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

Com S 228 — Introduction to Data Structures

Test on Implementation Data Structures

This test has 4 questions and pages numbered 1 through 10.

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

You may use recursion on this test.

Unless otherwise stated in a problem, when solving problems you may only use standard features of the language that we discussed in class, classes and utilities 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>        <new.h>         <stddef.h>         <stdlib.h>      <string.h>
"Deallocate.h"  "IntFlexVec.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"        "vector.h"
```

Parts of C++ You May *Not* Use

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

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

1. (15 points) Draw an abstract picture, like those drawn in class with boxes and arrows, of the program at the point where the comment "`// draw the picture for this point`" appears. Please draw the program's heap storage to the right, under the word *heap* below. (Don't ask us what these pictures are like, you're supposed to know that. In any case stylistic details won't matter a great deal.)

```
struct Foo {
  char x;
  Foo *link;
};

int main()
{
  int *a;
  a = new int;
  int *b = new int;
  *a = 7;
  *b = 5;
  const int* c;
  c = a;

  Foo *head = new Foo;
  head->x = 't';
  head->link = new Foo;
  head->link->x = 'l';
  head->link->link = 0;

  // draw the picture for this point
}
```

stack *heap*

2. Consider the following declarations.

```
typedef char String50[51];
enum FoodGroup {sweet, vegetable};

struct SweetType {
  int      calories;
  int      servingSize;
  String50 flavor;
};

struct VegetableType {
  String50 mainVitamin;
  Boolean  tastesGood;
};

struct FoodInfoType {
  String50  name;
  FoodGroup tag;
  union {
   SweetType     yummy;
   VegetableType veggie;
  };
};

FoodInfoType desserts[20];
```

(a) (5 points) Write a C++ expression that is true if the element at index 3 in `desserts` is from the food group `sweet`, and is false otherwise.

(b) (10 points) Suppose that the element at index 3 in `desserts` is from the food group `sweet`. Write a C++ expression that gives the value of the 0th character of the flavor of the element at index 3 in `desserts`. (This expression should have type `char`.)

3. (30 points) Implement the following specification. (Hint: double check that your code is not "off by one", and don't try to use a stack for this.)

```
// Reverse.h
extern char * Reverse(const char *s);
  // PRE: s is a valid pointer to a null-terminated array of characters
  // && enough storage is available to be allocated for the result.
  // POST: FCTVAL is a pointer to newly allocated storage containing the
  // reverse of s.
  // That is, FCTVAL points to the first character in a
  // freshly-allocated null-terminated array of characters of the same length
  // as the length of the string s, but having the characters other than the
  // null character in reverse order.
  // Example: if s points to an array containing "aBcD12" (and a null char),
  //   then FCTVAL points to an array containing "21DcBa" (and a null char).
  // Example: if s points to an array containing "gtl" (and a null char),
  //   then FCTVAL points to an array containing "ltg" (and a null char).
  // Example: if s points to an array containing "" (and a null char),
  //   then FCTVAL points to an array containing "" (and a null char).
```

write your code for the file Reverse.C starting below

```
// Reverse.C
```

4. Consider the specification of a set of integers on the following page.

   (a) (10 points) There are two member function declarations missing from the class, which will cause problems when objects of the type are used. You are *not* to implement them, but instead write down their prototypes. *Give the declarations only.* (That is, do *not* give specifications.)

   (b) (50 points) On the following pages, implement with a dynamic singly-linked list only the constructor and the member functions `IsEmpty`, `Insert`, and `Delete` from the specification. *Do not implement the destructor or the member function* `IsElt`*!* Also do *not* implement the missing operations. (We are only having you do a subset to make the test shorter!)

       You do *not* have to write concrete pre- and postcondition specifications for these functions.

       You must use a dynamic singly-linked list in your solution. (Writing down a different kind of implementation will get you zero (0) points.)

       We suggest that you use as a data member, a pointer variable for the list head. You can either prevent duplicates in this list during an `Insert`, or you can take out any duplicates during a `Delete`. (Hint: you should give declarations in the `Set.pri` file for the nodes of the list.)

       You may assume throughout that there is sufficient memory available to satisfy all dynamic allocation requests.

       There is space on the following pages to write the `Set.pri` file, and to write the `Set.C` file.

```
// Set.h
#include "bool.h"

typedef int ItemType;

class Set {
public:
  // ABSTRACTLY: a collection of ItemType values.

Set();
  // MODIFIES: self
  // POST: the collection is empty

~Set();                              // don't write this one for the test
  // MODIFIES: self
  // POST: the memory occupied by self has been reclaimed,
  // and thus the abstract value of self is undefined.

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

Boolean IsElt( ItemType e ) const;  // don't write this one for the test
  // POST: FCTVAL == (e is in the collection)

void Insert( ItemType e );
  // MODIFIES: self
  // POST: self == self<entry> union {e}

void Delete( ItemType e );
  // MODIFIES: self
  // POST: self == self<entry> - {e}
  // (This also reclaims any memory used to store e in self.)

#include "Set.pri"
};
```

write your code for the file Set.pri starting below

```
// Set.pri
```

write your code for the file Set.C starting below

```
// Set.C
```