Spring, 1995 Name: ____________________

My TA's name is: ____________________ My Section Day and Time : ____________

Com S 228 — Introduction to Data Structures

Final Exam

This test has 5 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

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>`	`<errno.h>`	`<fstream.h>`	`<iomanip.h>`
`<iostream.h>`	`<limits.h>`	`<math.h>`	`<new.h>`	`<stddef.h>`
`<stdlib.h>`	`<string.h>`	`<strstream.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. (10 points) Briefly answer the following question. Why is a chained hash table faster on the average than one that uses linear probing?

2. In a real-time program, there are timing deadlines that must be met in *all* cases, not just on the average. If a program is too slow, and thus misses a deadline, something bad happens, like a plane crash. Briefly answer the following questions for this setting.

   (a) (5 points) Of the searching algorithms studied in class this semester, (i) which would be best for a real-time program, and (ii) why would it be best?

   (b) (5 points) Of the sorting algorithms studied in class this semester, (i) which would be best for a real-time program, and (ii) why would it be best?

3. (50 points) For use in the problem below, consider the following file, `TreeAndList.h`, which declares two types: a type of binary tree nodes, and a type of singly-linked list nodes.

```
// TreeAndList.h

#ifndef _TreeAndList_h
#define _TreeAndList_h 1

struct TreeNodeType {
  double       data;
  TreeNodeType *leftChild;
  TreeNodeType *rightChild;
};

struct ListNodeType {
  double       data;
  ListNodeType *next;
};
#endif
```

You are to implement the following function specification.

```
// ListToSearchTree.h

#include <stddef.h>
#include "TreeAndList.h"

extern TreeNodeType * ListToSearchTree( const ListNodeType * head );
  // PRE: head is either NULL or a pointer to a ListNodeType object
  // that is a singly-linked list containing no duplicates.
  // POST: (head == NULL) --> (FCTVAL == NULL)
  // && (head != NULL) --> FCTVAL is a pointer to a freshly allocated
  // binary search tree containing the data items of the list starting at head.
  // EXAMPLE: if head points to the list:
  //              <6.2, 9.5, 4.2, 5.4, 22.8, 3.7>,
  // with 6.2 the data in the node pointed to directly by head,
  // then FCTVAL points to freshly allocated storage containing the tree:
  //                                  6.2
  //                                 /   \
  //                              4.2     9.5
  //                              /  \       \
  //                            3.7   5.4     22.8
```

Hint: use one or more helping functions. (For full credit when using non-trivial helping functions, you must give their specifications. However, you can give these as comments in the code; you don't have to write separate specification files for them.)

There is space on the next page for your answer.

write your code for the file ListToSearchTree.C starting below

```
// ListToSearchTree.C
```

4. (25 points) In this problem you are to design and specify (not implement) a minimal but effectively complete type, `Dictionary`. Recall that by a "minimal" design, we mean one that doesn't have any operations that one could program using the other operations. (Don't waste time and lose points by specifying other, unneeded, operations. Don't ask us what "specify" and "effectively complete" mean, you're supposed to know that. The specifications should be of the form used in class, which you should also know.)

   The `Dictionary` objects should have no prior limit on their size, and will have abstract values that are sets of tuples, each tuple having a word and a definition (as in an English dictionary). Clients will want to sometimes pass `Dictionary` objects by value to functions. The objects of the type `Dictionary` should be mutable; that is, there should be operations that change a `Dictionary` object's abstract value.

   Be sure your design has at least one constructor, a destructor that returns any allocated storage to the C++ system, and whatever other operations are needed.

   Your design is to be recorded below by completing the specification header file we have started for you. (We have chosen to use the `String` class discussed earlier in the class instead of using C++ character arrays.)

   Do **not** *implement your design, just write the specification!* There is more space on the next page.

```
// Dictionary.h

#include "String.h"

class Dictionary {
public:
  // ABSTRACTLY: a set of tuples of the form:
  //                      (word: String, definition: String)

  // CLASSINV: IF e1 and e2 are two different tuples in the set,
  //           THEN word(e1) != word(e2).
```

5. (a) (5 points) Briefly describe in words (i.e., name) the data structure that would be most appropriate for implementing the class `Dictionary` of the previous problem. Choose a data structure that allows the client to look up definitions of words in O(1) time, on the average. It should be a data structure we studied in class. (Don't ask us what "O(1) time, on the average" means, you're supposed to know that.)

   (b) (10 points) Corresponding to your answer in the previous part of this problem, write the C++ private declarations that would go in the `Dictionary.pri` file. (Hint: you might want to think about what a constructor would do, but don't implement a constructor.)

   (c) (5 points) Write the `ABSTRACTION MAP` comment that is appropriate for your private declarations. (Don't ask us what this kind of comment is, you're supposed to know that.)