

COMP 2510

Assignment 4

1 Introduction

The purpose of this assignment is to use a linked list to sort & search for records. Each record consists of a name & a score. The program reads in records from standard input, inserts them into a linked list in a specific order & then either prints out all the sorted records or only those with specific names.

2 Data Structures

For this assignment, we'll be using a linked list of structures. Each structure stores information about a student:

```
#define NAMESIZE 20
typedef struct name  name;

struct name {
    char  last[NAMESIZE];    /* last name */
    char  first[NAMESIZE];   /* first name */
};

typedef struct record record;

struct record {
    name  name;
    int   score;
};
```

accessing something like last name would be something like node -> data.name.last.

Each node of the linked list is given by:

```
typedef struct node  node;

struct node {
    record  data;
    node    *next;
};
```

Note that a node contains a record & a pointer to the next node; a record contains a name & a score. A name in turn consists of a last name & a first name.

The linked list is ordered in increasing order of last names. If two or more nodes contain records with the same last name, they are ordered in increasing order of their first names. Finally, if several nodes have the same last & first names, they are ordered in increasing order of their scores. **Note that the program uses case-insensitive comparison when comparing strings.** (Note also that the standard ANSI C library does not have a function that performs case-insensitive string comparison.)

3 The Program

The program basically reads records from standard input & inserts them into the linked list in the order specified above. This reading & inserting of records continues until end-of-file is encountered when reading from standard input. If the program is invoked without arguments, it then proceeds to print out all the records (to standard output) in sorted order.

However, the program supports 2 options that specify the last & first names of the records to print:

`-fNAME` look for records whose first name is `NAME`
`-lNAME` look for records whose last name is `NAME`

If one of the 2 options is specified, only records with the specified first or last name (depending on whether it is `-f` or `-l`) are printed. If both are specified, printed records match both the specified first & last names. Note that the records are printed in sorted order (to standard output) if there is more than one match. If there are no matches, the message “No matches” should be printed (again to standard output).

Note that each of the 2 options can only be specified once. It is an error to repeat the same option. (A usage message should be printed to standard error in such a case.) The following are valid invocations of the program which we'll call `lsort`:

```
lsort
lsort -lsimpson
lsort -fhomer
lsort -fhomer -lsimpson
lsort -lsimpson -fhomer
```

unsure as to whether to do a 2 function structure, or a single, really large one. most likely the 2 function structure, similar to ans03.

The following invocations are invalid:

```
lsort -a
lsort -fhomer -fnd
lsort -f homer
```

sscanf naturally does the valid line checking.

We now look at the input format. Input consists of lines each specifying a record. A valid line consists of the last name, the first name, the score & possibly comments, all separated by spaces & in that order. This means that there must be at least 3 words in a valid line. Furthermore, the score must be an integer, the last & first names must each consist of one word whose length is at most `NAMESIZE-1`. (However, there is no restrictions on the characters in the word.) The program skips any line that's not valid. You may assume that each line has fewer than 256 characters.

The following are examples of valid lines:

`BUFSIZE = 256; NAMESIZE = 20;`

```
SIMPSON12 homer 25 # names can contain any character
flanders ned 099 # leading 0's are OK for score
Burns Montgomery 89
Simpson Bart 35
Simpson Lisa 90
simpson bart 25
```

Check to see if leading zeros are taken out when converting to int (most likely true).

And here are some invalid ones:

```
simpson marge # invalid score (# is not a valid score)
simpson bart 34.5 # 34.5 is not an integer
```

Checking for floating point numbers might be difficult, maybe do `sscanf(%, %s, %d, %f)?`

The sorted records are printed to standard output. For the 6 valid lines above, the output is:

```
burns montgomery 89
flanders ned 99
simpson bart 25
simpson bart 35
simpson lisa 90
simpson12 homer 25
```

Note that

- there are no leading zeros in the score

before storing in the list, do a tolower, and then use the standard idiom for sorting.

- the last & first names are in all lowercase
- case-insensitive string comparison is used when determining sorting order (this makes sense since names are printed in all lowercase)
- record members are separated by single spaces

Your program will basically be tested using I/O redirection & comparing your output with a reference output using a file comparison program. Your program will be deemed incorrect if its redirected output does not match exactly the reference output. Some sample input & output files will be provided.

The program must explicitly destroy the linked list before exiting.

This should've been bolded, destroy the list before exiting, similar to unlinking the file in asn01.

4 Submission & Grading

This assignment is due at noon, Wednesday, April 9, 2008. You'll need to submit through subversion as with previous assignments.

We'll be providing one or more makefiles. Your program must compile without errors or warnings with the provided makefiles. Unless this requirement is met, we'll not look into any issue you may have with the marking of your assignment. If your program actually fails to compile with the gcc makefile, you may receive a score of 0 for the assignment. Otherwise, the grade breakdown for this assignment is approximately as follows:

Coding style and code clarity	10%
Validation (input & command-line)	15%
Sorting order	30%
Searching records (-f & -l options)	25%
Output format	10%
Destroying linked list	10%