CSCI 2100B Data Structures

Assignment 1
Due Date: ~~11~~25 February 2020

Written Exercises

There is no written exercise.

Programming Exercises

1. A well-formed string of parentheses is a character string that contains only
   - the left and right round parentheses ('(' and ')'),
   - the left and right braces ('{' and '}'), and
   - the left and right square brackets ('[' and ']'),

   and for each of the left parenthesis, there is a corresponding right parenthesis <u>at the correct position</u>. The following are examples of well-formed strings of parentheses:

   ```
   ()
   ()[()]
   [{{()[()]}}()]
   ```

   The following are not well-formed string of parentheses:

   ```
   (
   ()[()
   ()]({)
   ```

   Write a program (**brackets.c**) that makes use of a stack to check whether a string of characters is a well-formed string of parentheses.

   - You should read the input string from **stdin**, and use the following string as the input prompt:
     Please input the string of parenthesis:
   - There is no space or any other characters in the string of parentheses.
   - You <u>should</u> use the file **stack.h** header file defined in the lecture notes.
   - You <u>should</u> use the stack implementation (versions 1.0, 2.0, or later) defined in the lecture notes — do remember to modify the type of stack elements from **int** to **char**.
   - The output of the program should be either
     Yes, the input is a well-formed string of parentheses.
     or
     No, the input is NOT a well-formed string of parentheses.

2. Write a program to simulate the following scenario. You <u>should</u> use the **queue.h** header file defined in the lecture notes.

   There is a supermarket that has only 4 cashiers, numbered 0, 1, 2 and 3. Therefore, in general, customers form four queues, waiting to check out in front of each of these three cashiers. For simplicity, we assume that at most one customer comes to check out every minute. When a customer comes, he will first check which of these three queues is the shortest, and will then join the shortest queue. If a customer finds that there are two queues of the same length, then he will prefer cashier 0 to cashier 1, cashier 1 to cashier 2, and cashier 2 to cashier 3, because cashier 0 is the closest to the exit. A customer starts to check out as soon as the customer before him in the queue finishes checking out.

The input to the program is read from a file. The first line in the file specifies the information of the customer who comes in the first minute; the second line specifies the information of the customer who comes in the second minute, and so on. An empty line indicates that no customer comes in that minute. Each line contains two numbers. The first is an integer that indicates the number of minutes the customer needs to wait while being served at a cashier. The second is a floating point number that indicates the total price of the goods he buys. There are one or more spaces between these two numbers. For example, if the file contains only the following five lines:

```
3  10.9
1    2.1
10 100.5

10 200
```

Then it means that there are totally 4 customers: the first comes in the first minute, the second in the second minute, *etc*., and no customer comes in the fourth minute. The first customer will need to spend 3 minutes checking out, and he will need to pay $10.9; the second customer will need to spend 1 minute checking out, and he will need to pay $2.1, and so on. No more customers will come after the fourth customer (who comes in the fifth minute) is served. You may also assume that there is no error in the input data, so no error checking is necessary.

Write a program (**supermarket.c**) that reads an input file (file name is "supermarket.dat"). The total number of lines in the input file is not known in advance. The program performs the simulation, and outputs the following information

- The average time (over all four counters) a customer needs to wait after he joins a queue until he starts to check out.
- The total amount of money each of the cashiers receives.
- The total number of customers each of the cashiers serves.

You should represent a customer as a structure defined as

```
struct customer {
    int checkoutTime;
    float payment;
};
```

We also define a type **customerT** for customers, as follows:

```
typedef struct customer *customerT;
```

The values of the fields should be properly initialized when a Customer structure is created for a customer. The type of queue elements in the **queue.h** header file and the queue implementation file **queue.c** should therefore be modified from **int** to **customerT**.

A cashier is an object of the following class:

```
struct cashier {
    int numberOfCustomersServed; /* This should be initialized to 0 */
    int totalCustomerWaitingTime; /* This should be initialized to 0 */
    float totalAmountReceived; /* This should be initialized to 0 */
    queueADT customerQ; /* This should be initialized to an empty queue */;
```

```
        /* This is a Queue of customer structures*/
}
```

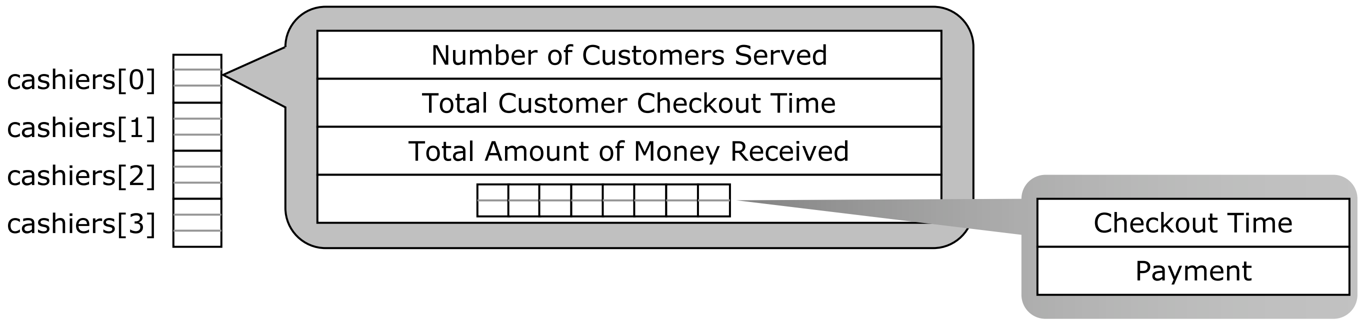
We also define a type **cashierT** for cashiers, as follows:

typedef struct cashier cashierT;

Finally, the three cashiers should be implemented as an array in **main**:

cashierT[4] cashiers;

The following diagram shows the picture as seen by you (who are supposed not to know the contents of **queue.c**).



cashiers[0]
cashiers[1]
cashiers[2]
cashiers[3]
Number of Customers Served
Total Customer Checkout Time
Total Amount of Money Received
Checkout Time
Payment