CS2124 Data Structures
Project 1 - Priority Queue

Reminder: This project is like an exam. The program you submit should be the work of only you and, optionally, one other partner. You are not allowed to read, copy, or rewrite the solutions written by anyone other than your partner (including solutions from previous terms or from other sources such as the internet). Copying another person's code, writing code for someone else, or allowing another to copy your code are cheating, and can result in a grade of zero for all parties. If you are in doubt whether an activity is permitted collaboration or cheating, ask the instructor.

Every instance of cheating will be reported to UTSA's Student Conduct and Community Standards office (SCCS). At minimum, this will result in a zero for the project/exam and **at most a grade of a C for the course**.

1 Overview

The program you're writing will facilitate distributing a budget among multiple departments. Each department has a list of items they wish to purchase and you'll simulate a simple algorithm to determine which items are purchased.

2 Input File Format

Reminder you can open a file to read from by doing the following:

```
/* Open the fileName */
FILE *f = fopen( fileName, "r" );

if( f==NULL )
{
    printf("ERROR - Invalid file %s\n", fileName);
    exit(-1);
}
```

The input file will list the department followed by the names/prices of the items this department desires. Here is the generic format the input files will follow:

```
<department-name-string>

<item1-name-string>
<item1-price-double>

<item2-name-string>
<item2-price-double>
```

```
....

<itemlast-name-string>
<itemlast-price-double>
```

Each string will contain no space chars and will have a length of at most 30 characters (not including the NULL terminator). The prices will be doubles. Here's a simple example input file:

```
Mathematics

Graph-Paper
150

Coffee
300
```

3 Algorithm for Distributing the Budget

- Each department is given an initial priority of 0. This priority will represent the total amount of money spent on that department so far. Each time an item is purchased you'll update that priority accordingly.
- Add to each department's list of *itemsDesired* the items they have requested in their respective files.
- Add each department to a priority queue.
- While the remaining budget is > 0 do the following:
  - Remove and save the front department of the priority queue.
  - While the price of the next item desired by this department is > the remaining budget, move that item to the department's *itemsRemoved* list.
  - if there are no desired items in this department's *itemsDesired* list, then they are given a scholarship equal to $1000 or the remaining the budget (whichever is smaller).
  - Otherwise, the next desired item is purchased for this department.
  - In either case, remember to:
    - Update the priority of the department
    - Add the item to their list of *itemsReceived* list
    - Add this department back to the priority queue
    - Deduct the amount spent from the remaining budget

4 Output

Your output will start with an list of the items in the order which they were purchased (including their prices and which department they were purchased for).

```
ITEMS PURCHASED
---------------------------
<department-name> - <item1-name>    - <item1-price>
<department-name> - <item2-name>    - <item2-price>
<department-name> - <item3-name>    - <item3-price>
...
<department-name> - <itemlast-name> - <itemlast-price>
```

After, the above you should list of the departments with the items they received and the ones they did not receive (be sure to also **print remaining items in the** *itemsDesired* **list**). You'll also list the total amount of money spent on the specific department and what percentage of the total budget that is.

```
<department1-name>
Total Spent       = $<department1-priority>
Percent of Budget = <department1-priority>/<total-budget>%
---------------------------
ITEMS RECEIVED
<item1-name>    - <item1-price>
<item2-name>    - <item2-price>
....
<itemlast-name> - <itemlast-price>

ITEMS NOT RECEIVED
<item1-name>    - <item1-price>
<item2-name>    - <item2-price>
....
<itemlast-name> - <itemlast-price>


<department2-name>
Total Spent       = $<department2-priority>
Percent of Budget = <department2-priority>/<total-budget>%
---------------------------
ITEMS RECEIVED
<item1-name>    - <item1-price>
<item2-name>    - <item2-price>
....
<itemlast-name> - <itemlast-price>

ITEMS NOT RECEIVED
<item1-name>    - <item1-price>
<item2-name>    - <item2-price>
....
<itemlast-name> - <itemlast-price>


...
```

```
<departmentlast-name>
Total Spent       = $<departmentlast--priority>
Percent of Budget = <departmentlast--priority>/<total-budget>%
---------------------------
ITEMS RECEIVED
<item1-name>    - <item1-price>
<item2-name>    - <item2-price>
....
<itemlast-name> - <itemlast-price>

ITEMS NOT RECEIVED
<item1-name>    - <item1-price>
<item2-name>    - <item2-price>
....
<itemlast-name> - <itemlast-price>
```

Your solution's output should match the provided sample output. To better match the sample output, use the following partial code segments to print your prices:

```
//Printing prices as items are purchased:
sprintf( priceString, "$%.2lf", /* your price here */ );
printf( "Department of %-30s - %-30s - %20s\n", /* department */, /* item */, priceString );

//Printing list of items:
sprintf( priceString, "$%.2lf", /* your price here */ );
printf( "%-30s - %20s\n", /* item */, priceString );
```

5 Deliverables:

Upload your *resourceManagement.c* file to Canvas under Project 1. If you created any other *.c* or *.h* files for your solution be sure to submit those as well as an updated makefile **Do not zip your files**. Be sure to double check the materials you submitted are the correct versions.

To receive full credit, your code must compile and execute. You should use valgrind to ensure that you do not have any memory leaks.

Continued on next page ↪

6 Grading Notes:

The project graded out of 20 points. Here's a rough breakdown of points awarded (each higher grade assumes all prior criteria are met):

- 10/20 - Correctly reading and storing all of the values in the input file
- 12/20 - Algorithm from Section 3 is at least partially developed
- 14/20 - The budget is fully distributed to the departments
- 16/20 - The output format is correct
- 18/20 - The output is correct except for minor errors
- 20/20 - Correctly printing output

Additional deductions applied to the above scores:

- Code that does not compile on the CS Linux lab machines will receive a deduction of **at least** $2pts$.
  - Code that does not compile with non-trivial errors usually receive very few points (e.g. $< 5pts$).
  - This because I cannot test your code.
  - Be sure to check your code!
- Not mallocing your data will receive a deduction of **at least** $2pts$.
- Memory leaks will cause a reduction of grade by $1pt$ to $2pts$.
- Memory errors will cause a reduction of grade by $1pt$ to $2pts$.
  - Be sure to look for these when you run Valgrind!
  - Examples: invalid reads, invalid writes, Conditional jump based on uninitiallized value, etc.

If your worked with a partner, remember that both you and your partner should submit your solution (even if they are completely the same).