EXPERIMENT NO. 4

Author: Harsheet Chordiya

Roll No.: 37

Sem and Section: 3 CSEB

Source file: expt04.c

Date: 24/12/2021

==========

Aim: To study and implement different linear data structures using arrays - Stack ADT and Queue ADT.

PROBLEM STATEMENT:

Use an array-based allocation to initialize a Stack, a Queue, and a Circular Queue and implement the permissible operations on them. Write a menu driven program in C to test these data structures. The solution involving use of structure(s) to realize the mentioned data structures will be preferred (but not essential).

==========

THEORY

==========

linked list- In data structure, a linked list is a linear collection of data elements whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which when combined together represent a sequence. doubly linked list - We add a pointer to the previous node in a doubly-linked list. Thus, we can go in either direction: forward or backward., Singly Linked list.- It is the most common. Each node has data and a pointer to the next node.

Circular Linked list.- A circular linked list is a variation of a linked list in which the last element is linked to the first element. This forms a circular loop.

Stacks-

A stack is a recursive data structure.

A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. push adds an item to the top of the stack, pop removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top.

Queues

A queue is a container of objects (a linear collection) that are inserted and removed according to the first-in first-out (FIFO) principle. New additions to a line made to the back of the queue, while removal (or serving) happens in the front. In the queue only two operations are allowed enqueue and dequeue. Enqueue means to insert an item into the back of the queue, dequeue means removing the front item. The picture demonstrates the FIFO access.

Algorithm for operations on circular queue:

INIT(QUEUE,FORNT,REAR,COUNT)

This algorithm is used to initialize circular queue.

FRONT = 1;

REAR = 0;

COUNT = 0;

Return;

INSERT-ITEM(QUEUE, FRONT, REAR, MAX, COUNT, ITEM)

This algorithm is used to insert or add item

into circular queue.

1. If (COUNT = MAX) then

   Display “Queue overflow”;

   Return;

2. Otherwise

   If (REAR = MAX) then

   i. REAR = 1;

   Otherwise

i. REAR = REAR + 1;

QUEUE(REAR) := ITEM;

COUNT = COUNT + 1;

3. Return;

REMOVE-ITEM(QUEUE, FRONT, REAR, COUNT, ITEM)

This algorithm is used to remove or delete item

from circular queue.

1. If (COUNT = 0) then

Display “Queue underflow”;

Return

2. Otherwise

ITEM = QUEUE(FRONT)l

If (FRONT =MAX) then

i. FRONT = 1;

Otherwise

i. FRONT = FRONT + 1;

COUNT= COUNT + 1;

3. Return

EMPTY-CHECK(QUEUE,FRONT,REAR,MAX,COUNT,EMPTY)

This is used to check queue is empty or not.

1. If(COUNT = 0) then

EMPTY = true;

2. Otherwise

EMPTY = false;

3. Return

FULL-CHECK(QUEUE,FRONT,REAR,MAX,COUNT,FULL)

This algorithm is used to check queue is full or not.

1. If (COUNT = MAX) then

   FULL = true;

2. Otherwise

   FULL = false;

3. Return

==========

PROGRAM

==========

```
//Header File Inclusions
#include<stdio.h>
#include<stdbool.h>
#include<stdlib.h>
// pre-processor directives
#define MAX 1000
// stack ADT
typedef struct stack {
    int arr[MAX];
    int top;
} stack;

// queue ADT
typedef struct queue{
    int arr[MAX];
    int front;
    int rear;
} queue;
```

```
// circular queue ADT
typedef struct circularqueue{
    int front;
    int rear;
    int arr[MAX];
} cqueue;
//Function Declarations/Signatures/Prototypes
stack* initialize_stack(stack *);
bool empty_stack(struct stack *);
bool full_stack(stack *stk);
void push_stack(stack *, int );
void pop(stack *);
void topval(stack *);
int getsize();
int getchoice(int );
stack* Stack(stack* , bool );
queue* initialize_queue(queue *);
bool is_Q_Full(queue *);
bool is_Q_Empty(queue *);
void insert(queue *, int );
void delete(queue *);
void atRear(queue *);
void atFront(queue *);
void display(queue *);
queue* Queue(queue* , bool );
cqueue* initialize_cqueue(cqueue *);
bool is_CQ_Full(cqueue *);
bool is_CQ_Empty(cqueue *);
void insert_cq(cqueue *, int );
```

```
void delete_cq(cqueue *);

void atRear_CQ(cqueue *);

void atFront_CQ(cqueue *);

void display_CQ(cqueue *);

cqueue* CQueue(cqueue* , bool );

void menu_driver();



// the main()

int main (){

    menu_driver();

    return 0;

}



// Function Defination/Body

stack* initialize_stack(stack *stk) {

    stk->top = -1;

    return stk;

}


bool empty_stack(struct stack *stk) {return stk->top == -1;}


bool full_stack(stack *stk) {return stk->top == MAX - 1;}


void push_stack(struct stack *stk, int element){

    if (full_stack(stk))

        printf("\nStack is full -- push\n");

    else {

        stk->top += 1;
```

```
        stk->arr[stk->top] = element;
    }
    return;
}

void pop(struct stack *stk) {
    if (empty_stack(stk))
        printf("\nstack is empty -- pop\n");
    else
        stk->top -= 1;
    return;
}

void topval(struct stack *stk) {
    if (!empty_stack(stk))
        printf("\ntopval = %d\n",stk->arr[stk->top]);
    else
        printf("\nstack is empty -- topval\n");
}

int getsize(){
    int size;
    printf("\nEnter the size : ");
    do
    {
        fflush(stdin);
        scanf("%d",&size);
        if (size > 1000){
            printf("\nRe-enter the size : ");
        }
```

```
    } while (size > 1000);


    return size;
}


int getchoice(int limit){
    int choice;
    printf("\nEnter the choice : ");
    do
    {
        fflush(stdin);
        scanf("%d",&choice);
        if (choice > limit){
            printf("\nRe-enter the choice : ");
        }


    } while (choice > limit);


    return choice;
}


stack* Stack(stack* st, bool jump){
    printf("\n\nYou are using stack ADT !\n");
    int choice_stack, val;
    // stack* st = (stack*)malloc(sizeof(stack));
    jump ? printf("\n\tInput 1 to initialize stack (must for new stack)") :
printf("\n");
    printf("\n\tInput 2 to pop()");
    printf("\n\tInput 3 to get topval");
```

```
printf("\n\tInput 4 to push()");
while (1){
    printf("\n\nYou are using stack ADT !\n");
    choice_stack = getchoice(4);
    switch (choice_stack){
    case 1:
        st = initialize_stack(st);
        break;
    case 2:
        pop(st);
        break;
    case 3:
        topval(st);
        break;
    case 4:
        printf("\n Enter the element you qant to push into the stack : ");
        scanf("%d",&val);
        push_stack(st,val);
        break;
    default:
        break;
    }

    printf("\nDo you want to continue with the current stack");
    printf("\n\tPress Y for yes !");
    printf("\n\tPress any key for NO !\n");
    fflush(stdin);
    char skip;
    scanf("%c",&skip);
    if (skip == 'Y') continue;
```

```
        else {
            // free(st);
            break;
        }
    }
    return st;
}

queue* initialize_queue(queue *q){
    q->front = -1;
    q->rear = -1;
    return q;
}

bool is_Q_Empty(queue *q) {return q->front == - 1 && q->rear == -1;}

bool is_Q_Full(queue *q) {return q->rear == MAX - 1;}

void insert(queue *q, int val){

    if (is_Q_Full(q)){
        printf("\nQueue is full\n");
        return;
    }
    if (is_Q_Empty(q))
        q->front = 0;
    // q->rear += q->rear;
    printf("\n Enter the element you qant to inset into the queue : ");
    scanf("%d",&val);
    q->arr[++q->rear] = val;
```

```
}

void delete(queue *q){

    if (is_Q_Empty(q))
        printf("\nQueue is empty\n");
    else if (q->front == q->rear){
        q->front = -1;
        q->rear = -1;
    }
    else {
        q->front += 1;
    }
}

void atRear(queue *q){
    if (is_Q_Empty(q))
        printf("\nQueue is empty\n");
    else
        printf("\nrear = %d\n",q->arr[q->rear]);
}

void atFront(queue *q){
    if (is_Q_Empty(q))
        printf("\nQueue is empty\n");
    else
        printf("\nFront = %d\n",q->arr[q->front]);
}

void display(queue *q){
```

```
    if (is_Q_Empty(q))
        printf("\nQueue is empty\n");
    else {
        int i;
        for (i=q->front; i<=q->rear; i++){
            printf(" %d",q->arr[i]);
        }
        printf("\n");
    }
}

queue* Queue(queue* q, bool jump){
    int choice_queue, val;
    // queue* q = (queue*)malloc(sizeof(queue));
    jump? printf("\n\tInput 1 to initialize queue") : printf("\n");
    printf("\n\tInput 2 to insert()");
    printf("\n\tInput 3 to delete()");
    printf("\n\tInput 4 to atRear()");
    printf("\n\tInput 5 to atFront()");
    printf("\n\tInput 6 to display()");
    while (1){
        printf("\n\nYou are using queue ADT !\n");
        choice_queue = getchoice(6);
        switch (choice_queue){
        case 1:
            q = initialize_queue(q);
            break;
        case 2:
            insert(q,val);
            break;
```

```
        case 3:
            delete(q);
            break;
        case 4:
            atRear(q);
            break;
        case 5:
            atFront(q);
            break;
        case 6:
            display(q);
            break;
        default:
            break;
        }

        printf("\nDo you want to continue with the current queue");
        printf("\n\tPress Y for yes !");
        printf("\n\tPress any key for NO !\n");
        fflush(stdin);
        char skip;
        scanf("%c",&skip);
        if (skip == 'Y') continue;
        else {
            // free(q);
            break;
        }
    }
    return q;
}
```

```
cqueue* initialize_cqueue(cqueue *cq){
    cq->front = -1;
    cq->rear = -1;
    return cq;
}

bool is_CQ_Full(cqueue *cq) {return (cq->rear == MAX - 1 && cq->front == 0) ||
(cq->front == cq->rear + 1);}

bool is_CQ_Empty(cqueue *cq) {return cq->front == - 1;}

void insert_cq(cqueue *cq, int val){
    if (is_CQ_Empty(cq)){
        cq->front = 0;
        cq->rear = 0;
    }
    else if (is_CQ_Full(cq)){
        printf("\nCQueue is full\n");
        return;
    }
    else
        cq->rear = (cq->rear + 1) % MAX;
    printf("\n Enter the element you qant to inset into the queue : ");
    scanf("%d",&val);
    cq->arr[cq->rear] = val;
}

void delete_cq(cqueue *cq){
    if (is_CQ_Empty(cq)) {printf("\nQueue is empty\n");}
```

```
    if (cq->front == cq->rear) {
        cq->front = -1;
        cq->rear = -1;
    }
    else {
        cq->front = (cq->front + 1) % MAX;
    }
}

void atRear_CQ(cqueue *cq){
    if (is_CQ_Empty(cq))
        printf("\nQueue is empty\n");
    else
        printf("\nrear = %d\n",cq->arr[cq->rear]);
}

void atFront_CQ(cqueue *cq){
    if (is_CQ_Empty(cq))
        printf("\nQueue is empty\n");
    else
        printf("\nFront = %d\n",cq->arr[cq->front]);
}

void display_CQ(cqueue *cq){
    if (is_CQ_Empty(cq))
        printf("\nQueue is empty\n");
    else {
        int i;
        for (i=cq->front; i<=cq->rear; i++){
            printf(" %d",cq->arr[i]);
```

```
        }
        printf("\n");
    }
}

cqueue* CQueue(cqueue* cq, bool jump){
    int choice_cqueue, val;
    // cqueue* cq = (cqueue*)malloc(sizeof(cqueue));
    jump?
    printf("\n\tInput 1 to initialize circular queue") :
    printf("\n");
    printf("\n\tInput 2 to insert()");
    printf("\n\tInput 3 to delete()");
    printf("\n\tInput 4 to atRear()");
    printf("\n\tInput 5 to atFront()");
    printf("\n\tInput 6 to display()");
    while (1){
        printf("\n\nYou are using circular queue ADT !\n");
        choice_cqueue = getchoice(6);
        switch (choice_cqueue){
        case 1:
            cq = initialize_cqueue(cq);
            break;
        case 2:
            insert_cq(cq,val);
            break;
        case 3:
            delete_cq(cq);
            break;
```

```
        case 4:
             atRear_CQ(cq);
             break;
         case 5:
             atFront_CQ(cq);
             break;
         case 6:
             display_CQ(cq);
             break;
         default:
             break;
         }

         printf("\nDo you want to continue with the current circular queue");
         printf("\n\tPress Y for yes !");
         printf("\n\tPress any key for NO !\n");
         fflush(stdin);
         char skip;
         scanf("%c",&skip);
         if (skip == 'Y') continue;
         else {
             // free(cq);
             break;
         }
    }
    return cq;
}

void menu_driver(){
```

```
int size,choice;
bool jump = true;
// size = getsize();
stack* st = (stack*)malloc(sizeof(stack));
queue* q = (queue*)malloc(sizeof(queue));
cqueue* cq = (cqueue*)malloc(sizeof(cqueue));

while (1){
    printf("\n\tInput 1 for stack");
    printf("\n\tInput 2 for queue");
    printf("\n\tInput 3 for circular queue");
    choice = getchoice(3);
    switch (choice){
    case 1:
        if (!jump){
            empty_stack(st);
        }
        st = Stack(st,jump);
        break;
    case 2:
        if (jump){
            q->front = -1;
            q->rear = -1;
        }
        q =Queue(q,jump);
        break;
    case 3:
        if (jump){
            cq->front = -1;
            cq->rear = -1;
```

```
            }
            cq = CQueue(cq,jump);
            break;
        }
        printf("\nDo you want to continue with any previously defined ADT ?\n");
        printf("\nIf yes then press 'Y");
        printf("\nElse press 'N' to continue with new ADT");
        printf("\nElse press any key except 'Y' or 'N' to exit the program\n");
        char skip;
        fflush(stdin);
        scanf("%c", &skip);
        if (skip == 'Y'){
            jump = false;
        }
        else if (skip == 'N'){
            jump = true;
        }
        else{
            break;
        }
    }
}
```

```
================================================================================
```

EXECUTION TRAIL

```
================================================================================

        Input 1 for stack

        Input 2 for queue

        Input 3 for circular queue

Enter the choice : 1



You are using stack ADT !

        Input 1 to initialize stack (must for new stack)

        Input 2 to pop()

        Input 3 to get topval

        Input 4 to push()

You are using stack ADT !

Enter the choice : 1

Do you want to continue with the current stack

        Press Y for yes !

        Press any key for NO !

Y



You are using stack ADT !

Enter the choice : 2
```

```
stack is empty -- pop

Do you want to continue with the current stack
        Press Y for yes !
        Press any key for NO !
N

Do you want to continue with any previously defined ADT ?

If yes then press 'Y
Else press 'N' to continue with new ADT
Else press any key except 'Y' or 'N' to exit the program
Y

        Input 1 for stack
        Input 2 for queue
        Input 3 for circular queue
Enter the choice : 1


You are using stack ADT !


        Input 2 to pop()
        Input 3 to get topval
        Input 4 to push()

You are using stack ADT !
```

```
Enter the choice : 3

stack is empty -- topval

Do you want to continue with the current stack
        Press Y for yes !
        Press any key for NO !
Y


You are using stack ADT !

Enter the choice : 4

 Enter the element you want to push into the stack
: 35

Do you want to continue with the current stack
        Press Y for yes !
        Press any key for NO !
Y


You are using stack ADT !

Enter the choice : 3

topval = 35

Do you want to continue with the current stack
```

```
        Press Y for yes !
        Press any key for NO !
N

Do you want to continue with any previously defined ADT ?

If yes then press 'Y
Else press 'N' to continue with new ADT
Else press any key except 'Y' or 'N' to exit the program
N

        Input 1 for stack
        Input 2 for queue
        Input 3 for circular queue
Enter the choice : 2

        Input 1 to initialize queue
        Input 2 to insert()
        Input 3 to delete()
        Input 4 to atRear()
        Input 5 to atFront()
        Input 6 to display()

You are using queue ADT !

Enter the choice : 1
Do you want to continue with the current queue
        Press Y for yes !
        Press any key for NO !
Y
```

```
You are using queue ADT !

Enter the choice : 3

Queue is empty

Do you want to continue with the current queue
        Press Y for yes !
        Press any key for NO !
Y


You are using queue ADT !

Enter the choice : 4

Queue is empty

Do you want to continue with the current queue
        Press Y for yes !
        Press any key for NO !
N

Do you want to continue with any previously defined ADT ?
If yes then press 'Y
Else press 'N' to continue with new ADT
Else press any key except 'Y' or 'N' to exit the program
Y
```

```
        Input 1 for stack
        Input 2 for queue
        Input 3 for circular queue
Enter the choice : 2


        Input 2 to insert()
        Input 3 to delete()
        Input 4 to atRear()
        Input 5 to atFront()
        Input 6 to display()

You are using queue ADT !

Enter the choice : 2

 Enter the element you want to insert into the queue : 55

Do you want to continue with the current queue
        Press Y for yes !
        Press any key for NO !
Y


You are using queue ADT !

Enter the choice : 2

 Enter the element you want to insert into the queue : 64
```

```
Do you want to continue with the current queue
        Press Y for yes !
        Press any key for NO !
Y


You are using queue ADT !

Enter the choice : 4

rear = 64

Do you want to continue with the current queue
        Press Y for yes !
        Press any key for NO !
Y


You are using queue ADT !

Enter the choice : 5

Front = 55

Do you want to continue with the current queue
        Press Y for yes !
        Press any key for NO !
N
```

```
Do you want to continue with any previously defined ADT ?

If yes then press 'Y
Else press 'N' to continue with new ADT
Else press any key except 'Y' or 'N' to exit the program
N

        Input 1 for stack
        Input 2 for queue
        Input 3 for circular queue
Enter the choice : 3

        Input 1 to initialize circular queue
        Input 2 to insert()
        Input 3 to delete()
        Input 4 to atRear()
        Input 5 to atFront()
        Input 6 to display()

You are using circular queue ADT !

Enter the choice : 1

Do you want to continue with the current circular queue
        Press Y for yes !
        Press any key for NO !
Y


You are using circular queue ADT !
```

```
Enter the choice : 3

Queue is empty

Do you want to continue with the current circular queue
        Press Y for yes !
        Press any key for NO !
Y



You are using circular queue ADT !

Enter the choice : 6
55 64
Do you want to continue with the current circular queue
        Press Y for yes !
        Press any key for NO !
N

Do you want to continue with any previously defined ADT ?

If yes then press 'Y
Else press 'N' to continue with new ADT
Else press any key except 'Y' or 'N' to exit the program
K
```

==========

VIVA-VOICE

==========

1.Can a stack be calles a FILO structure? Justify your answer.

Ans: Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

In FILO (first in, last out), the added or new elements go and appear at the top of the stacks. When the user deletes, those that are on top will be removed. Any elements or items placed on top will be accessed first

2. Can a stack be implemented using queues only(when arrays and linked list are not available as auxiliary data structure)? If yes, how many queues will be required? Elaborate on the general mechanism for this arrangement.

Ans: A stack can be implemented using two queues. Let stack to be implemented be ‘s’ and queues used to implement be ‘q1’ and ‘q2’. Stack ‘s’ can be implemented in two ways: Method 1 (By making push operation costly) This method makes sure that newly entered element is always at the front of ‘q1’, so that pop operation just dequeues from ‘q1’. ‘q2’ is used to put every new element at front of ‘q1’. 1.push(s, x) operation’s step are described below: Enqueue x to q2 One by one dequeue everything from q1 and enqueue to q2.Swap the names of q1 and q2 2.pop(s) operation’s function are described below: Dequeue an item from q1 and return it. Method 2 (By making pop operation costly) In push operation, the new element is always enqueued to q1. In pop() operation, if q2 is empty then all the elements except the last, are moved to q2. Finally the last element is dequeued from q1 and returned. 1.push(s, x) operation: Enqueue x to q1 (assuming size of q1 is unlimited).2.pop(s) operation: One by one dequeue everything except the last element from q1 and enqueue to q2.Dequeue the last item of q1, the dequeued item is result, store it.Swap the names of q1 and q2 Return the item stored in step 2.

3. Can a queue be implemented using stacks only(when arrays and linked list are not available as auxiliary data structure)? If yes, how many stacksw ill be required? Elaborate on the general mechanism for this arrangement.

Ans: A queue can be implemented using two stacks. Let queue to be implemented be q and stacks used to implement q be stack1 and stack2. q can be implemented in two ways:

Method 1 (By making enQueue operation costly): This method makes sure that oldest entered element is always at the top of stack 1, so that deQueue operation just pops from stack1. To put the element at top of stack1, stack2 is

used. enQueue(q, x): While stack1 is not empty, push everything from stack1 to stack2. Push x to stack1 (assuming size of stacks is unlimited).Push everything back to stack1.Here time complexity will be O(n)deQueue(q): If stack1 is empty then error Pop an item from stack1 and return it Here time complexity will be O(1)

4. List few applications of circular queue or a ring buffer.

Ans: Applications of Circular Queues are:

Circular queue is a subtle modification of Linear Queue, specially with fixed length with better space handling. Generally all the applications of Linear Queues are the application of Circular Queues.Memory management: Circular queue is used in memory management.Process Scheduling: A CPU uses a queue to schedule processes.Traffic Systems: Queues are also used in traffic systems.

==========

CONCLUSION

==========

In this experiment we have studied what is the stacks queues and circular queues and their operations. We have also learnt how to write the algorithms for stacks queue and circular queue functions. Sir explained us how the stacks queues and circular queue can be implemented using arrays and the code for this in detail.