CSCI2100B Data Structures

Assignment 1
Due Date: 6 February 2026

Written Exercises

1. Sujuan uses 13 buckets (bucket 0, bucket 1, …, bucket 12) in the implementation of symbol table ADT, and to use quadratic probing to resolve collision.

   a) The key of the first entry is "Morningside College", and the hash code of the key is 7. At which bucket will the first entry be stored?
   b) The key of the second entry is "S. H. Ho College", and the hash code of the key is 7. At which bucket will the second entry be stored?
   c) The key of the third entry is "C. W. Chu College", and the hash code of the key is 8. At which bucket will the third entry be stored?
   d) The key of the fourth entry is "Wu Yee Sun College", and the hash code of the key is 7. At which bucket will the fourth entry be stored?
   e) The key of the fifth entry is "Lee Woo Sing College", and the hash code of the key is 8. At which bucket will the fifth entry be stored?

2. Repeat Question 1, assuming linear probing is used to resolve collision.

Programming Exercises

3. Write a program that reads a line from **stdin**. The program prints 'yes' to **stdout** if the line contains a number of lowercase characters in the English alphabet, followed by the same number of uppercase characters in the English alphabet. That is, the last character is the uppercase of the first lowercase character; the second last character is the uppercase of the second lowercase character; and so on. The program prints 'no' to **stdout** otherwise.

   - You may assume that the maximum length of the input character string is 80.
   - You should use the file stack.h defined in the lecture notes.
   - You may use the file stack.c (any version) defined in the lecture notes — do remember to modify the type stackElementT from int to char.
   - The program should keep reading in new lines and determining whether it contains a palindrome, until the line contains the word '*quit*'.
   - You do **not** need to do any error-checking, *e.g.*, whether the user types any special characters, blanks, punctuation marks, *etc.*, or empty lines. In all these cases, the program simply prints 'no'.

The following diagram shows a scenario when the program is run.

```
a
no

aA
yes

abcCBA
yes

bccbb
no

abcCBAE
no

Abba
no

quit
```

4. Write a program to simulate the following scenario using the queue ADT.

   In a bank there are five teller counters. Counters 0 and 1 are reserved for customers holding VIP banking accounts, while counters 2, 3 and 4 can be used by customers holding VIP banking accounts and customers holding ordinary banking accounts. Customers waiting to be served form five queues in front of the five counters, as depicted in Figure 1.

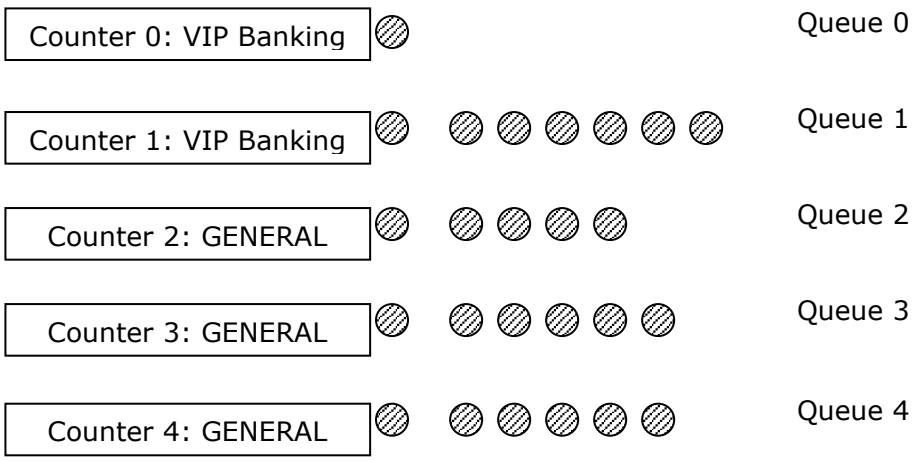


Figure 1: A bank with two VIP Banking counters and three general counters

For simplicity, we assume that at most one customer comes to the bank every minute. When a customer comes, he always joins the shortest queue. Of course, a customer holding ordinary banking account can only choose from queues 2, 3 and 4, while a customer holding a VIP banking account can choose from queues 0, 1, 2, 3 or 4. We assume that if there are two queues of the same length, a customer always prefers

counter 0 to counter 1, counter 1 to counter 2, counter 2 to counter 3, and counter 3 to counter 4.

For simplicity, we assume that if a customer decides to join a queue, then he must be served at that counter, and he cannot leave the queue and rejoin any other queue, even if other counters become idle. The same applies to customers joining queues 0, 1, 2, 3 and 4.

The input to the program is read from a text file (**bank.dat**). The length of each line in the file is at most 80 characters. 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. A blank line indicates that no customer comes in that minute. Otherwise, each line begins with either the character 'V' or the character 'O', indicating wither the customer is a VIP banking account holder or an ordinary banking account holder. A line then contains an integer that indicates how many minutes the customer needs to be served. For example, if the file contains only the following five lines:

```
V 3

O 5
O 1
V 5
```

then it means that there are totally 4 customers: the first comes in the first minute, the second in the third minute, *etc*., and no customer comes in the second minute. The first customer is a VIP banking account holder who needs to be served at a counter for three minutes, the second customer is an ordinary banking account holder who needs to be served at a counter for five minutes, and so on. There are no more customers coming after the fourth customer.

Write a program that reads an input file (file name is always "bankcustomers.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 a VIP account holder needs to wait after he joins a queue until he leaves the queue and starts to be served.
- The average time an ordinary account holder needs to wait after he joins a queue until he leaves the queue and starts to be served.
- The average time an ordinary account holder needs to wait after he joins queue 1 until he leaves queue 1 and starts to be served.
- The average time an ordinary account holder needs to wait after he joins queue 2 until he leaves queue 2 and starts to be served.
- The average time an ordinary account holder needs to wait after he joins queue 3 until he leaves queue 3 and starts to be served.
- The total number of ordinary account holders served by counter 0.
- The total number of customers each of the counter serves.

The output should be stored in the output file (**bank.out**) in the following format:

On average a VIP account holder needs to wait for 2.25 minutes after he joins a queue until he leaves the queue and starts to be served.
On average an ordinary account holder needs to wait for 5.25 minutes after he joins a queue until he leaves the queue and starts to be served.
Counter 0 serves 10 VIP account holders.
Counter 1 serves 8 VIP account holders.
Counter 2 serves 2 VIP account holders and 10 ordinary account holders.
Counter 3 serves 5 VIP account holders and 7 ordinary account holders.
Counter 4 serves 3 VIP account holders and 15 ordinary account holders.

You should represent a customer as a structure, the type of which is defined as

```
typedef enum {vipBanking, ordinaryBanking} accountType;

typedef struct customerDataT {
    int timeOfArrival;
    int ServiceTimeStillNeeded;
    accountType A;
} customerDataT;
```

The type **queueElementT** in **queue.h** is redefined as

```
typedef struct customerDataT *queueElementT;
```

A counter is a structure of the following type:

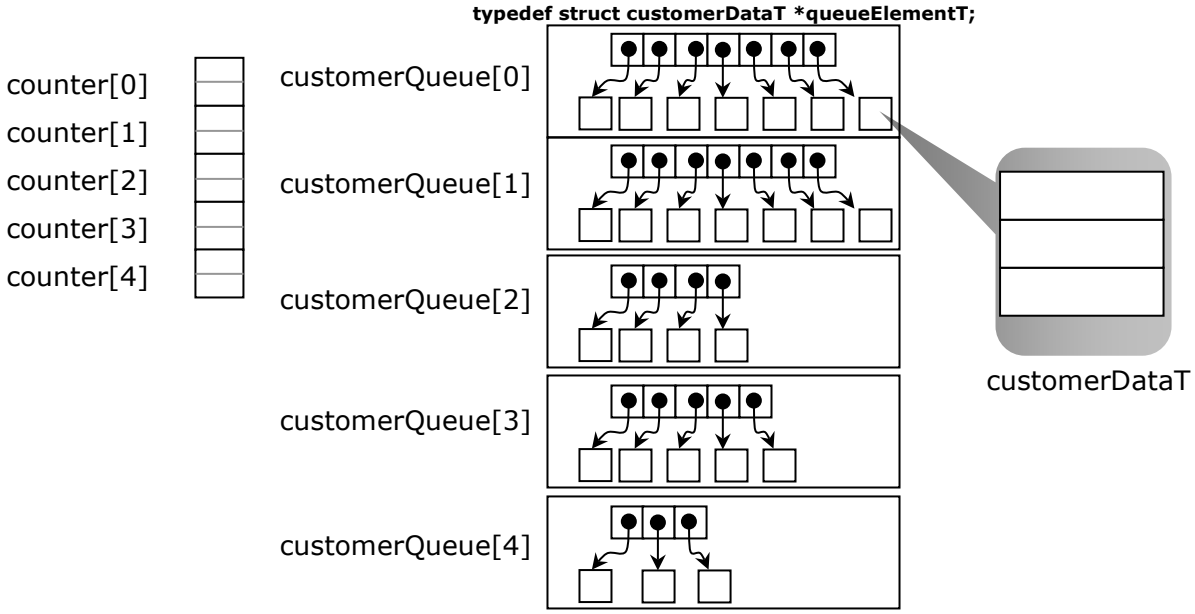
```
typedef struct counterT {
    int numberOfCustomersServed;
    int totalServiceTime;
} counterT;
```

Finally, there are five counters and five queues in the system:

```
counterT counter[5];
queueADT customerQueue[5];
```

customerQueue[i] is the queue of customers waiting in front of counter[i], i = 0, 1, …, 4. The front of customerQueue[i] is the customer being served at counter[i], so he will be dequeued after having been served. A new customer who wish to be served at counter[i] will be enqueued to customerQueue[i].

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



Hint: the main part of the program can be written as a while loop:

```
time = 1;
while ( not end of input file ) {
        read a line from the input file;

        // What happens in this minutes …
        // Just simulate what happens and record all necessary information

        time++; // OK, now we move to the next minute
}
```