```cpp
{ cout << data + 3 * k << endl; }

private:
    int data;
};

void main()
{
    A a(1);
    B b(2);
    C c(3);

    A* pa;
    B* pb = &b;
    C* pc = &c;

    a.display();     //
    a.display(2);    //
    b.display();     //
    b.display(4);    //
    c.display();     //
    c.A::display(0); //

    pa = pb;
    pa->display();   //
    pa->display(-2); //

    pb = pc;
    pb->display();   //
    pb->display(1);  //
}
```
4. **[20 pts] Doubly-Linked List**

Use the following for the nodes in a doubly-linked list:

```cpp
class DNode {
    public:
        ...  
        DNode *DeleteNode();
        DNode *prev;
        DNode *next;
        char data;
};
```

The last node of the list has its pointer `next = NULL` and the first node has its pointer `prev = NULL`.

(a) **[8 pts]** One of the attractions of the doubly-linked list is that the deletion of a node can be carried out by letting the node detach itself from the list. Implement the member function `DeleteNode()` which deletes the current node from the list and returns a `pointer` to it. You may assume the current node has a previous node as well as a next node.
(b) [12 pts] Write a non-member helper function that takes a head pointer to a non-empty doubly-linked list and a target character, and duplicates that character every time it appears in the list. For example, if the target is ‘a’ and the list is b, a, n, a, n, a, then afterward the list would be b, a, a, n, a, n, a, a. If the target is ‘c’ and list is a, g, t, c, c, g, a, then afterward the list would be a, g, t, c, c, c, c, g, a. If the target is ‘z’ and the list is w, x, y, then afterward the list would be w, x, y.

    void dlistDoubleUp(DNode* & head, char target)
    {
    
    }
5. **27 pts** Queues

A circular queue is a queue that has one more member function — `QAdvance()` — which takes the first entry in the queue and moves it to the back. The regular queue function `QInsert(const T& item)` adds an element to the queue, where T is some predefined data type. The function `QDelete()` returns the `data` stored at the deleted node (not a pointer to the node).

A circular queue is often implemented via a circularly-linked list. Recall that in a circularly-linked list, the “last” node in the list has its `next` field pointing to the “first” node in the list, instead of being `NULL`. In a circularly-linked list being used for a circular queue, there is only one pointer needed to keep track of the entire list — the one to the tail of the list.

Use the following struct for the nodes in a circularly-linked list:

```cpp
struct CNode
{
    T data;
    CNode* next;
};
```

The list is represented by a class `CircularQueue` which has a private data member `tail`. Implement its member functions whose prototypes are given as below.

```cpp
void CircularQueue::QInsert(const T& item);
T CircularQueue::QDelete();
void CircularQueue::QAdvance();
```