HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

BBM203 PROGRAMMING LAB.
ASSIGNMENT 2

Subject	**:** Data Structures and Algorithms
Submission Date	**: 14.11.2019**
Deadline	**: 04.12.2019 23:59:59**
Programming Language	**:** C
Advisor	**:** R.A. Alaettin UÇAN

1. INTRODUCTION / AIM:

In this experiment, you are expected to gain knowledge on C language by applying it to stack and priority queue. Prior knowledge on basic C syntax is required.

2. BACKGROUND INFORMATION

2.1. Stack

In computer science, a stack is an abstract data type that serves as a collection of elements, with two principal operations:

push, which adds an element to the collection, and
pop, which removes the most recently added element that was not yet removed.

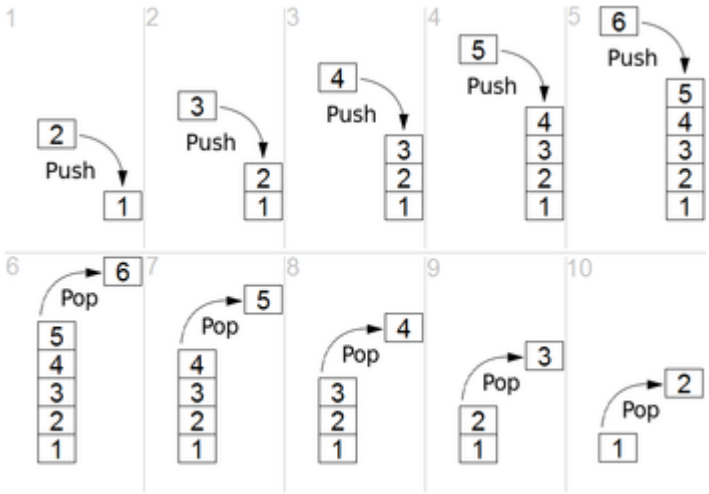

Figure 1 Simple representation of a stack runtime with push and pop operations.

The order in which elements come off a stack gives rise to its alternative name, LIFO (Last In First Out). Additionally, a peek operation may give access to the top without modifying the stack. The name "stack" for this type of structure comes from the analogy of a set of physical items stacked on top of each other, which makes it easy

to take an item from the top of the stack, while accesing to an item deeper in the stack may require taking off multiple other items above the requested item first.

2.2. Queue

In computer science, a queue is a collection in which the entities in the collection are kept in order and the principal operations on the collection are the addition of entities to the rear terminal position, known as enqueue, and removal of entities from the front terminal position, known as dequeue. This makes the queue a First-In-First-Out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first item to be removed. This is equivalent to the requirement that once a new item is added, all elements that were added before that item have to be removed before the new item can be removed. Often a peek or front operation is also entered, returning the value of the front element without dequeuing it. A queue is an example of a linear data structure, or more abstractly a sequential collection.

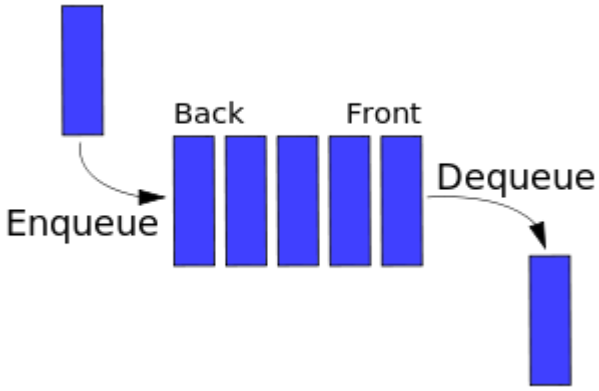

Figure 2 Representation of a FIFO (first in, first out) queue

2.2.1 Priority Queue

Priority queue is an abstract data type which is like a regular queue or stack data structure, but additionally each element has got a "priority" associated with it. In a priority queue, an element with a high priority is served before an element with a low priority.

2.3. DYNAMIC MEMORY ALLOCATION

The task of fulfilling an allocation request consists of locating a block of unused memory of sufficient size. Memory requests are satisfied by allocating portions from a large pool of memory called the heap or free store. At any given time, some parts of the heap are in use, while some are "free" (unused) and thus available for future allocations.

The C dynamic memory allocation functions are defined in stdlib.h header.

Function	Description
malloc	allocates the specified number of bytes
realloc	increases or decreases the size of the specified block of memory. Reallocates it if needed
calloc	allocates the specified number of bytes and initializes them to zero
free	releases the specified block of memory back to the system

2.4. RECURSION

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function. The C programming language supports recursion, i.e., a function to call itself. However, while using recursion, programmers need to be careful to define an exit condition from the function; otherwise, it will go into an infinite loop.

2.5. MEMORY LEAK

A memory leak is a type of resource leak that occurs when a computer program incorrectly manages memory allocations in such a way that memory which is no longer needed is not released. A memory leak may also happen when an object is stored in memory but cannot be accessed by the running code. You can use Valgrind to check for Memory Leak in your program.

2.6. ONLINE RESOURCES

- Wikipedia C Entry: https://en.wikipedia.org/wiki/C_(programming_language)
- C Tutorial : https://www.tutorialspoint.com/cprogramming
- Valgrind : http://valgrind.org/info/about.html

3. EXPERIMENT

In this experiment, you are expected to implement a passenger ticket sales automation system by using stack and priority queue data structures.

For this purpose; a list of the routes, flights, quotas and passengers must be read from the input file.

You should use stack and queue structures to implement all these operations in the C programming language. To store the stacks and queues, you must use dynamic memory allocation.

There are three different seat classes for each flight: *Business*, *Economy* and *Standard*. The number of seats in each class in each aircraft is indicated in the input file. *Business* and *Economy* class seats are limited. These seats are sold on the principle of "first come first served". However, according to the relevant law, "persons with diplomatic identity" must be given priority in the business class. On the other hand, for the same law, veterans must be given priority in the economy class. In this project, you must store “the empty seats in the Stack” and “passengers who want to buy tickets in the Priority Queue” data structure.

Commands:

- **addseat [**flight**] [**class**] [**count**] :** Adds new seats to the flight. If there is no flight, it creates a new flight and adds the number of seats specified in the corresponding seat class to the flight.
  Arguments:
  [flight] = alphanumeric, flight name
  [class] = enum (business, economy, standard), seat class
  [count] = integer, seat amount
  Output:
  *addseats **[**flight**] [**business_seats_#**] [**economy_seats_#**] [**standard_seats_#**]*

Input:
addseat tk1212 business 2
Output:
addseats tk1212 2 0 0

- **enqueue [**flight**] [**class**] [**passenger_name**] [**priority**]:** Adds a new passenger to the ticket queue.
  Arguments:
  [flight] = alphanumeric, flight name
  [class] = enum (business, economy, standard), seat class
  [passenger_name] = alphanumeric, passenger name
  [priority] = enum (veteran, diplomat), Veterans have priority in the economy class and diplomats in the business class. if not specified, the passenger has no priority. In other cases, the passenger is added to the end of the queue. If two priority passengers are added to the queue in succession, the FIFO principle applies.
  Output:
  *queue **[**flight**] [**passenger_name**] [**class**] [**passenger_count_in_queue**]*

Input:
enqueue tk1212 business psg_bus_1
Output:
queue tk1212 psg_bus_1 business 1

- **sell [**flight**]:** It allows to sell tickets to passengers waiting in the queue. First, tickets are sold to those waiting for business class. Then tickets are sold to those waiting for the economy class. After that, tickets are sold to those waiting for the standard class. In case of "no tickets in the business and economy classes" and "available standard tickets", those who wish to buy a business and economy class ticket are sold a standard class ticket.
  Arguments:
  [flight] = alphanumeric, flight name
  Output:
  *sold **[**flight**]** **[**business_tickets_#**]** **[**economy_ tickets _#**]** **[**standard_ tickets _#**]***

Input:
sell tk1212
Output:
sold tk1212 2 3 4

- **close [**flight**]:** This command makes that flight *Read Only*, only information about that flight can be retrieved after this command. Editing such as "adding, selling" cannot be done after this command. It also gives a list of passengers waiting in the queue to get tickets as output, if any.
  Arguments:
  [flight] = alphanumeric, flight name
  Output:
  *closed **[**flight**]** **[**total_tickets_count**]** **[**waiting_passenger_count**]***
  waiting [passenger_1_name]
  waiting [passenger_2_name]
  …

Input:
close tk1212
Output:
closed tk1212 9 2 *waiting psg_eco_3* *waiting psg_eco_4*

- **report [**flight**]:** This command lists the passengers on the flight concerned. The command creates lines for each passenger consisting of "flight number, passenger name and seat class".
  Arguments:
  [flight] = alphanumeric, flight name
  Output:
  *report **[**flight**]***
  *business **[**business_tickets_n**]***
  [passenger_1_name]
  [passenger_2_name]
  …
  [passenger_n_name]
  *economy **[**economy_ tickets _m**]***

[passenger_1_name]
[passenger_2_name]
...
[passenger_m_name]
*standard **[**standard_ tickets _k**]***
[passenger_1_name]
[passenger_2_name]
...
[passenger_k_name]
end of report

Input:
report tk1212
Output:
report tk1212 *business 2* *psg_bus_3* *psg_bus_1* *economy 3* *psg_eco_6* *psg_eco_7* *psg_eco_1* *standard 4* *psg_bus_2* *psg_bus_4* *psg_std_1* *psg_eco_2* *end of report tk1212*

- **info [**passenger_name**]:** This command prints the passenger's flight number, wanted and sold seat class. In the exceptional case where the passenger cannot buy a ticket, it is sufficient to type "none" for the sold seat class.
  Arguments:
  [passenger_name] = alphanumeric, flight name
  Output:
  info [passenger_name] [flight] [wanted_seat_class] [sold_seat_class]

Input:
info psg_bus_4
Output:
info psg_bus_4 tk1212 business standard

3.1. EXECUTION

The name of the compiled executable program should be “sellticket”. Your program should read the input and output file names from the command line, so it will be executed from the command line as follows:

```
sellticket [input file name] [output file name]
e.g.: ./sellticket input.txt output.txt
```

You can see sample input and output files in Piazza page. The program must run on DEV (dev.cs.hacettepe.edu.tr) UNIX machines. So make sure that it compiles and runs in one of the UNIX labs. If we are unable to compile or run, the project risks getting zero point. It is recommended that you test the program using the same mechanism on the sample files (provided) and your own inputs. You must compare your own output with the sample output. If your output is different from the one in the sample file, the project risks getting zero point, too.

3.2. INPUT/OUTPUT FORMAT

In the commands file, the numbers are separated by spaces and newlines. You need to check the input for errors. An example commands file can be given as follows:

```
addseat tk1212 business 2
addseat tk1212 economy 3
addseat tk1212 standard 4
enqueue tk1212 business psg_bus_1
enqueue tk1212 economy psg_eco_1
enqueue tk1212 business psg_bus_2
enqueue tk1212 economy psg_eco_2
enqueue tk1212 standard psg_std_1
enqueue tk1212 economy psg_eco_3
enqueue tk1212 economy psg_eco_4
enqueue tk1212 business psg_bus_3 diplomat
enqueue tk1212 business psg_bus_4
enqueue tk1212 economy psg_eco_6 veteran
enqueue tk1212 economy psg_eco_7 veteran
sell tk1212
report tk1212
close tk1212
info psg_bus_4
info psg_eco_2
info psg_eco_3
info psg_eco_6
```

Your program should write outputs to the output file, so that each line in the output file will contain the result of multiplication. If the above input file is given, the output file should be as below:

```
addseats tk1212 2 0 0
addseats tk1212 2 3 0
addseats tk1212 2 3 4
queue tk1212 psg_bus_1 business 1
queue tk1212 psg_eco_1 economy 1
queue tk1212 psg_bus_2 business 2
```

```
queue tk1212 psg_eco_2 economy 2
queue tk1212 psg_std_1 standard 1
queue tk1212 psg_eco_3 economy 3
queue tk1212 psg_eco_4 economy 4
queue tk1212 psg_bus_3 business 3
queue tk1212 psg_bus_4 business 4
queue tk1212 psg_eco_6 economy 5
queue tk1212 psg_eco_7 economy 6
sold tk1212 2 3 4
report tk1212
business 2
psg_bus_3
psg_bus_1
economy 3
psg_eco_6
psg_eco_7
psg_eco_1
standard 4
psg_std_1
psg_bus_2
psg_bus_4
psg_eco_2
end of report tk1212
closed tk1212 9 2
waiting psg_eco_3
waiting psg_eco_4
info psg_bus_4 tk1212 business standard
info psg_eco_2 tk1212 economy standard
info psg_eco_3 tk1212 economy none
info psg_eco_6 tk1212 economy economy
```

3.3. DESIGN EXPECTATIONS

You must use dynamic memory allocation for your solution. Writing spaghetti code, or static arrays could render your entire assignment “unacceptable” and make it subject to huge (if not complete) grade loss.

3.4. VALID PLATFORMS

Your code will be compiled on gcc. You should not assume the availability of any other non-standard libraries/features. If you use a different compiler, it is your responsibility to ensure that your code has no compilation issues with the version of gcc on the dev server.

4. EVALUATION

4.1. REQUIRED FILES

You should create and submit a ZIP archive in the following structure for evaluation. An invalid structured archive will cause you partial or full score loss.

You are required to submit a Makefile[1], which will be used to compile your program to generate the “sellticket” executable.

Directory	Files	Description
Source	*.c, *.h, Makefile	Program source/header files and Makefile
Report	*.pdf	Your report (Only pdf format is accepted)

4.2. REPORTS

You must write a report which is related to your program. The topics that should be included in the report are:

- Problem definition
- Methods and solution
- Functions implemented and not implemented

4.3. GRADING

Make sure that the source code you submitted is compiled with gcc compiler under Linux OS. Applications that produce compilation or runtime errors cannot be graded.

- The program you have submitted will be tested with five different inputs. You can achieve 13 points for each correct output file, total 65 points.
- Your source code must have "Dynamic memory allocation, stack, priority queue and comment lines". You can achieve 5 points for each item, total 20 points.
- You can achieve 5 points for each topic, total 15 points for report.
- Points you can get from report is directly correlated with the total points you get from execution and code review sections. For example if you get 57/85 from first two sections points you can from report section would be at most 10 points.
- In the event of a memory leak, you will be reduced by half the point you deserve.

LAST REMARKS:

- The output of your program will be graded automatically. Therefore, any difference of the output (even a smallest difference) from the sample output will cause an error

[1] Make file example http://mrbook.org/tutorials/make/

and you will get 0 from execution. Keep in mind that a program that does not work 100% right is a program that works wrong.

- There will be incorrect lines in the input file. Your program should continue to run without giving a runtime error. For the wrong lines in the input file, you must type "error" in the corresponding line in the output file.
- Regardless of the length, use **UNDERSTANDABLE** names to your variables, classes and functions.
- Write **READABLE SOURCE CODE** block
- **You will use online submission system to submit your experiments. https://submit.cs.hacettepe.edu.tr/ Deadline is: 04.12.2019 23:59:59. No other submission method (email or etc.) will be accepted.**
- Do not submit any file via e-mail related with this assignment.
- **SAVE** all your work until the assignment is graded.
- The assignment must be original, **INDIVIDUAL** work. Duplicate or very similar assignments are both going to be punished. General discussion of the problem is allowed, but **DO NOT SHARE** answers, algorithms or source codes.
- You can ask your questions through course's piazza page and you are supposed to be aware of everything discussed in the page.

REFERENCES

1. The C Programming Language (Second Edition), B.W. Kernighan, D. M. Ritchie
2. Let Us C (Computer Science) 8th Edition, Y. P. Kanetkar