KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
Deemed to be University U/S 3 of the UGC Act, 1956

A MINI-PROJECT REPORT

ON

"CIRCULAR LINKED LIST"

Semester – III

Section – IT 6

Group – 9

Submission Date –18.11.2018

Submitted by:

Sr#	Roll No	Name	Digital Signature
1	1706439	RENEKA MONDAL	Reneka
2	1706451	SARTHAK ASHISH JAIN	Sarthak.
3	1706460	SOMA MOHANTY	Soma Mohanty

Evaluation Criteria:

Sr#	Section	Weightage
1	Section 2	4
2	Section 3	3
3	Section 4	2
4	Section 5	1

Section – 1

Problem Statement:

In this project we covered circular linked list to:-

-To insert,delete and update a key at beginning,end or at any intermediate position

-Find the largest,smallest element

-Remove the duplicates

-Find pairs with given sum

-Swap kth node from beginning with pth node from the end

Section - 2

2.1. ADT (Abstract Data Type)

It is a logical description on how we view the data and the operations that are allowed without regard on how they will be implemented . Hence, we can say that ADT (Abstract Data Type) is basically a type whose behavior can be defined with a set of values and set of operations.

In our given problem statement we are using queses as an abstract data type.

The queue abstract data type is defined by the following structure and operations. A queue is structured, as described above, as an ordered collection of items which are added at one end, called the “rear,” and removed from the other end, called the “front.”

2.2. Data Structures and Algorithms used

In the given problem statement we are using queue algorithms.

A Queue contains elements of same type arranged in sequential order. Operations takes place at both ends, insertion is done at end and deletion is done at front. Following operations can be performed:
enqueue() − Insert an element at the end of the queue.
dequeue() − Remove and return the first element of queue, if the queue is not empty.
peek() − Return the element of the queue without removing it, if the queue is not empty.
size() − Return the number of elements in the queue.
isEmpty() − Return true if the queue is empty, otherwise return false.

From these definitions, we can clearly see that the definitions do not specify how these ADTs will be represented and how the operations will be carried out. There can be different ways to implement an ADT, for example, the List ADT can be implemented using arrays, or singly linked list or doubly linked list. Similarly, stack ADT and Queue ADT can be implemented using arrays or linked lists.

2.3. Problem Solution Approach

```
//function code to insert data in the queue

void insertq(int data)
{
        struct qnode* nn=(struct qnode*)malloc(sizeof(struct qnode));
        struct qnode* temp;
        nn->data=data;
        nn->next=NULL;
        if(isempty())
        {
                nn->next=nn;
                rear=front=nn;
                return;
        }
        rear->next=nn;
        nn->next=front;
        rear=nn;
}


//function code to delete node from the queue

void deleteq()
{
        struct qnode* temp=front;
        if(isempty())
        {
                printf("\n\tQUEUE IS ALREADY EMPTY \n\tPLEASE TRY SOMETHING ELSE ");
                return;
        }
        if(front==rear)
```

```
        {
                front=rear=NULL;
                printf("\n\t%d DELETED",temp->data);
                free(temp);
                return;
        }
        front=temp->next;
        printf("\n\t%d DELETED",temp->data);
        free(temp);
        rear->next=front;
}

//fuction code to update element at any position

void updateq(int key,int data)
{
        int a,y;
        a=key;
        int x=count();
        if(key>x)
        {
                printf("\n\tKEY IS TOO LARGE");
                return;
        }
        if(key<0)
        {
                printf("\n\tPLEASE ENTER A VALID KEY");
                return;
        }
        struct qnode* temp=front;
        if(isempty())
        {
                printf("\n\tQUEUE IS EMPTY \n\tITEM UPDATION FAILED!");
```

```
                return;
        }
        while(key--)
        {
                temp=temp->next;
        }
        y=temp->data;
        temp->data=data;
        printf("\n\tKEY UPDATED SUCCESFULLY! \n\tPREVIOUS DATA AT %d KEY : %d
\n\tUPDATED DATA AT %d KEY : %d",a,y,a,temp->data);
}



//function code for finding smallest and largest element in the queue

void find()
{
        if(isempty())
        {
                printf("\n\tQUEUE IS EMPTY!");
                return;
        }
        struct qnode*temp=front;
        int small=front->data;
        int large=front->data;
        do
        {
                if(temp->data>large)
                {
                        large=temp->data;
                }
                else
```

```
                {
                        small=temp->data;
                }
                temp=temp->next;
        }while(temp!=front);
        printf("\n\tSMALLEST ELEMENT IS : %d \n\tLARGEST ELEMENT IS : %d",small,large);
}

//function code to swap kth node from begining with pth node from end

void swap(int k,int p)
{
        int i,j;
        int x=count();
        printf("x=%d\n",x);
        if(isempty())
        {
                printf("\n\tQUEUE IS EMPTY \n");
                return;
        }
        if(k>x||p>x)
        {
                printf("\n\tINALID POSITION\n");
                return;
        }
        if(k<0||p<0)
        {
                printf("\n\tINVALID POSITION\n");
                return;
        }
        if(k-1==(x-p))
        {
```

```
                printf("\n\tKth node from begining and pth node from end are at same
position\n");
                return;
        }
                struct qnode* temp1=front;
                struct qnode* temp2=front;
                struct qnode* prev1=front;
                struct qnode* prev2=front;
                struct qnode* temp;

        if(k==1 || p==x)
        {
                prev1=rear;
        }

        else
        {
                for(i=1;i<k-1;i++)
                {
                        prev1=prev1->next;
                }
        }
        temp1=prev1->next;
        temp=temp1->next;
        for(j=1;j<(x-p);j++)
        {
                prev2=prev2->next;
        }
        temp2=prev2->next;
        printf("%d %d\n",temp1->data,temp2->data);

        prev1->next=temp2;
        prev2->next=temp1;
```

```
        temp1->next=temp2->next;
        temp2->next=temp;
        if(k==1)
        {
                front=temp2;
        }
        if(p==1)
        {
                rear=temp1;
        }



}
//fuction code for finding pairs with given sum

void gisum(int sum)
{
if(isempty())
        {
                printf("\n\tQUEUE IS EMPTY");
                return;
        }

struct qnode *temp=front;
struct qnode *prev;
int flag=0;
do
{
        temp=temp->next;
   for(prev=temp->next;prev!=front;prev=prev->next)
          {
          if(prev->data+temp->data==sum)
        {
```

```
                printf("\n\t PAIR IS : %d , %d",prev->data,temp->data);
                flag=1;
        }
         }
}while(temp!=front);
if(flag==0)
printf("\n\tNO PAIR FOUND WITH THE GIVEN SUM");
}
//fuction code to remove duplicate

void deldup()
 {
        if(isempty())
        {
                printf("\n\tQUEUE IS EMPTY");
                return;
        }
  struct qnode *temp=rear;
  struct qnode *agla;
  struct qnode *cur;
  struct qnode* prev;
  do
  {
          temp=temp->next;
    agla=temp->next;
    prev=temp;
          do
          {

           cur=agla;
      if(agla->data==temp->data)
       {
                        agla=agla->next;
```

```
            free(cur);
            prev->next=agla;
        }
        else
        {
                    prev=prev->next;
            agla=agla->next;
                     }
      }while(agla!=front);
    }while(temp->next->next!=front);
    rear=temp->next;
    printf("\nrear %d\n",rear->data);


}

}
```

2.4. Time and Space Complexity

The worst case time complexity of the code is :

Only find() has O(n)+O(n), swap() has O(K-1), gism() has O(n^2), delduc() has O(n(n-1)) and all others have constant order.

The worst case space complexity of the code is:

All function have space complexity size of structure

Section - 3

Source Code:

```
//inserting deleting and updating key in a single circular queue
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>

struct qnode
{
	int data;
	struct qnode* next;
}*front,*rear;

//function to check whether the queue is empty or not
bool isempty()
{
	if(!front)
	return true;
	else
	return false;
}
```

```
//function to insert data in the queue
void insertq(int data)
{
	struct qnode* nn=(struct qnode*)malloc(sizeof(struct qnode));
	struct qnode* temp;
	nn->data=data;
	nn->next=NULL;
	if(isempty())
	{
		nn->next=nn;
		rear=front=nn;
		return;
	}
	rear->next=nn;
	nn->next=front;
	rear=nn;
}



//function to delete node from the queue
```

```
void deleteq()
{
	struct qnode* temp=front;
	if(isempty())
	{
		printf("\n\tQUEUE IS ALREADY EMPTY \n\tPLEASE TRY SOMETHING ELSE ");
		return;
	}
	if(front==rear)
	{
		front=rear=NULL;
		printf("\n\t%d DELETED",temp->data);
		free(temp);
		return;
	}
	front=temp->next;
	printf("\n\t%d DELETED",temp->data);
	free(temp);
	rear->next=front;
}
```

```
//function to count the number of elements in the queue
int count()
{
	struct qnode *temp=front;
	int ctr=0;
	do
	{
		ctr++;
		temp=temp->next;
	}while(temp!=front);

	return ctr;
}

//fuction to update element at any position
void updateq(int key,int data)
{
	int a,y;
	a=key;
	int x=count();
```

```
    if(key>x)
    {
        printf("\n\tKEY IS TOO LARGE");
        return;
    }
    if(key<0)
    {
        printf("\n\tPLEASE ENTER A VALID KEY");
        return;
    }
    struct qnode* temp=front;
    if(isempty())
    {
        printf("\n\tQUEUE IS EMPTY \n\tITEM UPDATION
FAILED!");
        return;
    }
    while(key--)
    {
        temp=temp->next;
    }
```

```
        y=temp->data;
        temp->data=data;
        printf("\n\tKEY UPDATED SUCCESFULLY! \n\tPREVIOUS
DATA AT %d KEY : %d \n\tUPDATED DATA AT %d KEY :
%d",a,y,a,temp->data);
}



//funtion to display the elements of the queue
void displayq()
{
        struct qnode* temp=front;
        if(isempty())
        {
                printf("\n\tQUEUE IS EMPTY!");
                return;
        }
        printf("\n\tELEMENTS IN THE QUEUE ARE : \n");
        do
        {
                printf("\t%d",temp->data);
```

```
            temp=temp->next;
        }while(temp!=front);
}

//finding smallest and largest element in the queue
void find()
{
        if(isempty())
        {
                printf("\n\tQUEUE IS EMPTY!");
                return;
        }
        struct qnode*temp=front;
        int small=front->data;
        int large=front->data;
        do
        {
                if(temp->data>large)
                {
                        large=temp->data;
                }
```

```
            else
            {
                  small=temp->data;
            }
            temp=temp->next;
      }while(temp!=front);
      printf("\n\tSMALLEST ELEMENT IS : %d \n\tLARGEST
ELEMENT IS : %d",small,large);
}

//function to swap kth node from begining with pth node from end
void swap(int k,int p)
{
      int i,j;
      int x=count();
      printf("x=%d\n",x);
      if(isempty())
      {
            printf("\n\tQUEUE IS EMPTY \n");
            return;
      }
```

```
        if(k>x||p>x)
        {
                printf("\n\tINALID POSITION\n");
                return;
        }
        if(k<0||p<0)
        {
                printf("\n\tINVALID POSITION\n");
                return;
        }
        if(k-1==(x-p))
        {
                printf("\n\tKth node from begining and pth node from
end are at same position\n");
                return;
        }
                struct qnode* temp1=front;
                struct qnode* temp2=front;
                struct qnode* prev1=front;
                struct qnode* prev2=front;
                struct qnode* temp;
```

```
if(k==1 || p==x)
{
        prev1=rear;
}

else
{
        for(i=1;i<k-1;i++)
        {
                prev1=prev1->next;
        }
}
temp1=prev1->next;
temp=temp1->next;
for(j=1;j<(x-p);j++)
{
        prev2=prev2->next;
}
temp2=prev2->next;
printf("%d %d\n",temp1->data,temp2->data);
```

```
        prev1->next=temp2;
        prev2->next=temp1;
        temp1->next=temp2->next;
        temp2->next=temp;
        if(k==1)
        {
                front=temp2;
        }
        if(p==1)
        {
                rear=temp1;
        }



}
//fuction for finding pairs with given sum
void gisum(int sum)
{
if(isempty())
        {
```

```
            printf("\n\tQUEUE IS EMPTY");
            return;
        }

struct qnode *temp=front;
struct qnode *prev;
int flag=0;
do
{
        temp=temp->next;
   for(prev=temp->next;prev!=front;prev=prev->next)
          {
          if(prev->data+temp->data==sum)
        {
            printf("\n\t PAIR IS : %d , %d",prev->data,temp->data);
            flag=1;
        }
          }
}while(temp!=front);
if(flag==0)
printf("\n\tNO PAIR FOUND WITH THE GIVEN SUM");
```

```
}
//fuction to remove duplicate
void deldup()
 {
       if(isempty())
       {
             printf("\n\tQUEUE IS EMPTY");
             return;
       }
  struct qnode *temp=rear;
  struct qnode *agla;
  struct qnode *cur;
  struct qnode* prev;
  do
  {
        temp=temp->next;
    printf("temp %d\n",temp->data);
    agla=temp->next;
    prev=temp;
        do
        {
```

```
        printf("A ");


         cur=agla;
       if(agla->data==temp->data)
        {
                    agla=agla->next;
           free(cur);
           prev->next=agla;
        }
        else
        {
                  prev=prev->next;
           agla=agla->next;
                }
      }while(agla!=front);
   }while(temp->next->next!=front);
   rear=temp->next;
   printf("\nrear %d\n",rear->data);
  displayq();

}
```

```
void main()
{
    printf("\n\t OPERATIONS THAT CAN BE PERFORMED USING THIS PROGRAM : \n\n");
    int ch;
    int indata;
    int del,x,a;
    while(1)
    {
    printf("\n");
    printf("\n\t1:INSERT AN ELEMENT IN THE QUEUE");
    printf("\n\t2:DELETE AN ELEMENT FROM THE QUEUE");
    printf("\n\t3:DISPLAY ELEMENTS OF THE QUEUE");
    printf("\n\t4:UPDATE A KEY IN THE QUEUE");
    printf("\n\t5:FIND SMALLEST AND LARGEST ELEMENT");
    printf("\n\t6:SWAP TWO NODES");
   printf("\n\t7:FIND THE GIVEN SUM PAIR");
   printf("\n\t8:DUPLICATE DELETION");
    printf("\n\n\tPRESS ANY KEY TO EXIT!");
```

```
        printf("\n");
        printf("\n\tPLEASE ENTER YOUR CHOICE : ");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:printf("\n\tENTER THE DATA YOU WANT TO
INSERT : ");
                        scanf("%d",&indata);
                        insertq(indata);
                        printf("\n\tDATA INSERTED
SUCCESFULLY!");
                        break;
                case 2:deleteq();
                     break;
                case 3:displayq();
                        break;
                case 4:printf("\n\tENTER THE KEY YOU WANT TO
UPDATE : ");
                        scanf("%d",&x);
                        printf("\n\tENTER THE DATA TO REPLACE :
");
                        scanf("%d",&a);
```

```
                    updateq(x,a);
                    break;
            case 5:find();
                    break;
            case 6:printf("\n\tENTER THE Kth NODE FROM
BEGINING : ");
                    scanf("%d",&x);
                    printf("\n\tENTER THE Pth NODE FROM END :
");
                    scanf("%d",&a);
                    swap(x,a);
                    break;
            case 7:printf("\n\tENTER THE SUM TO BE FOUND: ");
                    scanf("%d",&x);
                    gisum(x);
                     break;
            case 8:printf("\n\tDUPLICATE DELETION:");
                    deldup();
                    break;
            default:exit(0);
      }
```

```
  }
}
```

Section - 4

I/O:

4.1. Test Case 1

OPERATIONS THAT CAN BE PERFORMED USING THIS PROGRAM :

1:INSERT AN ELEMENT IN THE QUEUE

2:DELETE AN ELEMENT FROM THE QUEUE

3:DISPLAY ELEMENTS OF THE QUEUE

4:UPDATE A KEY IN THE QUEUE

5:FIND SMALLEST AND LARGEST ELEMENT

6:SWAP TWO NODES

7:FIND THE GIVEN SUM PAIR

8:DUPLICATE DELETION

PRESS ANY KEY TO EXIT!

PLEASE ENTER YOUR CHOICE : 1

ENTER THE DATA YOU WANT TO INSERT:4

DATA INSERTED SUCCESFULLY!

4.2. Test Case 2

OPERATIONS THAT CAN BE PERFORMED USING THIS PROGRAM :

1:INSERT AN ELEMENT IN THE QUEUE

2:DELETE AN ELEMENT FROM THE QUEUE

3:DISPLAY ELEMENTS OF THE QUEUE

4:UPDATE A KEY IN THE QUEUE

5:FIND SMALLEST AND LARGEST ELEMENT

6:SWAP TWO NODES

7:FIND THE GIVEN SUM PAIR

8:DUPLICATE DELETION

PRESS ANY KEY TO EXIT!

PLEASE ENTER YOUR CHOICE : 1

ENTER THE DATA YOU WANT TO INSERT:6

DATA INSERTED SUCCESFULLY!

4.3. Test Case 3

OPERATIONS THAT CAN BE PERFORMED USING THIS PROGRAM :

1:INSERT AN ELEMENT IN THE QUEUE

2:DELETE AN ELEMENT FROM THE QUEUE

3:DISPLAY ELEMENTS OF THE QUEUE

4:UPDATE A KEY IN THE QUEUE

5:FIND SMALLEST AND LARGEST ELEMENT

6:SWAP TWO NODES

7:FIND THE GIVEN SUM PAIR

8:DUPLICATE DELETION

PRESS ANY KEY TO EXIT!

```
 PLEASE ENTER YOUR CHOICE : 1

ENTER THE DATA YOU WANT TO INSERT:7

DATA INSERTED SUCCESFULLY!
```

4.4. Test Case 4

```
OPERATIONS THAT CAN BE PERFORMED USING THIS PROGRAM :

 1:INSERT AN ELEMENT IN THE QUEUE

 2:DELETE AN ELEMENT FROM THE QUEUE

 3:DISPLAY ELEMENTS OF THE QUEUE

 4:UPDATE A KEY IN THE QUEUE

 5:FIND SMALLEST AND LARGEST ELEMENT

 6:SWAP TWO NODES

 7:FIND THE GIVEN SUM PAIR

 8:DUPLICATE DELETION

  PRESS ANY KEY TO EXIT!


 PLEASE ENTER YOUR CHOICE : 1

ENTER THE DATA YOU WANT TO INSERT:6

DATA INSERTED SUCCESFULLY!
```

4.5. Test Case 5

```
OPERATIONS THAT CAN BE PERFORMED USING THIS PROGRAM :

 1:INSERT AN ELEMENT IN THE QUEUE

 2:DELETE AN ELEMENT FROM THE QUEUE
```

```
 3:DISPLAY ELEMENTS OF THE QUEUE

 4:UPDATE A KEY IN THE QUEUE

 5:FIND SMALLEST AND LARGEST ELEMENT

 6:SWAP TWO NODES

 7:FIND THE GIVEN SUM PAIR

 8:DUPLICATE DELETION

  PRESS ANY KEY TO EXIT!


 PLEASE ENTER YOUR CHOICE : 1

ENTER THE DATA YOU WANT TO INSERT:8

DATA INSERTED SUCCESFULLY!
```

4.6. Test Case 6

```
OPERATIONS THAT CAN BE PERFORMED USING THIS PROGRAM :

 1:INSERT AN ELEMENT IN THE QUEUE

 2:DELETE AN ELEMENT FROM THE QUEUE

 3:DISPLAY ELEMENTS OF THE QUEUE

 4:UPDATE A KEY IN THE QUEUE

 5:FIND SMALLEST AND LARGEST ELEMENT

 6:SWAP TWO NODES

 7:FIND THE GIVEN SUM PAIR

 8:DUPLICATE DELETION

  PRESS ANY KEY TO EXIT!


 PLEASE ENTER YOUR CHOICE : 3
```

ELEMENTS IN THE QUEUE ARE :

4 6 7 6 8

4.7. Test Case 7

OPERATIONS THAT CAN BE PERFORMED USING THIS PROGRAM :

1:INSERT AN ELEMENT IN THE QUEUE

2:DELETE AN ELEMENT FROM THE QUEUE

3:DISPLAY ELEMENTS OF THE QUEUE

4:UPDATE A KEY IN THE QUEUE

5:FIND SMALLEST AND LARGEST ELEMENT

6:SWAP TWO NODES

7:FIND THE GIVEN SUM PAIR

8:DUPLICATE DELETION

PRESS ANY KEY TO EXIT!

PLEASE ENTER YOUR CHOICE : 5

SMALLEST ELEMENT IS : 4

LARGEST ELEMENT IS :8

4.8. Test Case 8

OPERATIONS THAT CAN BE PERFORMED USING THIS PROGRAM :

1:INSERT AN ELEMENT IN THE QUEUE

2:DELETE AN ELEMENT FROM THE QUEUE

3:DISPLAY ELEMENTS OF THE QUEUE

4:UPDATE A KEY IN THE QUEUE

5:FIND SMALLEST AND LARGEST ELEMENT

6:SWAP TWO NODES

7:FIND THE GIVEN SUM PAIR

8:DUPLICATE DELETION

PRESS ANY KEY TO EXIT!

PLEASE ENTER YOUR CHOICE : 7

ENTER THE SUM TO BE FOUND: 13

PAIR IS:6,7

PAIR IS:7,6

Section - 5

Critique of the subject:

The language Data Structures are implemented by programming language by the data types and the references and operations by the particular language like C, C++,Java and so on. EXPERT DATA STRUCTURE WITH C is used as our reference book which uses simple language which is easy to understand and the programs in this book covers the basics to advance. The mid semester questions where challenging but lecture notes were so good that we did not face any kind of difficulty.The chapters covered includes stack,queue,linked list,searching,tress,graph.The subject gives a new direction to think and each day we grow with new idea