Spring, 1995 Name: ______________________
My TA's name is: ______________________ My Section Day and Time : ____________

Com S 228 — Introduction to Data Structures

Test on Abstraction

This test has 4 questions and pages numbered 1 through 7.

Reminders

This test is open notes; you may also use both of the course texts, but no other books.

If you need more space, use the back of a page. Note when you do that on the front.

This test is timed. We will not grade your test if you try to take more than the time allowed. Therefore, before you begin, please take a moment to look over the entire test so that you can budget your time.

Indentation is important to us for "clarity" points; if your code is sloppy or hard to read, you will lose points. To be sure of clarity, please indent and use brackets as described in class. Also for helping functions, include PRE, MODIFIES, and POST comments (as described in class) if the function is more than a line or so long.

Correct syntax also matters. Check your code over for syntax errors. You will lose points if your code has syntax errors.

Subset of C++ You May Use

Unless otherwise stated in a problem, when solving problems you may only use standard features of the language that we discussed in class, and helping functions that you define yourself. So for headers, you may only use macros, objects, and functions declared in the following header files (see HR appendix G, DD appendix B).

```
<assert.h>   <ctype.h>     <iomanip.h>       <iostream.h>   <limits.h>
<math.h>     <stdlib.h>    <stddef.h>        <string.h>
"ask_yn.h"   "bool.h"      "clist.h"         "cqueue.h"     "cset.h"
"cstack.h"   "password.h"  "pretend_bool.h"  "prompt.h"     "rand2.h"
"swap.h"
```

Parts of C++ You May *Not* Use

Unless otherwise stated in a problem, you are prohibited from using recursion and derived classes (don't worry if you don't know what that is) and the following C++ keywords.

```
asm          catch        delete       goto         template
throw        try          virtual
```

1. (15 points) Complete the following table, by filling in for each row, the appropriate big-O measure, and the name of the complexity class (e.g., "logarithmic", "constant", etc.).

function	big-O measure	class name
$a(N) = 0.172 \cdot N^3 + 33.5 \cdot N^2 + 4 \cdot N$		
$b(N) = 75 \cdot N^2 + 1.3 \cdot N + 10^5$		
$c(N) = 1.7 \cdot 2^N + 10^7 \cdot N^4 + 30.1 \cdot N^3$		

2. (5 points) Briefly answer the following question. Why is information hiding a good idea?

3. (20 points) Implement the following specification, using the type `CharList` from the header file `"clist.h"` (which is on the next 2 pages). In the specification, the notation `<a,B,c,D>` is used for a sequence (part of the abstract value of a list), such that the sequence's front is the character 'a' its end is the character 'D', and its elements are the characters 'a', 'B', 'c', 'D' in that order.

```
// DeleteCapitals.h
#include "clist.h"
extern void DeleteCapitals(CharList & theList);
  // PRE: theList is not empty
  // MODIFIES: theList
  // POST: the cursor of theList is beyond the end of theList
  // && the elements of theList are the same as theList<entry>,
  // except that all capital letters (A-Z) in theList<entry> are removed.
  // Example: if theList<entry>'s sequence is <a,B,B,c,D>, it becomes <a,c>.
  // Example: if theList<entry>'s sequence is <C,e>, it becomes <e>.
  // Example: if theList<entry>'s sequence is <a,G,T,3,L>, it becomes <a,3>.
```

Implement the specification as a client of `CharList`. Write a *complete* implementation, including any `#include` directives you may need.

______________ *write your code for the file DeleteCapitals.C below* ______________

```
// DeleteCapitals.C
```

```
// SPECIFICATION FILE (clist.h)
#ifndef clist_h
#define clist_h 1
#include "bool.h"
const int MAX_LENG = 100;
class CharList { // ABSTRACTLY: an ordered sequence of values and a cursor,
                 // which is a position in the list or 1 beyond the tail end
    // CLASSINV: the sequence is empty --> the cursor is beyond the tail end
public:
    CharList();
        // MODIFIES: self
        // POST: sequence of values is empty
    Boolean IsEmpty() const;
        // POST: FCTVAL == (sequence is empty)
    Boolean IsFull() const;
        // POST: FCTVAL == true if no more
        // items can be stored, false otherwise
    void Reset();
        // PRE: the sequence is not empty
        // MODIFIES: self
        // POST: the cursor is at the front
    Boolean EndOfList() const;
        // POST: FCTVAL == (list cursor is beyond end of list)
    void Advance();
        // PRE: the cursor is not beyond end
        // MODIFIES: self
        // POST: cursor points to next item
    char CurrentItem() const;
        // PRE: cursor is not beyond end
        // POST: FCTVAL == item at cursor
    void InsertBefore( char someChar );
        // PRE: another item can be stored
        // MODIFIES: self
        // POST: self has someChar at cursor
        // && someChar precedes the item at cursor<entry> (if it exists)
    void InsertAfter( char someChar );
        // PRE: another item can be stored
        // MODIFIES: self
        // POST: self has someChar at cursor
        // && someChar follows the item at cursor<entry> (if it exists)
    void Delete();
        // PRE: cursor is not beyond end
        // MODIFIES: self
        // POST: item at cursor<entry> is not in the sequence
        // && cursor at next item (if it exists)
#include "clist.pri"
};
#endif
```

4. (30 points) In this problem you are to complete an implementation of `CharStack` that uses a single data member of type `CharList`. That is, you are to write the implementation code for the function members of the class `CharStack`. The specification you are to implement is given in the header file `"cstack.h"` on the next page.

   We have started the implementation for you, by giving the declaration of the private data member in the file `"cstack.pri"`, which (as usual) is included by `"cstack.h"`.

```
// cstack.pri
private:
  CharList data;
  // ABSTRACTION MAP: the abstract sequence of elements is data's sequence
```

   We have started giving the implementation by giving the code for the constructor below. You are to finish the implementation. That is, you must write, in the file `cstack.C`, the rest of the code needed to finish the implementation of `CharStack` as a client of `CharList`. You are not allowed to add to or change the private data members. You are not allowed to change the constructor. *Write in any* `#include` *directives that are needed.*

   ______________ *write your code for the file cstack.C below* ______________

```
//  IMPLEMENTATION FILE (cstack.C)


CharStack::CharStack() : data() { }  // makes data be the empty CharList
```

```
//----------------------------------------------------------------------
//  SPECIFICATION FILE (cstack.h)
//  This module exports an ADT for a stack of characters.
//----------------------------------------------------------------------
#ifndef cstack_h
#define cstack_h 1

#include "clist.h"
#include "bool.h"

class CharStack {

    // ABSTRACTLY: a sequence of component values of type char

public:
    CharStack();
        // MODIFIES: self
        // POST: the sequence is empty

    Boolean IsEmpty() const;
        // POST: FCTVAL == (sequence is empty)

    Boolean IsFull() const;
        // POST: FCTVAL == true if no more
        // items can be stored, false otherwise

    void Push(char newItem );
        // PRE: the sequence isn't full
        // MODIFIES: self
        // POST: newItem is the first in the sequence, rest is self<entry>

    char Top() const;
        // PRE: the sequence isn't empty
        // POST: FCTVAL == first item in the sequence

    void Pop();
        // PRE: the sequence isn't empty
        // MODIFIES: self
        // POST: the sequence is the tail of the sequence in self<entry>

#include "cstack.pri"
};
#endif
```