**UCS 2312 Data Structures Lab**

**Assignment 5: QueueADT and its application**

**Date of Assignment: 10.10.2023**

Create an ADT for the circular queue data structure using Singly Linked List with the following functions. Each node which consists of Job ID and Burst time. [CO1, K3]

1. createQueue(Q,size) – initialize size
2. enqueue(Q, data) – enqueue data into the queue
3. dequeue(Q)– returns the element at front
4. isEmpty(Q) – returns 1 if queue empty, otherwise returns 0
5. display(Q) – display the elements in queue

Test the operations of queueADT with the following test cases

|  |  |
| --- | --- |
| **Operation** | **Expected Output** |
| dequeue(Q) | Empty |
| enqueue(Q, J1, 2) | (J1,2) |
| enqueue(Q, J2, 13) | (J1,2), (J2,13) |
| enqueue(Q, J3, 5) | (J1,2), (J2,13),(J3,5) |
| dequeue(Q) | (J1,2) |
| display(Q) | (J2,13),(J3,5) |
| dequeue(Q) | (J2,13) |
| display(Q) | (J3,5) |

**Best practices to be followed:**

* Design before coding
* Usage of algorithm notation
* Use of multi-file C program
* Versioning of code

**Application using Queue**

1. Job scheduling

Insert queue with the following contents

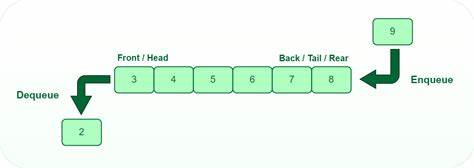
(J1,2), (J2,4), (J3,8), (J4,5), (J5,2), (J6,7), (J7,4), (J8,3) (J9,6) & (J10,6)

Insert the job into the queue whichever is empty. If it is not empty, insert the job into the queue whichever is having minimum average time

Display the jobs waiting in both the queues along with their CPU burst time.

2. Build Queue using stacks

**Data Structure – Queue:**



**Algorithm – Build Queue using Stacks**

**Algorithm: Enqueue**

Input – Pointer to stack 1, pointer to stack 2, data to be added to queue

Output – void

1. if (queueIsFull(s1, s2))

print Queue is Full

1. else

push(s1, data)

**Algorithm: Dequeue**

Input – Pointer to stack 1, pointer to stack 2

Output – int

1. if (queueIsEmpty(s1, s2))

print Queue is Empty

1. else

if (isEmpty(s2))

while (!isEmpty(s1))

push(s2, pop(s1))

return pop(s2)

**Algorithm: queueIsFull**

Input – Pointer to stack 1, pointer to stack 2

Output – int

1. if (s1->top + s2->top < s1->size-2)

return 0

1. return 1

**Algorithm: queueIsEmpty**

Input – Pointer to stack 1, pointer to stack 2

Output – int

1. if (s1->top == -1 && s2->top == -1)

return 1

1. return 0

**queue.h code:**

struct emp

{

char jid[100];

int time;

};

struct queue

{

struct emp e;

struct queue \*next;

};

int isEmpty(struct queue\* ptr[])

{

if(ptr[0]->next == NULL && ptr[1]->next == NULL)

return 1;

else

return 0;

}

void display(struct queue \*q[])

{

for(struct queue\* i = q[0]->next ; i!=NULL ; i = i->next)

printf("%s ", i->e.jid);

}

void enqueue(struct queue\* ptr[], struct emp e)

{

struct queue \*temp = (struct queue\*)malloc(sizeof(struct queue));

strcpy(temp->e.jid, e.jid);

temp->e.time = e.time;

temp->next = NULL;

if(ptr[0]->next == NULL)

ptr[0]->next = temp;

else

ptr[1]->next = temp;

ptr[1] = temp;

}

struct emp\* dequeue(struct queue\* ptr[])

{

struct queue \*ptr1;

struct emp \*e = (struct emp \*)malloc(sizeof(struct emp));

if(ptr[0]->next != NULL)

{

ptr1 = ptr[0]->next;

strcpy(e->jid, ptr1->e.jid);

e->time = ptr1->e.time;

ptr[0]->next = ptr1->next;

free(ptr1);

return e;

}

else

return NULL;

}

**main.c code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include"queue.h"

int main()

{

int choice = 1;

struct queue \*Q[2];

for(int i = 0 ; i < 2 ; i++)

Q[i] = (struct queue\*)malloc(sizeof(struct queue));

Q[0]->next = NULL;

Q[1]->next = NULL;

struct emp e;

printf("Dequeued %s", dequeue(Q)->jid);

display(Q);

printf("\n");

strcpy(e.jid, "J1");

e.time = 2;

enqueue (Q, e);

display(Q);

printf("\n");

strcpy(e.jid, "J2");

e.time = 13;

enqueue(Q, e);

display(Q);

printf("\n");

strcpy(e.jid, "J3");

e.time = 5;

enqueue(Q, e);

display(Q);

printf("\n");

printf("Dequeued %s\n", dequeue(Q)->jid);

display(Q);

printf("\n");

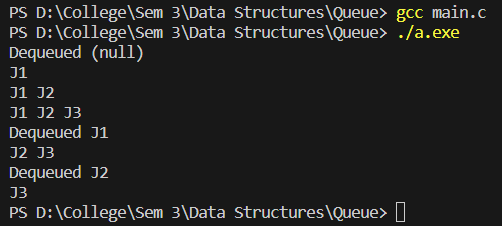
printf("Dequeued %s\n", dequeue(Q)->jid);

display(Q);

printf("\n");

}

**Output Screen:**

****

**APPLICATIONS:**

1. **Job Scheduling**

**empMain.c code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include"empqueueAdt.h"

int waitingTime(struct queue\* ptr[])

{

if(isEmpty(ptr)) return 0;

int sum = 0;

struct queue \*temp = ptr[0]->next ;

while(temp != NULL){

sum += temp->e.time;

temp = temp->next;

}

return sum;

}

void allocateJob(struct queue\* ptr1[], struct queue\* ptr2[], struct emp job)

{

if(waitingTime(ptr1) <= waitingTime(ptr2)) enqueue(ptr1, job);

else enqueue(ptr2, job);

}

int main()

{

int choice = 1;

struct queue \*ptr1[2];

for(int i = 0 ; i < 2 ; i++)

ptr1[i] = (struct queue\*)malloc(sizeof(struct queue));

ptr1[0]->next = NULL;

ptr1[1]->next = NULL;

struct queue \*ptr2[2];

for(int i = 0 ; i < 2 ; i++)

ptr2[i] = (struct queue\*)malloc(sizeof(struct queue));

ptr2[0]->next = NULL;

ptr2[1]->next = NULL;

while(choice)

{

printf("\n\n1. Enqueue\n2. Dequeue queue 1\n3. Dequeue queue 2\n0. Exit\nEnter choice: ");

scanf("%d", &choice);

switch(choice)

{

case 1:

{

struct emp job;

printf("Enter job ID: ");

scanf("%s", &job.jid);

printf("Enter time: ");

scanf("%d", &job.time);

allocateJob(ptr1, ptr2, job);

printf("QUEUE1 > ");

display(ptr1);

printf("\nQUEUE2 > ");

display(ptr2);

break;

}

case 2:

{

struct emp \*deq = dequeue(ptr1);

if(deq != NULL){

printf("%s is removed", deq->jid);

}

printf("\nQUEUE1 > ");

display(ptr1);

printf("\nQUEUE2 > ");

display(ptr2);

break;

}

case 3:

{

struct emp \*deq = dequeue(ptr2);

if(deq != NULL){

printf("%s is removed", deq->jid);

}

printf("\nQUEUE1 > ");

display(ptr1);

printf("\nQUEUE2 > ");

display(ptr2);

break;

}

}

}

}

**empQueueAdt.h code:**

struct emp

{

char jid[100];

int time;

};

struct queue

{

struct emp e;

struct queue \*next;

};

int isEmpty(struct queue\* ptr[])

{

if(ptr[0]->next == NULL && ptr[1]->next == NULL) return 1;

else return 0;

}

void display(struct queue \*q[])

{

for(struct queue\* i = q[0]->next ; i!=NULL ; i = i->next){

printf("%s ", i->e.jid);

}

}

void enqueue(struct queue\* ptr[], struct emp e)

{

struct queue \*temp = (struct queue\*)malloc(sizeof(struct queue));

strcpy(temp->e.jid, e.jid);

temp->e.time = e.time;

temp->next = NULL;

if(ptr[0]->next == NULL){

ptr[0]->next = temp;

}

else

ptr[1]->next = temp;

ptr[1] = temp;

}

struct emp\* dequeue(struct queue\* ptr[])

{

struct queue \*ptr1;

struct emp \*e = (struct emp \*)malloc(sizeof(struct emp));

if(ptr[0]->next != NULL){

ptr1 = ptr[0]->next;

strcpy(e->jid, ptr1->e.jid);

e->time = ptr1->e.time;

ptr[0]->next = ptr1->next;

free(ptr1);

return e;

}

else{

return NULL;

}

}

1. **Queue using Stack**

**queue.h code:**

struct stack

{

int top;

int a[100];

int size;

};

void create(struct stack \*s,int size)

{

s->size=size;

s->top=-1;

}

int isFull(struct stack \*s)

{

if(s->top<(s->size-1))

return 0;

return 1;

}

void push(struct stack \*s,int data)

{

s->a[++s->top]=data;

}

int isEmpty(struct stack \*s)

{

if(s->top==-1)

return 1;

return 0;

}

int pop(struct stack \*s)

{

return(s->a[s->top--]);

}

int peek(struct stack \*s)

{

if(isEmpty(s))

return -1;

else

return s->a[s->top];

}

int queueIsFull(struct stack \*s1, struct stack \*s2)

{

if(s1->top + s2->top < s1->size-2)

return 0;

return 1;

}

int queueIsEmpty(struct stack \*s1, struct stack \*s2)

{

if(s1->top == -1 && s2->top == -1)

return 1;

return 0;

}

void enqueue(struct stack \*s1, struct stack \*s2, int data)

{

if(queueIsFull(s1, s2))

printf("QUEUE IS FULL\n");

else

push(s1, data);

}

int dequeue(struct stack \*s1, struct stack \*s2)

{

if(queueIsEmpty(s1, s2))

printf("QUEUE IS EMPTY\n");

else

{

if(isEmpty(s2))

{

while(!isEmpty(s1))

push(s2, pop(s1));

}

return pop(s2);

}

}

**queue.c code:**

#include <stdio.h>

#include <stdlib.h>

#include "queue.h"

void main()

{

struct stack \*s1=(struct stack \*)malloc(sizeof(struct stack));

struct stack \*s2=(struct stack \*)malloc(sizeof(struct stack));

int size,choice,data;

printf("Enter size : ");

scanf("%d",&size);

create(s1,size);

create(s2,size);

while(choice+1)

{

printf("\n-1: EXIT\n1: ENQUEUE\n2: DEQUEUE\nChoice : ");

scanf("%d",&choice);

switch(choice)

{

case -1:

break;

case 1:

printf("Enter element : ");

scanf("%d",&data);

enqueue(s1,s2, data);

break;

case 2:

if(!queueIsEmpty(s1, s2))

printf("Element Removed = %d\n",dequeue(s1, s2));

else printf("QUEUE IS EMPTY");

break;

default:

printf("Invalid choice\n");

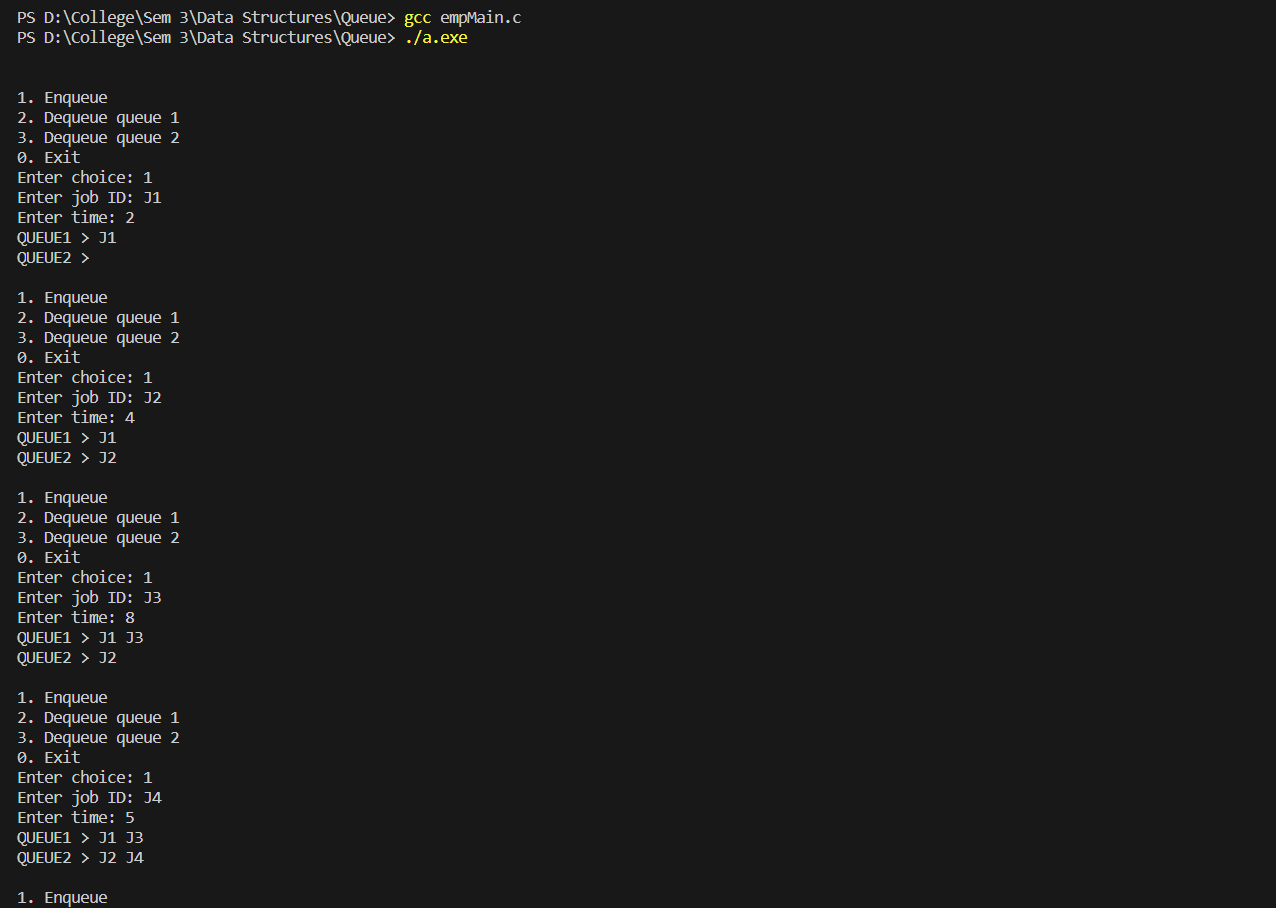
}

}

}

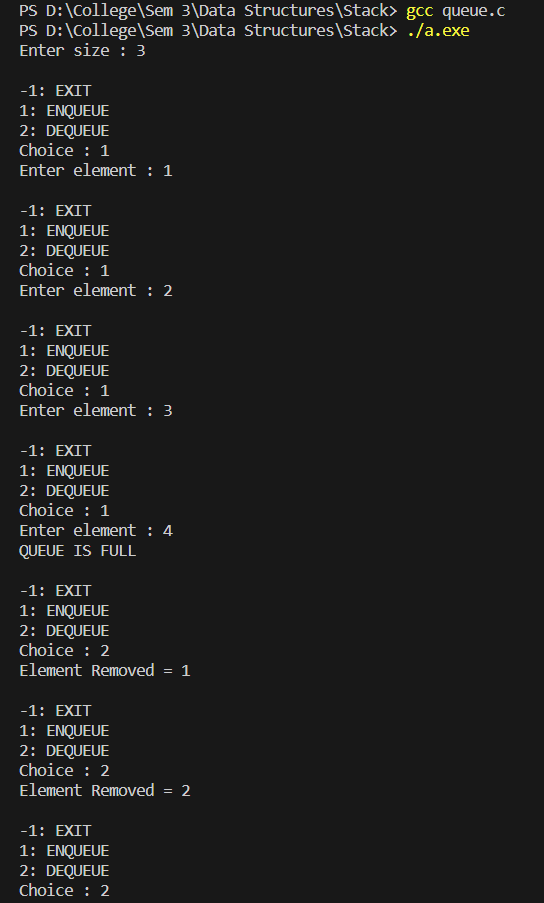
**Output Screen:**

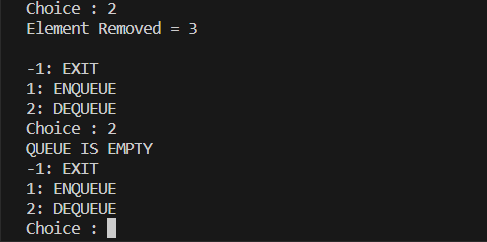
**Job Scheduling –**

****

****

**Queue Using Stack –**

****

****

**Learning Outcome:**

