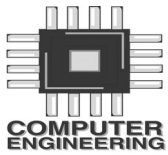
Department of Electrical & Computer Engineering
University of Maryland at College Park

ENEE 114: Programming Concepts for Engineers
Fall 2003, Lecturer: Aamer Jaleel



Project #3: Link Lists & File I/O

Due by December 14th, 2003 at 11:59pm.

1 Overview:

This project will provide you with practice using dynamic data structures. The project requires you to read a file that contains data, sort the data the file, and then write the data to a separate file. Through this project, you will learn the concepts of structures, pointers to structures, and dynamic memory allocation.

2 Project Description:

2.1 Part 0 - Structure Definitions and the Header File

You are going to be maintaining a database of all students attending some university. Since the university has many courses, we will be maintaing a structure called: course. It has the following member elements

- Course Name - 5 char long
- Course ID - integer
- Course Section - integer

For each student in the university, the university keeps track of the following data structure called "student" with the following member elements:

- First Name - 100 char long
- Last Name - 100 char long
- SSN - 15 char long
- Number of Courses Attending - integer
- Array of Courses - a 10 elmement array of *course* elements
- next - Pointer to the next student structure in the list

Create a file called "students.h" that has within it the definition of these two structures listed above. You will be adding to this header file function prototypes that we will discuss later.

2.2 Part 1 - Creating and Adding to a Link List

You are provided with a data file called students.txt that has the data for a number of students. You are required to read this data into a linked list data structure. The sample input file will end with a line that has five stars. An example of an input file with two students is:

```
----- student.txt----
Lisa
Sanders
111-11-1111
```

```
3
ENEE 114 0103
CMSC 412 0101
ENME 515 0123
Bart
Simpson
222-22-2222
1
CMSC 412 0101
*****
----- end input.txt -----
```

For each student, the data file is formatted line by line in the following manner:

```
<first name>
<last name>
<SSN>
<number of courses they are taking>
<course name> <course id> <course section>
```

You are required to read this data from the input file and maintain each students record in a sorted linked list. The linked list must be sorted based on the last name of the student. Once you have read the entire data from the input file, you are required to print the sorted data to the output file called *"output.txt"*. A sample output file is provided to you at the end of this project description.

2.2.1 Reading Data From the Input File

```
// Opens up the file for reading
// reads in the data from the file on a student by student basis
// Once the data for a student has been read
// add the student to the link list by calling AddToList
void ReadInputFile( char *filename );
```

2.2.2 Adding a Student to the Linked List

```
// Adds The argument student to the LinkList
void AddToList( struct student *addStudent );
```

2.2.3 Printing the Linked List To Output File

```
// Opens up the file for writing
// Goes through the linked list and prints the data to the output file
void PrintToFile( char *filename );
```

2.3 Part 2 - Deleting From Link List

You are provided with a file "delete.txt" that contains a list of students who have dropped out for the semester, so you need to remove them from the linked list. Read the file "delete.txt" that has a list of SSN and remove each student with that SSN from the linked list. Once you have deleted everybody from the linked list, print the updated list to the output file called "update.txt"

```
----- delete.txt -----
111-11-1111
*****
------ end delete.txt -----
```

2.3.1 Reading Data From the Delete File

```
// Opens up the file for reading
// reads each students SSN from the file
// Deletes the student with that SSN from the list by calling DeleteStudent
void ProcessDeleteFile( char *filename );
```

2.3.2 Deleting a Student From The Linked List

```
// Looks through the linked list for a student matching the SSN passed in
// if it finds it, deletes it from the linked list
void DeleteStudent( char *SSN );
```

2.3.3 Delete Entire Link List

```
// Deletes the Entire Linked List
void DeleteList();
```

3 File Submissions

You are going to **submit the files student.h and student.c**. The header file student.h has within it the structure definitions and the function prototypes. The file student.c has within it the implementation of the functions and also has the main.

4 Skeleton Code

4.1 Header File - student.h

```
/* Put Course Structure Definition Here */


/* Put Student Structure Definition Here */

void ReadInputFile( char *filename );
void AddToList( struct student *addStudent );
void PrintToFile( char *filename );
void ProcessDeleteFile( char *filename );
void DeleteStudent( char *SSN );
void DeleteList();
```

4.2 Source File - student.c

```
#include <stdio.h>
```

```
#include <string.h>
#include "student.h"

// Global Linked List initially set to zero
struct student *head = NULL;

int main ( void ) {

  // Read the input file and create the linked list
  // ReadInputFile calls helper function AddToList to
  // create list in sorted order
  ReadInputFile( "student.txt" );

  // print the current list to the outputfile output.txt
  PrintToFile( "output.txt" );

  // Read and process the dropped students file delete.txt
  // It uses helper function DeleteStudent to remove
  // the from the list

  ProcessDeleteFile( "delete.txt" );

  // Print the updated linked list to the file update.txt
  PrintToFile( "update.txt" );

  // Delete the Entire Linked List before exiting the program
  DeleteList();

  return 0;

}

void ReadInputFile( char *filename ) {

  struct student tmp; // temporary student structure

  // open up the input file for read mode
  // if it can't open print error message

  // while you still have data to read
  // read one students data into temporary structure tmp
  // after you have gotten all the data for one student
  // call the function AddToList

}
void ProcessDeleteFile( char *filename ) {
  // create string deleteSSN

  // open up the file for read mode, print error message if not present

  // while you still have data to read
  // read a string from the file
  // call the function DeleteStudent with argument SSN that you read in
```

```
}

void DeleteList( void ) {

  // start from the head and remove each entry
  // from the list and free up the space via calling the function free

}

void PrintToFile( char *filename ) {

  // open up file for write mode

  // print the linked list student by student to the
  // output file. Sample output in section 5.1 & 5.2

}

void AddToList( struct student *addStudent ) {

  // malloc a new student
  // copy contents of argument student to the new student

  // is linked list empty? if so, new student is head

  // if not, search through the linked list to find appropriate
  // spot within the linked list

  // Concerns:
  // - what if new student ends up being the first? change head
  // - what if new student ends up in the middle? update pointers
  // - what if new studend is last?

}

void DeleteStudent( char *SSN ) {

  // search through the linked list to see if SSN exists
  // If so, remove it from the linked list
  // Concerns:
  // - what if deleted student is the head? update head
  // - what if deleted student is in the middle? update pointers
  // - what if deleted student is at the tail?

}
```

5 Sample Output Files

5.1 Sample output.txt

```
SSN: 111-11-1111
```

```
Name: Sanders, Lisa
Registered For 3 Classes
  - ENEE0114-0103
  - CMSC0412-0101
  - ENME0515-0123

SSN: 222-22-2222
Name: Simpson, Bart
Registered For 1 Classes
  - CMSC0412-0101
```

5.2 Sample update.txt

```
SSN: 222-22-2222
Name: Simpson, Bart
Registered For 1 Classes
  - CMSC0412-0101
```

5.3 Sample Run

```
Adding To List: Lisa, Sanders
Adding To List: Bart, Simpson

Deleted Student Entry: 111-11-1111
```