HACETTEPE UNIVERSITY COMPUTER SCIENCE ENGINEERING DEPARTMENT

BBM203 SOFTWARE LABORATORY I

PROGRAMMING ASSIGNMENT 2

Advisors: Asst. Prof. Dr. Burcu CAN, Asst. Prof. Dr. Adnan ÖZSOY, Assoc. Prof. Dr. Sevil ŞEN, Res. Assist. Burçak ASAL

Subject : Stack, Queue, Dynamic Memory Allocation
Due Date : 12/11/2017 (23:29)

Student ID : 21228817
Name : Ali Utku ÜNLÜ
E-Mail : aliutkuunlu@gmail.com

1.Introduction

In this assignment this topic titles expected from us;

- Stack implementation,
- Queue Implementation,
- Dynamic memory allocation.

In order to do that we have a simulation like this:

Imagine a system that has exactly one server and at least one client. Every clients and server has their stack and queue.

Client size is depend on the input file 1. First line of this input file shows us to how many clients do we have. For example, if the number is 4 then we know that first 3 of them are clients and last one is our server.

Then we get sizes of que and stacks from other lines. First one is queue size and the last one is stack size.

Format: 4 3 (4 for queue, 3 for stack)

I designed a struck type for clients and server which is called as Item.

Item type has;

- Integer Queue size,
- Integer Stack size,
- Integer Front (which we need for queue and initial value is -1)
- Integer Rear (which we need for queue and initial value is -1)
- Integer Top (which we need for stack and initial value is -1)
- Integer History Top (which we need for history to keep activities and initial value is -1)
- Char Array Queue,
- Char Array Stack,
- Char Array History

When I start to create Items I kept them in an Item type array.

When all the initialization end it is time to start read commands from input file 2.

First line of this input also show us how many command line do we have.

Starting from second line we have commands like these;

Command “A”:

Format: A 2 e

“2” means we will work on the second client. (Assuming item size 4)

“A” means we will add to specific client’s queue. (which is second client)

“e” means the process that will be add to specified queue.

Command “I”:

Format: I 2 e

“2” means we will work on the second client. (Assuming item size 4) or we will work on the server (Assuming item size 2)

“I” means we will add to specific client’s or server’s stack.

“e” means the process that will be add to specified stack.

Command “S”:

Format: S 2 G

“G” means nothing to us so we ignore them.

“2” means we will work on the second client. (Assuming item size 4)

“S” means specific client sent process or an interrupt to server’s queue starting from own stack, if stack is empty then sent from own queue.

Command “O”

Format: O G G

“G” means nothing to us so we ignore them.

“O” means server start to run own process or interrupt.

We have there type of error:

1- Queue Full Error:

If a specific item’s queue is full during to process addition, we will give an error with character ‘1’.

2- Stack Full Error:

If a specific item’s stack is full during to interrupt addition, we will give an error with character ‘2’.

3- Stack and Queue Empty Error:

If a specific item’s stack and queue are empty during sending and operating command (“S”, “O”) we will give an error with character ‘3’.

2.Functions

- Line Count Function:

It calculates the given input file’s line number. We know the first line of input file has the line numbers but my design wants to know first line number before reading the file.

Prototype is:

int File_Line_Count(char* s); s is our argument which is given in the command line. It returns an integer value corresponding to line number.

Using like this:

int lineCount1 = File_Line_Count(argv[1]);

- Read File Function:

It takes two parameters one of them is our first command line argument and the other one is an empty two dimensional char array. It reads the argument and pass to the array.

Prototype is:
void Read_File(char* s, char** matrix); s is our command line argument which is given in the command line. Matrix is our 2D dynamic array which is stores the input data.

Using like this:
Read_File(argv[1], input1);

- Push Function:

This function designed for the stack implementation. It adds items to stack.

It takes three parameters which are element to add to stack, current top index of stack (by sending its reference) and which array we want to add the element. When item has added the function increase the top value by one.

Prototype is:
void push (char x, int *top, char array[]);

Using like this:
push ('a', &top, stack);

- Pop Function:

This function also designed for the stack implementation. In practice it removes the item on the top index. But in theory it just blocks us the access that item. When function called it decrease the current top value by one.

It takes a parameter which is current top index of stack (by sending its reference).

Prototype is:
void pop (int *top);

Using like this:
pop(&top);

- Is Stack Empty Function:

This function controls the stack if its empty it returns true, if it's not returns false. It takes a parameter which is current top index of stack (by sending its reference).

If the current top index is equal to -1 it means the stack is empty.

Prototype is:
bool isStackEmpty(int *top);

Using Like This:
If(isStackEmpty(&top))

- Is Stack Full Function:

This function controls the stack if its full it returns true, if it's not returns false. It takes two parameters one of them is current top index of stack (by sending its reference). Last one is size of the stack.

If the current top index is equal to size-1 it means the stack is full.

Prototype is:
bool isStackFull(int *top, int size);

Using Like This:
If(isStackFull(&top, size))

- Show Stack Function:

This function prints the element of stack. It takes three parameters first one is current top index of stack (by sending its reference) the second one is an array to print and last one is a output type File pointer which we write the output on it(by sending its reference).

Prototype is:
void showStack(int *top, char array[], FILE* output);

Using Like This:
showStack(&top, stack, output);

- EnQueue Function:

This function designed for the queue implementation. It adds items to queue.

It takes five parameters first one is the element to add, next one is rear value (by sending its reference), next one is front value (by sending its reference), next one is an array that we want to add item on it. Last one is the size of queue.

I used a circular queue for this assignment, when we add an item to queue we increment the rear value by one. What if my current rear value is equal to size -1 therefore my increment function become like this.

(*pRear) = ((*pRear)+1) % size;

It updates its value by its one more value's mode by size.

Prototype is:
void enQueue(char x, int *pRear, int *pFront, char array[], int size);

Using Like This:
enqueuer('t', &rear, &front, queue, size);

- DeQueue Function:

This function also designed for the queue implementation. In practice it removes the item on the front index. But in theory it just blocks us the access that item. When function called it increase the current front value by one same as in the enQueue function.

If current rear value and current front value are equal what means our queue is empty, then we update that values by '-1'.

It takes three parameters first one is rear value (by sending its reference), next one is front value (by sending its reference) and the last one is queue size.

Prototype is:
void deQueue(int *pRear, int *pFront, int size);

Using Like This:
deQueue(&rear, &front, size);

- Is Queue Empty Function:

This function controls the queue if its empty it returns true, if it's not returns false. It takes a parameter which is current front index of queue (by sending its reference). If it is equal to -1 it means the queue is empty.

Prototype is:
bool isQueueEmpty(int *pFront);

Using Like This:
If(isQueueEmpty(&front))

- Is Queue Full Function:

This function controls the queue if its full it returns true, if it's not returns false. It takes three parameters one of them is current rear index of queue (by sending its reference), next one is current front index of queue (by sending its reference) Last one is size of queue.

There are two possible situations for full queue:

First one is if rear value is equal to front -1 then it is full.

Second one is if front value is equal to 0 and rear value is equal to size-1 then it is also full.

Prototype is:
bool isQueueFull(int *pRear, int *pFront, int maxSize);

Using Like This:
If(isQueueFull(&rear, &front, size))

- Split Function:

This function split the given 2D array according to white spaces and fill the given string array.

It takes three parameters one of them our input matrix, next one is our empty string array and the last one is first arrays row numbers.

Prototype is:

```
void split(char** matrix,char** result, int lineCount);
```

Using Like This:

```
split(input1,splitInput1,lineCount1);
```