///\*\*\*~~~`ALLAH IS ALMIGHTY`~~~\*\*\*///

#include<stdio.h>

#include<stdlib.h>

struct Queue

{

int \*q;

int front;

int rear;

int size;

};

typedef struct Queue queue;

struct Circular\_Queue

{

int \*cq;

int size;

int front;

int rear;

};

typedef struct Circular\_Queue CQ;

struct Linked\_Queue

{

int data;

struct Linked\_Queue \*next;

};

typedef struct Linked\_Queue LQ;

LQ \*front=NULL,\*rear=NULL,\*prev=NULL;

///front=rear=NULL;

/// Implementation of Queue Using Stack.

/// Structure of Array's Stack :

struct Array\_Stack1

{

int \*st1;

int size;

int top;

};

typedef struct Array\_Stack1 ASt1;

struct Array\_Stack2

{

int \*st2;

int size;

int top;

};

typedef struct Array\_Stack2 ASt2;

/// Structure of Linked List's Stack :

struct Linked\_Stack1

{

int data;

struct Linked\_Stack1 \*prev;

};

typedef struct Linked\_Stack1 LSt1;

LSt1 \*top\_1=NULL;

struct Linked\_Stack2

{

int data;

struct Linked\_Stack2 \*prev;

};

typedef struct Linked\_Stack2 LSt2;

LSt2 \*top\_2=NULL;

/// Implementation of Queue Using Array

void Create\_Queue(queue \*Q)

{

printf("Enter a maximum size of the queue : ");

scanf("%d",&Q->size);

Q->q=(int\*)malloc((Q->size)\*sizeof(int));

}

void Enqueue(queue \*Q,int x)

{

if(Q->rear==Q->size-1)

{

printf("The queue is full.\n");

}

else

{

Q->rear++;

Q->q[Q->rear]=x;

}

}

int Dequeue(queue \*Q)

{

int x=-1;

if(Q->rear==Q->front)

{

printf("The queue is empty.\n");

}

else

{

Q->front++;

x=Q->q[Q->front];

}

return x;

}

void Display(queue \*Q)

{

if(Q->front==Q->rear)

{

printf("The queue is empty.\n");

}

else

{

printf("The elements of the queue : ");

for(int i=Q->front+1;i<=Q->rear;i++)

{

printf("%d ",Q->q[i]);

}

printf("\n");

}

}

int Is\_Empty(queue \*Q)

{

if(Q->front==Q->rear)

{

return 1;

}

else

{

return 0;

}

}

int Is\_Full(queue \*Q)

{

if(Q->rear==Q->size-1)

{

return 1;

}

else

{

return 0;

}

}

int Front\_Element(queue \*Q)

{

int x=-1;

if(Q->front==Q->rear)

{

printf("The queue is empty.\n");

}

else

{

x=Q->q[Q->front+1];

}

return x;

}

int Rear\_Element(queue \*Q)

{

int x=-1;

if(Q->front==Q->rear)

{

printf("The queue is empty.\n");

}

else

{

x=Q->q[Q->rear];

}

return x;

}

void Create\_Circular\_Queue(CQ \*Q)

{

printf("Enter a maximum size of circular queue : ");

scanf("%d",&Q->size);

Q->cq=(int\*)malloc((Q->size)\*sizeof(int));

Q->front=Q->rear=0;

}

void Circular\_Enqueue(CQ \*Q,int x)

{

if((Q->rear+1)%Q->size==Q->front)

{

printf("The Circular Queue Is Full.\n");

}

else

{

Q->rear=(Q->rear+1)%Q->size;

Q->cq[Q->rear]=x;

}

}

int Circular\_Dequeue(CQ \*Q)

{

int x=-1;

if(Q->front==Q->rear)

{

printf("The Circular Queue Is Empty.\n");

}

else

{

Q->front=(Q->front+1)%Q->size;

x=Q->cq[Q->front];

}

return x;

}

void Circular\_Display(CQ \*Q)

{

if(Q->rear==Q->front)

{

printf("The Circular Queue Is Empty.\n");

}

else

{

printf("The Elements of Circular Queue : ");

int i=Q->front;

while(i!=Q->rear)

{

i=(i+1)%Q->size;

printf("%d ",Q->cq[i]);

}

printf("\n");

}

}

void Circular\_Front\_Rear(CQ \*Q)

{

printf("Front : %d, Rear : %d\n",Q->front,Q->rear);

}

void Queue\_Front\_Rear(queue \*Q)

{

printf("The front value of Array-Queue : %d\n",Q->front);

printf("The rear value of Array-Queue : %d\n",Q->rear);

}

/// Implementation of Queue Using Linked List.

void Linked\_Equeue(int x)

{

LQ \*new\_node;

new\_node=(LQ\*)malloc(sizeof(LQ));

if(new\_node==NULL)

{

printf("The Queue Is Full.\n");

}

else

{

new\_node->data=x;

new\_node->next=NULL;

if(rear==NULL)

{

front=rear=new\_node;

}

else

{

rear->next=new\_node;

rear=new\_node;

}

}

}

int Linked\_Deueue()

{

int x=-1;

if(rear==NULL)

{

printf("The Queue Is Empty.\n");

}

else

{

LQ \*q=front;

x=q->data;

front=front->next;

free(q);

}

return x;

}

void Display\_Linked\_Queue()

{

if(front==NULL)

{

printf("The Queue Is Empty.\n");

}

else

{

LQ \*p=front;

printf("\nThe Elements of the Linked Queue : ");

while(p)

{

printf("%d ",p->data);

p=p->next;

}

printf("\n");

}

}

/// Double Ended Queue Implementation Using Array.

void Front\_Enqueue(queue \*dq,int x)

{

if(dq->front==-1)

{

printf(" No more element are enqueue along front.\n");

}

else

{

dq->front=dq->front-1;

dq->q[dq->front]=x;

}

}

int Front\_Dequeue(queue \*dq)

{

int x=-1;

if(dq->front==dq->rear)

{

printf(" The queue is empty.\n");

}

else

{

dq->front=dq->front+1;

x=dq->q[dq->front];

}

return x;

}

void Rear\_Enqueue(queue \*dq,int x)

{

if(dq->rear==dq->size-1)

{

printf(" No more element are enqueue along rear.\n");

}

else

{

dq->rear=dq->rear+1;

dq->q[dq->rear]=x;

}

}

int Rear\_Dequeue(queue \*dq)

{

int x=-1;

if(dq->front==dq->rear)

{

printf(" The queue is empty.\n");

}

else

{

x=dq->q[dq->rear];

dq->rear=dq->rear-1;

}

return x;

}

/// Double Ended Queue Implementation Using Linked List.

void Linked\_Front\_Enqueue(int x)

{

LQ \*new\_node;

new\_node=(LQ\*)malloc(sizeof(LQ));

if(new\_node==NULL)

{

printf(" The queue is full.\n");

}

else

{

new\_node->data=x;

new\_node->next=front;

front=new\_node;

}

}

int Linked\_Front\_Dequeue()

{

int x=-1;

if(front==NULL)

{

printf(" The queue is empty.\n");

}

else

{

LQ \*p=front;

x=p->data;

front=front->next;

free(p);

}

return x;

}

void Linked\_Rear\_Enqueue(int x)

{

LQ \*new\_node;

new\_node=(LQ\*)malloc(sizeof(LQ));

if(new\_node==NULL)

{

printf(" The queue is full.\n");

}

else

{

new\_node->data=x;

new\_node->next=NULL;

prev=rear;

rear->next=new\_node;

rear=new\_node;

}

}

int Linked\_Rear\_Dequeue()

{

int x=-1;

if(rear==NULL)

{

printf(" The queue is empty.\n");

}

else

{

LQ \*p=rear;

x=p->data;

rear=prev;

free(p);

}

return x;

}

/// Implementation of Queue Using Stack.

/// Using Array :

void Array\_Create\_Stack1(ASt1 \*St1)

{

printf(" Enter a maximum size of stack-1 : ");

scanf("%d",&St1->size);

St1->st1=(int\*)malloc((St1->size)\*sizeof(int));

}

void Array\_Create\_Stack2(ASt2 \*St2,ASt1 \*St1)

{

St2->st2=(int\*)malloc((St1->size)\*sizeof(int));

}

/// Enqueue :

void Array\_Enqueue\_Stacks(ASt1 \*St1,ASt2 \*St2,int x)

{

if((St1->top+St2->top+2)==St1->size)

{

printf("\n The Queue Is Full.\n");

}

else

{

St1->top=St1->top+1;

St1->st1[St1->top]=x;

}

}

/// Dequeue :

int Pop\_Stack1(ASt1 \*St1)

{

return St1->st1[St1->top];

}

void Push\_Stack2(ASt2 \*St2,int x)

{

St2->top=St2->top+1;

St2->st2[St2->top]=x;

}

int Pop\_Stack2(ASt2 \*St2)

{

int x=St2->st2[St2->top];

St2->top=St2->top-1;

return x;

}

int Array\_Dequeue\_Stacks(ASt1 \*St1,ASt2 \*St2)

{

int x=-1;

if(St1->top==-1&&St2->top==-1)

{

printf(" The Queue Is Empty.\n");

}

else

{

if(St2->top==-1)

{

while(St1->top!=-1)

{

Push\_Stack2(St2,Pop\_Stack1(St1));

St1->top=St1->top-1;

}

x=Pop\_Stack2(St2);

}

else

{

x=Pop\_Stack2(St2);

}

}

return x;

}

void Array\_Display\_Stack1(ASt1 \*St1)

{

if(St1->top==-1)

{

printf(" Stack-1 Is Empty.\n");

}

else

{

printf(" The Elements of Stack-1 : ");

for(int i=0;i<=St1->top;i++)

{

printf("%d ",St1->st1[i]);

}

printf("\n");

}

}

void Array\_Display\_Stack2(ASt2 \*St2)

{

if(St2->top==-1)

{

printf(" Stack-2 Is Empty.\n");

}

else

{

printf(" The Elements of Stack-2 : ");

for(int i=0;i<=St2->top;i++)

{

printf("%d ",St2->st2[i]);

}

printf("\n");

}

}

void Array\_Display\_Queue(ASt1 \*St1,ASt2 \*St2)

{

if(St1->top==-1&&St2->top==-1)

{

printf("The Queue Is Empty.\n");

}

else

{

printf(" Queue's Elements : ");

for(int i=St2->top;i>=0;i--)

{

printf("%d ",St2->st2[i]);

}

for(int i=0;i<=St1->top;i++)

{

printf("%d ",St1->st1[i]);

}

printf("\n");

}

}

/// Using Linked List :

void Linked\_Enqueue\_Stacks(int x)

{

LSt1 \*new\_node;

new\_node=(LSt1\*)malloc(sizeof(LSt1));

if(new\_node==NULL)

{

printf(" The Queue Is Full.\n");

}

else

{

new\_node->data=x;

if(top\_1==NULL)

{

new\_node->prev=NULL;

top\_1=new\_node;

}

else

{

new\_node->prev=top\_1;

top\_1=new\_node;

}

}

}

void Linked\_Display\_Stack1()

{

if(top\_1==NULL)

{

printf(" Stack-1 Is Empty.\n");

}

else

{

LSt1 \*p=top\_1;

printf(" The elements of Stack-1 : ");

while(p)

{

printf("%d ",p->data);

p=p->prev;

}

printf("\n");

}

}

void Linked\_Display\_Stack2()

{

if(top\_2==NULL)

{

printf(" Stack-2 Is Empty.\n");

}

else

{

LSt2 \*p=top\_2;

printf(" The elements of Stack-2 : ");

while(p)

{

printf("%d ",p->data);

p=p->prev;

}

printf("\n");

}

}

void Linked\_Display\_Queue()

{

if(top\_1==NULL&&top\_2==NULL)

{

printf(" The Queue Is Empty.\n");

}

else

{

/// Display Stack-2:

LSt2 \*p2=top\_2;

printf(" The Elements of the Queue : ");

while(p2)

{

printf("%d ",p2->data);

p2=p2->prev;

}

/// Reverse Stack-1:

LSt1 \*p1=top\_1,\*q=NULL,\*tmp=NULL;

while(p1)

{

tmp=p1->prev;

if(p1==top\_1)

{

p1->prev=NULL;

}

else

{

p1->prev=q;

}

q=p1;

p1=tmp;

}

top\_1=q;

/// Display Stack-1:

p1=top\_1;

while(p1)

{

printf("%d ",p1->data);

p1=p1->prev;

}

printf("\n");

/// Reverse Stack-1:

p1=top\_1,q=NULL,tmp=NULL;

while(p1)

{

tmp=p1->prev;

if(p1==top\_1)

{

p1->prev=NULL;

}

else

{

p1->prev=q;

}

q=p1;

p1=tmp;

}

top\_1=q;

}

}

int Linked\_Pop\_Stack1()

{

return top\_1->data;

}

int Linked\_Pop\_Stack2()

{

int x=top\_2->data;

top\_2=top\_2->prev;

return x;

}

void Linked\_Push\_Stack2(int x)

{

LSt2 \*new\_node;

new\_node=(LSt2\*)malloc(sizeof(LSt2));

if(new\_node)

{

new\_node->data=x;

if(top\_2==NULL)

{

new\_node->prev=NULL;

}

else

{

new\_node->prev=top\_2;

}

top\_2=new\_node;

}

}

int Linked\_Dequeue\_Stacks()

{

int x=-1;

if(top\_1==NULL&&top\_2==NULL)

{

printf("\n The Queue Is Empty.\n");

}

else

{

if(top\_2==NULL)

{

while(top\_1)

{

Linked\_Push\_Stack2(Linked\_Pop\_Stack1());

top\_1=top\_1->prev;

}

x=Linked\_Pop\_Stack2();

}

else

{

x=Linked\_Pop\_Stack2();

}

}

return x;

}

int main()

{

int op;

do

{

printf("\n \*\*\*~~~MAIN MENU~~~\*\*\*\n\n");

printf("1 for creating a queue.\n");

printf("2 for enqueue.\n");

printf("3 for dequeue.\n");

printf("4 for checking the empty condition of queue.\n");

printf("5 for checking full condition of queue.\n");

printf("6 for front element of queue.\n");

printf("7 for rear element of queue.\n");

printf("8 for displaying the queue.\n\n");

printf("9 for Creating A Circular Queue.\n");

printf("10 for Circular Enqueue.\n");

printf("11 for Circular Dequeue.\n");

printf("12 for Displaying Circular Queue.\n");

printf("13 for Circular Front-Rear Value.\n\n");

printf("14 Enqueue Operation Using by Linked List.\n");

printf("15 for Dequeue.\n");

printf("16 for Display.\n\n");

printf("17 for Implementation of Double Ended Queue.\n\n");

printf("18 for Front and Rear Value of Array's Queue.\n\n");

printf("19 for Implementation of Queue Using Stack.\n\n");

printf("0 for exit.\n\n");

printf("Enter your option : ");

scanf("%d",&op);

switch(op)

{

case 1:

printf("\n");

queue Q;

Q.front=Q.rear=-1;

Create\_Queue(&Q);

break;

case 2:

printf("\n");

int x;

printf("Enter an integer number to insert into the queue : ");

scanf("%d",&x);

Enqueue(&Q,x);

break;

case 3:

printf("\n");

int d=Dequeue(&Q);

if(d>-1)

{

printf("%d is deleted from the queue.\n",d);

}

break;

case 4:

if(Is\_Empty(&Q))

{

printf("The queue is empty.\n");

}

else

{

printf("The queue is not empty.\n");

}

break;

case 5:

if(Is\_Full(&Q))

{

printf("The queue is full.\n");

}

else

{

printf("The queue is not full.\n");

}

break;

case 6:

printf("\n");

int f=Front\_Element(&Q);

if(f>-1)

{

printf("The front element of queue is : %d\n",f);

}

break;

case 7:

printf("\n");

int r=Rear\_Element(&Q);

if(r>-1)

{

printf("The rear element of queue is : %d\n",r);

}

break;

case 8:

Display(&Q);

break;

case 9:

printf("\n");

CQ cq;

Create\_Circular\_Queue(&cq);

break;

case 10:

printf("\n");

int x1;

printf("Enter an integer number to insert into the circular queue : ");

scanf("%d",&x1);

Circular\_Enqueue(&cq,x1);

break;

case 11:

printf("\n");

int cd=Circular\_Dequeue(&cq);

if(cd>-1)

{

printf("%d is deleted from the queue.\n",cd);

}

break;

case 12:

Circular\_Display(&cq);

break;

case 13:

Circular\_Front\_Rear(&cq);

break;

case 14:

printf("\n");

int x2;

printf("Enter an integer number to insert into the linked queue : ");

scanf("%d",&x2);

Linked\_Equeue(x2);

break;

case 15:

printf("\n");

int ldq=Linked\_Deueue();

if(ldq>-1)

{

printf("%d is deleted from the queue.\n",ldq);

}

break;

case 16:

Display\_Linked\_Queue();

break;

case 17:

printf("\n");

int op1;

do

{

printf(" 1 for Implementation of Double Ended Queue Using Array.\n");

printf(" 2 for Implementation of Double Ended Queue Using Linked List.\n\n");

printf(" 0 for Exit.\n\n");

printf(" Enter your choice : ");

scanf("%d",&op1);

switch(op1)

{

case 1:

printf("\n");

int op2,dx,dlx;

do

{

printf(" 1 for Enqueue to Front.\n");

printf(" 2 for Dequeue from Front\n");

printf(" 3 for Enqueue to Rear.\n");

printf(" 4 for Dequeue from Rear.\n\n");

printf(" 0 for Exit.\n\n");

printf(" Enter your choice : ");

scanf("%d",&op2);

switch(op2)

{

case 1:

printf("\n");

printf(" Enter an integer number to insert into double ended queue : ");

scanf("%d",&dx);

Front\_Enqueue(&Q,dx);

break;

case 2:

printf("\n");

dlx=Front\_Dequeue(&Q);

if(dlx>-1)

{

printf(" %d is deleted from the double ended queue.\n",dlx);

}

break;

case 3:

printf("\n");

printf(" Enter an integer number to insert into double ended queue : ");

scanf("%d",&dx);

Rear\_Enqueue(&Q,dx);

break;

case 4:

printf("\n");

dlx=Rear\_Dequeue(&Q);

if(dlx>-1)

{

printf(" %d is deleted from the double ended queue.\n",dlx);

}

break;

default:

printf("\n Enter a valid option.\n\n");

}

}while(op2!=0);

break;

case 2:

printf("\n");

int op3;

do

{

printf(" 1 for Enqueue to Front.\n");

printf(" 2 for Dequeue from Front\n");

printf(" 3 for Enqueue to Rear.\n");

printf(" 4 for Dequeue from Rear.\n\n");

printf(" 0 for Exit.\n\n");

printf(" Enter your choice : ");

scanf("%d",&op3);

switch(op3)

{

case 1:

printf("\n");

printf(" Enter an integer number to insert into double ended queue : ");

scanf("%d",&dx);

Linked\_Front\_Enqueue(dx);

break;

case 2:

printf("\n");

dlx=Linked\_Front\_Dequeue();

if(dlx>-1)

{

printf(" %d is deleted from the double ended queue.\n",dlx);

}

break;

case 3:

printf("\n");

printf(" Enter an integer number to insert into double ended queue : ");

scanf("%d",&dx);

Linked\_Rear\_Enqueue(dx);

break;

case 4:

printf("\n");

dlx=Linked\_Rear\_Dequeue();

if(dlx>-1)

{

printf(" %d is deleted from the double ended queue.\n",dlx);

}

break;

default:

printf("\n Enter a valid option.\n\n");

}

}while(op3!=0);

break;

default :

printf("\n Enter a valid option.\n\n");

}

}while(op1!=0);

break;

case 18:

printf("\n");

Queue\_Front\_Rear(&Q);

break;

case 19:

printf("\n");

int op4;

do

{

printf(" 1 for Queue Implementation Using Array's Stack.\n");

printf(" 2 for Queue Implementation Using Linked List Stack.\n\n");

printf(" 0 for Exit.\n\n");

printf(" Enter your option : ");

scanf("%d",&op4);

switch(op4)

{

case 1:

printf("\n");

int op5;

do

{

printf(" 1 for Create Stack-1 and Stack-2.\n");

printf(" 2 for Enqueue.\n");

printf(" 3 for Dequeue.\n");

printf(" 4 for Display Stack-1.\n");

printf(" 5 for Display Stack-2.\n");

printf(" 6 for Display Queue.\n\n");

printf(" 0 for Exit.\n\n");

printf(" Enter your option : ");

scanf("%d",&op5);

switch(op5)

{

case 1:

printf("\n");

ASt1 St1;

ASt2 St2;

Array\_Create\_Stack1(&St1);

Array\_Create\_Stack2(&St2,&St1);

St1.top=St2.top=-1;

printf("\n");

break;

case 2:

printf("\n");

int ax;

printf(" Enter an integer number to push into the stack-1 : ");

scanf("%d",&ax);

Array\_Enqueue\_Stacks(&St1,&St2,ax);

printf("\n");

break;

case 3:

printf("\n");

int adx=Array\_Dequeue\_Stacks(&St1,&St2);

if(adx>-1)

{

printf(" %d is deleted from the queue.\n",adx);

}

printf("\n");

break;

case 4:

printf("\n");

Array\_Display\_Stack1(&St1);

printf("\n");

break;

case 5:

printf("\n");

Array\_Display\_Stack2(&St2);

printf("\n");

break;

case 6:

printf("\n");

Array\_Display\_Queue(&St1,&St2);

printf("\n");

break;

default:

printf("\n Enter a valid option.\n\n");

}

}while(op5!=0);

break;

case 2:

printf("\n");

int op6;

do

{

printf(" 1 for Enqueue.\n");

printf(" 2 for Dequeue.\n");

printf(" 3 for Display Stack-1.\n");

printf(" 4 for Display Stack-2.\n");

printf(" 5 for Display Queue.\n\n");

printf(" 0 for Exit.\n\n");

printf(" Enter your option : ");

scanf("%d",&op6);

switch(op6)

{

case 1:

printf("\n");

int lx;

printf(" Enter an integer number to insert into the queue : ");

scanf("%d",&lx);

Linked\_Enqueue\_Stacks(lx);

printf("\n");

break;

case 2:

printf("\n");

int ldx=Linked\_Dequeue\_Stacks();

if(ldx>-1)

{

printf(" %d Is Deleted from the Queue.\n",ldx);

}

printf("\n");

break;

case 3:

printf("\n");

Linked\_Display\_Stack1();

printf("\n");

break;

case 4:

printf("\n");

Linked\_Display\_Stack2();

printf("\n");

break;

case 5:

printf("\n");

Linked\_Display\_Queue();

printf("\n");

break;

default:

printf("\n Enter an valid option.\n\n");

}

}while(op6!=0);

break;

default:

printf("\n Enter a valid option.\n\n");

}

}while(op4!=0);

break;

default:

printf("\nEnter a valid option.\n\n");

}

}while(op!=0);

return 0;

}