Introduction to C - Programming Assignment #5

Due date: November 18, 2011, 11:59pm

Objectives

1. To learn how to store strings and use string functions in string.h
2. To practice sorting a set of items with a tiered comparison system.

Problem: Guest List (guest.c)

It's come time to decide who exactly is going to come to Arup's wedding. Unfortunately, he and his fiancée know more people than their reception hall can accommodate. Thus, they have to choose only some of their friends and family to invite. In this program, you will help them with this task.

Each invitation goes to a family. We typically address each invitation to a single member of a family. In addition, each family is assigned a priority number. A lower priority number indicates that the family is more important. (Namely, we first invite everyone with priority 1 to the wedding, followed by everyone with priority 2, etc.) Thus, the information for each family receiving an invitation is:

1) First name (A string of 1-19 upper case letters)
2) Last name (A string of 1-19 upper case letters)
3) Number of people in the family (A positive integer in between 1-10, inclusive)
4) Priority number (A positive integer in between 1-20, inclusive)

It's taboo to only invite some families from a particular priority number because then a distinction has been made between people of equal importance and that can cause of social uneasiness. It's also clear that we want to invite guests in order of priority. Thus, to put together the complete guest list, we must do the following:

1) Sort the guest list by priority, break ties by last name, and then breaking those ties by first name.

2) Figure out which priority levels, starting at 1, will get invited. (Namely, if the total number of guests with priorities 1 through k does NOT exceed the capacity while the total number of guests with priorities 1 through k+1 does exceed the capacity, then the priority levels that get invited are 1 through k.)

3) Output the final guest list to a file in sorted order by last name, breaking ties by first name. (Note that we don't want the final guest list to indicate the priority numbers, as this would probably hurt people's feelings. These priority numbers are only meant to be used internally to determine who to invite.)

Implementation Requirements

Do NOT use structs for your solution. You must use multiple arrays to store all of the data separately. Admittedly, this is not a desired method of solution. In program 6, you will rewrite this program to work with structs. The goal of the two programs together is to show you the advantages of structs.

You are required to design your own functions. Any program written with only one function (main) will automatically receive a 20 point deduction.

It is suggested that you have functions that do the following tasks:

1) Read information from a file into data structures
2) Swap elements
3) Sort arrays
4) Print information to a file

It's likely that you'll find other functions to write that aren't specified here.

Input Specification (allguests.txt)

The input will be from the file allguests.txt.

The first line of the file contains a two positive integers, *n (n ≤ 1000)*, representing the number of families that Arup and his fiancée know, and *c (c ≤ 1000)*, representing the capacity of their reception hall, in number of people.

The next *n* lines each contain information about one family.

Each of these lines has the following information about one family separated by spaces: F, L, N ($1 \leq N \leq 10$) and P ($1 \leq P \leq 20$). F represents the first name, L represents the last name, N the number of people in the family and P represents the priority number of the family. You are guaranteed that no people listed will have the exact same first and last name. (Two people might have the same first name and a different two people might have the same last name, but no two people will share both.)

Output Specification (finalguestlist.txt)

The first line of the output file should have to positive integers, *k*, the number of families invited to the wedding, and *g*, the actual number of guests, separated by spaces.

The next k lines should have one family listed each. These lines should be sorted by last name, breaking ties by sorting by first name. The output for each of these lines should be as follows:

```
LAST, FIRST #PEOPLE
```

Sample Input File (allguests.txt)

```
10 30
BEN JOHNSON 4 2
DOUG ESPINOSA 3 2
SARAH TELLINGER 5 3
GRANT THOMPSON 5 2
JENNIFER WEST 7 6
JACKSON JOHNSON 1 5
MARTY MCFLY 4 1
ELIZABETH JAMES 2 6
MICKEY MOUSE 2 4
RAJ SHAH 2 5
```

Sample Output File (finalguestlist.txt)

```
8 26
ESPINOSA, DOUG 3
JOHNSON, BEN 4
JOHNSON, JACKSON 1
MOUSE, MICKEY 2
MCFLY, MARTY 4
SHAH, RAJ 2
TELLINGER, SARAH 5
THOMPSON, GRANT 5
```

Deliverables

The source file, *guest.c*, containing your solution to the problem.

Restrictions

Although you may use other compilers, your program must compile in gcc and run in the Code::Blocks environment. Your program should include a header comment with the following information: your name, course number, section number, assignment title, and date. Also, make sure you include comments throughout your code describing the major steps in solving the problem.

Grading Details

Your programs will be graded upon the following criteria:

1) Your correctness

2) Your programming style and use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.

3) Compatibility to gcc in Code::Blocks. If your program does not compile in this environment, you will get a **sizable** deduction from your grade.

Extra Credit (40pts)

Write a program that reads a message, then checks whether it's a palindrome (the letters in the message are the same from left to right and from right to left:

Example input/output

```
Enter a message:  He lived as a devil, eh?
Palindrome

Enter a message: Madam, I am Adam
Not a palindrome
```

Ignore all characters that aren't letters.

Implementation 1: Use integer variables to keep track of the positions in the array. This is worth 20 pts.

Implementation 2: Use pointers to keep track of the positions in the array. This is worth 20 pts.

Here are some example palindromes to test against.

- A man, a plan, a canal--Panama!
- Able was I ere I saw Elba.
- Too bad--I hid a boot.
- Do geese see God?
- Murder for a jar of red rum.
- Drab as a fool, aloof as a bard.

Deliverables

The source file, *palindrome1.c*, containing your solution to the problem using integers and *palindrome2.c,* containing your solution the problem using pointers.