CSE 373 Homework 1 Project Description

Assigned: Wednesday, April 3, 2002
Due: Wednesday, April 10, 2002
At the start of class

Introduction

For this homework project, you will review a complete "sample homework" program, develop a program that implements simple linked lists, and answer some questions. All the source files that we are providing for this assignment are in a zip file on the web site. You should download and unzip the file. The questions were handed out in class; you can also get a copy from the web site.

Sum. The Sum program is a very simple example of the structure and content of most of the programming projects that you will do this quarter. It is included so that you can get familiar with the directory structure and the various oddball features of C that we will be using for the homework assignments.

List. The List program is the first actual development task you will have. The main program is supplied in ADT/List/mainList.c, and you are to write the individual list management functions as defined in ADT/include/list.h. The functions are based very closely on the discussion in the textbook, however, they are not exactly the same.

Grading

The 7 homework assignments of the quarter will count for a total of 50% of your class grade, and thus each individual homework assignment will count for about 7% of the total class grade. The grading for this project is as follows.

Questions: 10 points
Implementation: 10 points

Total 20 points

NOTE: You need to turn in TWO things: the paper copy of your answer sheet, and the electronic copy of your implementation of list.c.

Compiling the programs

All the programs this quarter use only the most basic input and output functions and should be able to run anywhere that you can run standard C. I have written the programs on a Linux machine running the GNU tools, and I have compiled the same sources in MS VC 6, so they should work in either environment.

Directory Structure

All of the homework projects are implementations of one of the Abstract Data Types that we discuss in class. The directory structure of the files you receive is as follows:

ADT/	top level directory
ADT/include	header files
ADT/symbols	example symbol table files
ADT/lib	object files (compiled solution ADTs)
ADT/Sum	directory for the Sum project
ADT/List	directory for the List project
ADT/<other ADTs>	directory for each ADT
ADT/Lecture	examples from the lectures

Using the Math Sciences Computing Center

You are welcome to use any machine you like to complete these assignments. The MSCC lab is in the basement of the Communications building, rooms B-22 and B-27, and has numerous PCs available for your use if you like.

One caution about using the MSCC machines. There are no individual accounts. Everyone logs on to a machine using the same account: "lab". As a result, you should never leave your homework files on any of the lab machines.

The procedure is this:

1. Log on to a PC using the "lab" account.
2. Delete everything from the C:\temp directory.
3. The first time you work on a project, download the sources from the class web site and unpack the zip file containing the supplied sources into the C:\temp directory.
4. Do whatever development you like.
5. When you are done working, copy the entire C:\temp\ADT directory to your account on Dante, using SSH Secure File Transfer.
6. DELETE your files in C:\temp and EMPTY the recycle bin.
7. The next time you want to work on your project, copy the files down from Dante, work on them, and then repeat steps 5 and 6 when you are done.
8. Remember to DELETE your files in C:\temp and EMPTY the recycle bin when you are done.

Program: Sum

This program implements the sum() function that was described in the lectures, adding up the values of symbols that were defined in a simple symbol table file.

The main point of this program is to make sure that you can download and compile the sources, and to help you understand the directory structure and code features used in this class.

To do:

1. Download the zip file for homework 1 from the web site to the machine where you will be working.
2. Unpack the file.
3. If you are using MSVC, then:

a. Create a new project. The project type is Win32 Console Application. The location is C:\ TEMP\ADT\, and the project name is Sum. MSVC should create a bunch of files in the ADT\Sum directory.
b. In the workspace window, go to FileView and click on the "+" by "Sum files". Right-click on "Source files" and select "Add files to folder…" When the dialog box opens, you should see mainSum.c and sum.c. Add them to the project.
c. From the menu bar, select Project->Settings. Make sure that the top level project name is highlighted (Sum) and then select the C/C++ tab. Select the Preprocessor dropdown item. In the entry for Additional include directories, add "..\include". This tells MSVC where to find the header files.
d. Select the Debug tab. In the entry for Program arguments, and "..\symbols\sumsymbols.txt". This tells the program what symbol file to read when it is run.
e. Click OK to dismiss the Project Settings dialog.
f. From the menu bar, select Build->Rebuild All. If all is okay, you should see messages that say that the two source files have been compiled and Sum.exe has been linked.
g. Click the Execute Program button (or Ctrl-F5) and run the program.

4. If you are using Unix, then you should be able to go to the Sum directory and run the Makefile that is there to create a new executable. Note that this is a GNU make file, so you may need to specify

"gmake", depending on how your system is set up. This will create a program file called summer, which you can execute using "./summer ../symbols/sumsymbols.txt"

5. The output of the program is
   main: read 5 symbols.
   00000001 one
   00000002 two
   00000003 three
   00000004 four
   00000005 five
   main: sum is 15.
6. You are in business! Review the code as needed in order to answer the questions in the homework.

Program: List

This program requires you to implement a simple set of list management functions. The main program and the header files are supplied; you add code to the implementation file "list.c" to supply the actual list management functions.

To do:

1. Create the List project just as you did above for the Sum project. For the Program arguments entry, use "..\symbols\tvsymbols.txt" or any one of the other symbol table file names.
2. Note that although there is a file "list.c" it is very incomplete, and so the project will not link correctly. All the routines that you will implement are missing.
3. Write the routines that are needed by mainList as described below, rebuild the project, and run it. Debug until done. You can use #if 0 and #endif to chop out parts of mainList until you get them written if you like.
4. Review the code as needed in order to answer the questions in the homework.

The function headers for the routines you need to write are in ADT/include/list.h. Note that the functions are very similar, but NOT IDENTICAL, to the ones defined in the textbook. These functions implement a header node implementation of lists. The functions are as follows.

List CreateList(void);

Similar to MakeEmpty, but does not take an argument. It just creates a new list header node, fills it in correctly, and returns a pointer to the header.

void ClearList(List L);

Similar to DeleteList. This function steps through the List and frees each of the ListNode objects that the list contains. It does not release the header node. After this function executes, the header node still exists, and is pointed to by L. L->next is NULL.

void PrintList(List L, ElementPrintLabel P);

This function steps through the list from the first element to the last, and calls the ElementPrintLabel function P for each element. PrintList adds a printf("\n") after each element, so that one line is printed per symbol. Review the Sum function for an example of how to call a function that has been supplied as a pointer.

int CountListEntries(List L);

This function steps through the list from the first element to the last and counts how many elements there are.

Position InsertListEntry(List L, Position P, ElementType X);

This function creates a new ListNode structure, fills in the Element field correctly, and links the new node into the existing list after the node pointed to by P.

int ListIsEmpty(List L);

True (1) if there are no entries in the list. The header node does not count as an entry.

Position GetFirstListEntry(List L);

Returns a Position value pointing to the first entry in the list, NULL if the list is empty.

Position AdvanceListEntry(Position P);

Returns the Position value for the entry after the one pointed to by P.

ElementType GetListEntryElement(Position P);

Returns the Element value of the node at Position P.

int ListEntryIsLast(List L,Position P);

True (1) if Position P is the last entry in the List L.

Sample output from the finished program is:

```
[finson@walnut List]$ ./lister ../symbols/zerosymbols.txt
Printing the entire list.
List contains 0 nodes (0).
List is empty.
Clearing list ... cleared.

[finson@walnut List]$ ./lister ../symbols/onesymbol.txt
Printing the entire list.
00000001 onesymbol
List contains 1 nodes (1).
First entry is 00000001 onesymbol
Last entry is 00000001 onesymbol
Clearing list ... cleared.

[finson@walnut List]$ ./lister ../symbols/twosymbols.txt
Printing the entire list.
00000001 symOne
00000002 symTwo
List contains 2 nodes (2).
First entry is 00000001 symOne
Last entry is 00000002 symTwo
Clearing list ... cleared.

[finson@walnut List]$ ./lister ../symbols/tvsymbols.txt
Printing the entire list.
08048620 _start
08048720 main
```

```
08048c44 nodeCompareByName
08048c68 nodeCompareByValue
08048ca0 printDigraphHeader
08048cf0 printDigraphFooter
08048d24 printNodeLabel
08048d50 printNodeLinks
08048d90 MakeEmpty
08048dd4 FindTreeNode
08048e60 FindTreeNodeMin
08048e98 FindTreeNodeMax
08048ed4 InsertTreeNode
08049120 DeleteTreeNode
08049254 RetrieveTreeNodeElement
08049260 TraverseTree
List contains 16 nodes (16).
First entry is 08048620 _start
Last entry is 08049260 TraverseTree
Clearing list ... cleared.
```