//Stack Implementation using Arrays.

#include <stdio.h>

#include <stdlib.h>

typedef struct Stack

{

int top;

int size;

int \*arr;

}Stack;

int isEmpty(Stack \*stack)

{

if(stack->top==-1)

return 1;

return 0;

}

int isFull(Stack \*stack)

{

if(stack->size-1==stack->top)

return 1;

return 0;

}

void push(Stack \*stack,int ele)

{

if(isFull(stack))

printf("Overflow.\n");

else

stack->arr[++stack->top]=ele;

}

int pop(Stack \*stack)

{

if(isEmpty(stack))

{

printf("Underflow.\n");

return -9999;

}

return stack->arr[stack->top--];

}

int top(Stack \*stack)

{

if(isEmpty(stack))

{

printf("Underflow.\n");

return -9999;

}

return stack->arr[stack->top];

}

void display(Stack \*stack)

{

if(isEmpty(stack))

printf("Stack is Empty.\n");

else

{

printf("Stack looks like.......\n");

printf("%d <-----TOP\n",stack->arr[stack->top]);

int k;

for(k=stack->top-1;k>=0;k--)

printf("%d\n",stack->arr[k]);

}

}

void main()

{

int ch,n,v,p;

printf("Enter the max size of the stack : ");

scanf("%d",&n);

Stack stack={-1,n,(int \*)malloc(n\*sizeof(int))};

while(1)

{

printf("------------------------------------\n");

printf("1. Push.\n");

printf("2. Pop.\n");

printf("3. Top.\n");

printf("4. Display.\n");

printf("5. Check if the Stack is Full.\n");

printf("6. Check if the Stack is Empty.\n");

printf("7. Exit.\n");

printf("Enter your choice : \n");

scanf("%d",&ch);

printf("------------------------------------\n");

switch(ch)

{

case 1 :

printf("Enter the no. to be pushed : ");

scanf("%d",&v);

push(&stack,v);

break;

case 2 :

p=pop(&stack);

if(p!=-9999)

printf("Popped element : %d\n",p);

break;

case 3 :

p=top(&stack);

if(p!=-9999)

printf("Topmost element : %d\n",p);

break;

case 4 :

display(&stack);

break;

case 5 :

if(isFull(&stack))

printf("Stack is Full.\n");

else

printf("Stack is not Full.\n");

break;

case 6 :

if(isEmpty(&stack))

printf("Stack is Empty.\n");

else

printf("Stack is not Empty.\n");

break;

case 7 :

free(stack.arr);

exit(0);

default :

printf("Wrong Choice.\n");

}

}

}

//Stack Implementation using Linked List.

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int data;

struct node \*link;

}Node;

typedef struct Stack

{

Node\* top;

}Stack;

int isEmpty(Stack \*stack)

{

if(stack->top==NULL)

return 1;

return 0;

}

void push(Stack \*stack,int ele)

{

Node \*newnode=(Node \*)malloc(sizeof(\*newnode));

if(newnode==NULL)

{

printf("Overflow.\n");

return;

}

newnode->data=ele;

newnode->link=NULL;

if(stack->top==NULL)

stack->top=newnode;

else

{

newnode->link=stack->top;

stack->top=newnode;

}

}

int pop(Stack \*stack)

{

if(isEmpty(stack))

{

printf("Underflow.\n");

return -9999;

}

Node \*temp;

temp=stack->top;

stack->top=stack->top->link;

int tmp=temp->data;

free(temp);

return tmp;

}

int top(Stack \*stack)

{

if(isEmpty(stack))

{

printf("Underflow.\n");

return -9999;

}

return stack->top->data;

}

void display(Stack \*stack)

{

if(isEmpty(stack))

printf("Stack is Empty.\n");

else

{

printf("Stack looks like.......\n");

printf("%d <-----TOP\n",stack->top->data);

Node \*temp=stack->top->link;

while(temp!=NULL)

{

printf("%d\n",temp->data);

temp=temp->link;

}

}

}

void main()

{

int ch,ele,p;

Stack stack;

stack.top=NULL;

while(1)

{

printf("------------------------------------\n");

printf("1. Push.\n");

printf("2. Pop.\n");

printf("3. Top.\n");

printf("4. Display.\n");

printf("5. Check if the Stack is Empty.\n");

printf("6. Exit.\n");

printf("Enter your choice : \n");

scanf("%d",&ch);

printf("------------------------------------\n");

switch(ch)

{

case 1 :

printf("Enter the no. to be pushed : ");

scanf("%d",&ele);

push(&stack,ele);

break;

case 2 :

p=pop(&stack);

if(p!=-9999)

printf("Popped element : %d\n",p);

break;

case 3 :

p=top(&stack);

if(p!=-9999)

printf("Topmost element : %d\n",p);

break;

case 4 :

display(&stack);

break;

case 5 :

if(isEmpty(&stack))

printf("Stack is Empty.\n");

else

printf("Stack is not Empty.\n");

break;

case 6 :

//Freeing memory.

while(!isEmpty(&stack))

pop(&stack);

exit(0);

default :

printf("Wrong Choice.\n");

}

}

}

//Queue Implementation using Arrays.

#include <stdio.h>

#include <stdlib.h>

typedef struct Queue

{

int size;

int front;

int rear;

int \*arr;

}Queue;

int isFull(Queue \*queue)

{

if(queue->rear==(queue->front+queue->size))

return 1;

return 0;

}

int isEmpty(Queue \*queue)

{

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

return 1;

return 0;

}

void enqueue(Queue \*queue,int ele)

{

if(isFull(queue))

printf("Overflow.\n");

else

queue->arr[(++queue->rear)%queue->size]=ele;

}

int dequeue(Queue \*queue)

{

if(isEmpty(queue))

{

printf("Underflow.\n");

return -9999;

}

return queue->arr[(++queue->front)%queue->size];

}

int front(Queue \*queue)

{

if(isEmpty(queue))

{

printf("Underflow.\n");

return -9999;

}

return queue->arr[(queue->front+1)%queue->size];

}

int rear(Queue \*queue)

{

if(isEmpty(queue))

{

printf("Underflow.\n");

return -9999;

}

return queue->arr[(queue->rear)%queue->size];

}

void display(Queue \*queue)

{

if(isEmpty(queue))

printf("Queue is Empty.\n");

else

{

printf("Queue looks like.......\n");

if(queue->front+1==queue->rear)

printf("%d <----FRONT <----REAR\n",queue->arr[(queue->front+1)%queue->size]);

else

{

printf("%d <----FRONT\n",queue->arr[(queue->front+1)%queue->size]);

int k;

for(k=queue->front+2;k<queue->rear;k++)

printf("%d\n",queue->arr[k%queue->size]);

printf("%d <----REAR\n",queue->arr[(queue->rear)%queue->size]);

}

}

}

void main()

{

int n,v,p,ch;

printf("Enter the max size of the queue : ");

scanf("%d",&n);

Queue queue={n,-1,-1,(int \*)malloc(n\*sizeof(int))};

while(1)

{

printf("-------------------------------------------------------\n");

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

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

printf("3. Front.\n");

printf("4. Rear.\n");

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

printf("6. Check if the Queue is Full.\n");

printf("7. Check if the Queue is Empty.\n");

printf("8. Exit.\n");

printf("Enter your choice : \n");

scanf("%d",&ch);

printf("-------------------------------------------------------\n");

switch(ch)

{

case 1 :

printf("Enter the no. to be enqueued : ");

scanf("%d",&v);

enqueue(&queue,v);

break;

case 2 :

p=dequeue(&queue);

if(p!=-9999)

printf("Dequeued element : %d\n",p);

break;

case 3 :

p=front(&queue);

if(p!=-9999)

printf("Front element : %d\n",p);

break;

case 4 :

p=rear(&queue);

if(p!=-9999)

printf("Rear element : %d\n",p);

break;

case 5 :

display(&queue);

break;

case 6 :

if(isFull(&queue))

printf("Queue is Full.\n");

else

printf("Queue is not Full.\n");

break;

case 7 :

if(isEmpty(&queue))

printf("Queue is Empty.\n");

else

printf("Queue is not Empty.\n");

break;

case 8 :

free(queue.arr);

exit(0);

default :

printf("Wrong Choice.\n");

}

}

}

//Queue Implementation using Linked List.

#include <stdio.h>

#include <stdlib.h>

typedef struct node//It stores contents of each node

{

int data;

struct node \*link;

}Node;

typedef struct Queue

{

Node \*front;

Node \*rear;

}Queue;

int isEmpty(Queue \*queue)

{

if(queue->front==NULL)

return 1;

return 0;

}

void enqueue(Queue \*queue,int ele)

{

Node \*newnode=(Node \*)malloc(sizeof(\*newnode));

if(newnode==NULL)

{

printf("Overflow.\n");

return;

}

newnode->data=ele;

newnode->link=NULL;

if(queue->front==NULL)

queue->front=queue->rear=newnode;

else

{

queue->rear->link=newnode;

queue->rear=queue->rear->link;

}

}

int dequeue(Queue \*queue)

{

if(isEmpty(queue))

{

printf("Underflow.\n");

return -9999;

}

Node \*temp=queue->front;

queue->front=queue->front->link;

int tmp=temp->data;

free(temp);

if(queue->front==NULL)

queue->rear=NULL;

return tmp;

}

int front(Queue \*queue)

{

if(isEmpty(queue))

{

printf("Underflow.\n");

return -9999;

}

return queue->front->data;

}

int rear(Queue \*queue)

{

if(isEmpty(queue))

{

printf("Underflow.\n");

return -9999;

}

return queue->rear->data;

}

void display(Queue \*queue)

{

if(isEmpty(queue))

printf("Queue is Empty.\n");

else

{

printf("Queue looks like.......\n");

if(queue->front->link==NULL)

printf("%d <----FRONT <----REAR\n",queue->front->data);

else

{

printf("%d <----FRONT\n",queue->front->data);

Node \*temp=queue->front->link;

while(temp->link!=NULL)

{

printf("%d\n",temp->data);

temp=temp->link;

}

printf("%d <----REAR\n",temp->data);

}

}

}

void main()

{

int v,p,ch;

Queue queue={NULL,NULL};

while(1)

{

printf("---------------------------------------------------------\n");

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

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

printf("3. Front.\n");

printf("4. Rear.\n");

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

printf("6. Check if the Queue is Empty.\n");

printf("7. Exit.\n");

printf("Enter your choice : \n");

scanf("%d",&ch);

printf("---------------------------------------------------------\n");

switch(ch)

{

case 1 :

printf("Enter the no. to be enqueued : ");

scanf("%d",&v);

enqueue(&queue,v);

break;

case 2 :

p=dequeue(&queue);

if(p!=-9999)

printf("Dequeued element : %d\n",p);

break;

case 3 :

p=front(&queue);

if(p!=-9999)

printf("Front element : %d\n",p);

break;

case 4 :

p=rear(&queue);

if(p!=-9999)

printf("Rear element : %d\n",p);

break;

case 5 :

display(&queue);

break;

case 6 :

if(isEmpty(&queue))

printf("Queue is Empty.\n");

else

printf("Queue is not Empty.\n");

break;

case 7 :

//Freeing memory.

while(!isEmpty(&queue))

dequeue(&queue);

exit(0);

default :

printf("Wrong Choice.\n");

}

}

}

//Header File for some source files.

#ifndef STACK\_H\_INCLUDED

#define STACK\_H\_INCLUDED

typedef struct Stack

{

int top;

int size;

int \*arr;

}Stack;

int isEmpty(Stack \*);

int isFull(Stack \*);

void push(Stack \*,int);

int pop(Stack \*);

int top(Stack \*);

void display(Stack \*);

#endif // STACK\_H\_INCLUDED

//Implementing Queue using Two Stacks (Enqueue & Dequeue).

#include <stdio.h>

#include <stdlib.h>

#include "Stack.h"

typedef struct Queue

{

Stack \*E;//Stack used for Enqueue.

Stack \*D;//Stack used for Dequeue.

int size;

}Queue;

int is\_full(Queue \*queue)

{

if((queue->E->top+queue->D->top)==queue->size-2)//n-2 since we are using two stacks.

return 1;

return 0;

}

int is\_empty(Queue \*queue)

{

if(isEmpty(queue->E)&&isEmpty(queue->D))//both the stacks are empty.

return 1;

return 0;

}

void enqueue(Queue \*queue,int ele)

{

if(is\_full(queue))

printf("Overflow.\n");

else

push(queue->E,ele);//Inserting in Stack E.

}

int dequeue(Queue \*queue)

{

if(is\_empty(queue))

{

printf("Underflow.\n");

return -9999;

}

/\*If Stack D is empty then pop all elements from Stack E and push them into

Stack D and pop the topmost element of Stack D.\*/

if(isEmpty(queue->D))

{

while(!isEmpty(queue->E))

push(queue->D,pop(queue->E));

}

return pop(queue->D);

}

void show(Queue \*queue)

{

if(is\_empty(queue))

printf("Queue is Empty.\n");

else

{

//Printing Stack D from top to bottom and Stack E from bottom to top.

printf("Queue looks like.......\n");

int k;

if(queue->D->top==0&&isEmpty(queue->E))//1 element in queue

printf("%d <----FRONT <----REAR\n",queue->D->arr[0]);

else

if(queue->E->top==0&&isEmpty(queue->D))//1 element in queue

printf("%d <----FRONT <----REAR\n",queue->E->arr[0]);

else

{

if(!isEmpty(queue->D))

{

printf("%d <----FRONT\n",queue->D->arr[queue->D->top]);

for(k=queue->D->top-1;k>=0;k--)

printf("%d\n",queue->D->arr[k]);

printf("%d <----REAR\n",queue->E->arr[queue->E->top]);

k=0;//Updating k for using it later.

}

else

{

printf("%d <----FRONT\n",queue->E->arr[0]);

k=1;//Updating k for using it later.

}

for(;k<queue->E->top;k++)

printf("%d\n",queue->E->arr[k]);

printf("%d <----REAR\n",queue->E->arr[queue->E->top]);

}

}

}

void main()

{

int n,v,p,ch;

Queue queue;

printf("Enter the max size of the queue : ");

scanf("%d",&queue.size);

queue.E->top=queue.D->top=-1;

queue.E->size=queue.D->size=queue.size;

queue.E->arr=(int \*)malloc(queue.size\*sizeof(int));

queue.D->arr=(int \*)malloc(queue.size\*sizeof(int));

while(1)

{

printf("------------------------------------------------------\n");

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

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

printf("3. Display.\n");

printf("4. Check if the Queue is Full.\n");

printf("5. Check if the Queue is Empty.\n");

printf("6. Exit.\n");

printf("Enter your choice : \n");

scanf("%d",&ch);

switch(ch)

printf("------------------------------------------------------\n");

{

case 1 :

printf("Enter the no. to be enqueued : ");

scanf("%d",&v);

enqueue(&queue,v);

break;

case 2 :

p=dequeue(&queue);

if(p!=-9999)

printf("Dequeued element : %d\n",p);

break;

case 3 :

show(&queue);

break;

case 4 :

if(is\_full(&queue))

printf("Queue is Full.\n");

else

printf("Queue is not Full.\n");

break;

case 5 :

if(is\_empty(&queue))

printf("Queue is Empty.\n");

else

printf("Queue is not Empty.\n");

break;

case 6 :

free(queue.E->arr);

free(queue.D->arr);

exit(0);

default :

printf("Wrong Choice.\n");

}

}

}

//Reversing a stack using Recursion.

#include <stdio.h>

#include <stdlib.h>

#include "Stack.h"

Stack \*create\_stack(int n)

{

int ele,k;

Stack \*stack=(Stack \*)malloc(sizeof(\*stack));

stack->top=-1; stack->size=n; stack->arr=(int \*)malloc(n\*sizeof(int));

printf("Push the elements into the stack one by one......\n");

for(k=0;k<n;k++)

{

scanf("%d",&ele);

push(stack,ele);

}

return stack;

}

void push\_at\_bottom(Stack \*stack,int ele)

{

if(isEmpty(stack))

push(stack,ele);

else

{

int temp=pop(stack);

push\_at\_bottom(stack,ele);

push(stack,temp);

}

}

void reverse\_stack(Stack \*stack)

{

if(!isEmpty(stack))

{

int ele=pop(stack);

reverse\_stack(stack);

push\_at\_bottom(stack,ele);

}

}

void main()

{

int n;

printf("Enter the size of the stack : "); scanf("%d",&n);

Stack \*stack=create\_stack(n);

printf("-------------------------------------------------\n");

printf("Initially, "); display(stack);

printf("-------------------------------------------------\n");

reverse\_stack(stack);

printf("After Reversing, "); display(stack);

printf("-------------------------------------------------\n");

}

//Sorting a stack using Recursion.

#include <stdio.h>

#include <stdlib.h>

#include "Stack.h"

Stack \*create\_stack(int n)

{

int ele,k;

Stack \*stack=(Stack \*)malloc(sizeof(\*stack));

stack->top=-1; stack->size=n; stack->arr=(int \*)malloc(n\*sizeof(int));

printf("Push the elements into the stack one by one......\n");

for(k=0;k<n;k++)

{

scanf("%d",&ele);

push(stack,ele);

}

return stack;

}

void push\_at\_correct\_pos(Stack \*stack,int ele)

{

if(isEmpty(stack)||ele<=top(stack))

push(stack,ele);

else

{

int temp=pop(stack);

push\_at\_correct\_pos(stack,ele);

push(stack,temp);

}

}

void sort\_stack(Stack \*stack)

{

if(!isEmpty(stack))

{

int ele=pop(stack);

sort\_stack(stack);

push\_at\_correct\_pos(stack,ele);

}

}

void main()

{

int n;

printf("Enter the size of the stack : "); scanf("%d",&n);

Stack \*stack=create\_stack(n);

printf("---------------------------------------------------------\n");

printf("Initially, "); display(stack);

printf("---------------------------------------------------------\n");

sort\_stack(stack);

printf("After Sorting in Ascending order from TOP, ");display(stack);

printf("---------------------------------------------------------\n");

}

//Expression Evaluator.

#include <ctype.h>

#include <string.h>

#include <stdio.h>

#include "Stack.h"

int precedence(char c)

{

switch(c)

{

case '^' : return 4;

case 'u' : return 3;//special case for unary minus.

case '/' :

case '\*' : return 2;

case '+' :

case '-' : return 1;

default : return 0;

}

}

int isunary(char c)

{

return c=='+'||c=='-'?1:0;

}

int operate(int a,int b,char c)

{

switch(c)

{

case '^' :

return b==0?1:a\*operate(a,b-1,c);

case '/' :

return a/b;

case '\*' :

return a\*b;

case '+' :

return a+b;

case '-' :

return a-b;

default :

return 0;

}

}

void update(Stack \*op\_stack,Stack \*val\_stack)

{

if((char)top(op\_stack)=='u')

{

push(val\_stack,-pop(val\_stack));

pop(op\_stack);

}

else

push(val\_stack,operate(pop(val\_stack),pop(val\_stack),(char)pop(op\_stack)));

}

void evaluate\_till\_open\_parenthesis(Stack \*op\_stack,Stack \*val\_stack)

{

//pop top 2 from value stack and operate and push the result in value stack.

//associativity of arguments of function is L to Rcif it involves function calls.

while((char)top(op\_stack)!='(')

update(op\_stack,val\_stack);

pop(op\_stack);

}

void evaluate\_till\_priority\_exceeds(char c,Stack \*op\_stack,Stack \*val\_stack)

{

//'('s precedence is 0. Hence it will be taken care of.

while((!isEmpty(op\_stack))&&precedence(c)<=precedence((char)top(op\_stack)))

update(op\_stack,val\_stack);

}

int evaluate\_exp(const char \*in)

{

int l=strlen(in);

Stack \*val\_stack=(Stack \*)malloc(sizeof(Stack));

Stack \*op\_stack=(Stack \*)malloc(sizeof(Stack));

op\_stack->top=val\_stack->top=-1;

op\_stack->size=val\_stack->size=l;

op\_stack->arr=(int \*)malloc(l\*sizeof(int));

val\_stack->arr=(int \*)malloc(l\*sizeof(int));

int k,val;

for(k=0;k<l;k++)

{

//Special case for UNARY MINUS.

if(isunary(in[k])&&(k==0||!isdigit(in[k-1])))

push(op\_stack,'u');

if(isdigit(in[k]))

{

val=in[k]-'0';

while(k+1<l&&isdigit(in[k+1]))

val=val\*10+(in[++k]-'0');

push(val\_stack,val);

}

else

if(in[k]=='(')

push(op\_stack,in[k]);

else

if(in[k]==')')

evaluate\_till\_open\_parenthesis(op\_stack,val\_stack);

else

if(precedence(in[k])>precedence((char)top(op\_stack)))

push(op\_stack,in[k]);

else

{

evaluate\_till\_priority\_exceeds(in[k],op\_stack,val\_stack);

push(op\_stack,in[k]);

}

}

while(!isEmpty(op\_stack))

update(op\_stack,val\_stack);

int ans=val\_stack->arr[0];

free(op\_stack->arr);

free(op\_stack);

free(val\_stack->arr);

free(val\_stack);

return ans;

}

int main()

{

char \*in=(char \*)malloc(255\*sizeof(char));

printf("Enter the expression : ");

scanf("%s",in);

printf("Result = %d",evaluate\_exp(in));

free(in);

return 0;

}

// INFIX to POSTFIX Convertor.

#include <ctype.h>

#include <string.h>

#include <stdio.h>

#include "Stack.h"

int precedence(char c)

{

switch(c)

{

case '^' : return 3;

case '/' :

case '\*' : return 2;

case '+' :

case '-' : return 1;

default : return 0;

}

}

void pop\_till\_open\_parenthesis(Stack \*op\_stack,char \*pf,int \*pos)

{

while((char)top(op\_stack)!='(')

pf[(\*pos)++]=(char)pop(op\_stack);

pop(op\_stack);

}

void pop\_till\_priority\_exceeds(char c,Stack \*op\_stack,char \*pf,int \*pos)

{

//'('s precedence is 0. Hence it will be taken care of.

while((!isEmpty(op\_stack))&&precedence(c)<=precedence((char)top(op\_stack)))

pf[(\*pos)++]=(char)pop(op\_stack);

}

char \*toPostfix(const char \*in)

{

int l=strlen(in);

char \*pf=(char \*)malloc(l\*sizeof(\*pf));

Stack \*op\_stack=(Stack \*)malloc(sizeof(Stack));

op\_stack->top=-1;

op\_stack->size=l;

op\_stack->arr=(int \*)malloc(l\*sizeof(int));

int k,pos=0;

for(k=0;k<l;k++)

{

if(isalpha(in[k]))

pf[pos++]=in[k];

else

if(in[k]=='(')

push(op\_stack,in[k]);

else

if(in[k]==')')

pop\_till\_open\_parenthesis(op\_stack,pf,&pos);

else

if(precedence(in[k])>precedence((char)top(op\_stack)))

push(op\_stack,in[k]);

else

{

pop\_till\_priority\_exceeds(in[k],op\_stack,pf,&pos);

push(op\_stack,in[k]);

}

}

while(!isEmpty(op\_stack))

pf[pos++]=(char)pop(op\_stack);

pf[pos]='\0';

free(op\_stack->arr);

free(op\_stack);

return pf;

}

int main()

{

char \*in=(char \*)malloc(255\*sizeof(char));

printf("Enter the expression : ");

scanf("%s",in);

char \*pf=toPostfix(in);

printf("Postfix = %s",pf);

free(in);

free(pf);

return 0;

}

//Deque implementation using Array.

#include <stdio.h>

#include <stdlib.h>

typedef struct Deque

{

int size;

int front;

int rear;

int \*arr;

}Deque;

int isFull(Deque \*deque)

{

if(deque->rear==(deque->front+deque->size))

return 1;

return 0;

}

int isEmpty(Deque \*deque)

{

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

return 1;

return 0;

}

void insertFront(Deque \*deque,int ele)

{

if(isFull(deque))

printf("Overflow.\n");

else

{

if(deque->front==-1) //To avoid -ve front rear complexities.

{

deque->front+=deque->size;

deque->rear+=deque->size;

}

deque->arr[(deque->front--)%deque->size]=ele;

}

}

void insertRear(Deque \*deque,int ele)

{

if(isFull(deque))

printf("Overflow.\n");

else

deque->arr[(++deque->rear)%deque->size]=ele;

}

int deleteFront(Deque \*deque)

{

if(isEmpty(deque))

{

printf("Underflow.\n");

return -9999;

}

return deque->arr[(++deque->front)%deque->size];

}

int deleteRear(Deque \*deque)

{

if(isEmpty(deque))

{

printf("Underflow.\n");

return -9999;

}

return deque->arr[(deque->rear--)%deque->size];

}

int front(Deque \*deque)

{

if(isEmpty(deque))

{

printf("Underflow.\n");

return -9999;

}

return deque->arr[(deque->front+1)%deque->size];

}

int rear(Deque \*deque)

{

if(isEmpty(deque))

{

printf("Underflow.\n");

return -9999;

}

return deque->arr[(deque->rear)%deque->size];

}

void display(Deque \*deque)

{

if(isEmpty(deque))

printf("Deque is Empty.\n");

else

{

printf("Deque looks like.......\n");

if(deque->front+1==deque->rear)

printf("%d <----FRONT <----REAR\n",deque->arr[(deque->front+1)%deque->size]);

else

{

printf("%d <----FRONT\n",deque->arr[(deque->front+1)%deque->size]);

int k;

for(k=deque->front+2;k<deque->rear;k++)

printf("%d\n",deque->arr[k%deque->size]);

printf("%d <----REAR\n",deque->arr[(deque->rear)%deque->size]);

}

}

}

void main()

{

int ch,n,v,p,ipres=0,opres=0;

printf("Enter the max size of the deque : ");

scanf("%d",&n);

Deque deque={n,-1,-1,(int \*)malloc(n\*sizeof(int))};

while(1)

{

printf("------------------------------------------------------\n");

if(ipres)

printf("Input Restriction is ON.\n");

if(opres)

printf("Output Restriction is ON.\n");

if(!ipres)

printf("1. Insert from front.\n");

printf("2. Insert from rear.\n");

printf("3. Delete from front.\n");

if(!opres)

printf("4. Delete from rear.\n");

printf("5. Front.\n");

printf("6. Rear.\n");

printf("7. Display.\n");

printf("8. Check if the Deque is Full.\n");

printf("9. Check if the Deque is Empty.\n");

printf("10. Toggle Input Restriction.\n");

printf("11. Toggle Output Restriction.\n");

printf("12. Exit.\n");

printf("Enter your choice : \n");

scanf("%d",&ch);

printf("------------------------------------------------------\n");

switch(ch)

{

case 1 :

if(ipres) {

printf("Wrong Choice.\n");

break;

}

printf("Enter the no. to be inserted : ");

scanf("%d",&v);

insertFront(&deque,v);

break;

case 2 :

printf("Enter the no. to be inserted : ");

scanf("%d",&v);

insertRear(&deque,v);

break;

case 3 :

p=deleteFront(&deque);

if(p!=-9999)

printf("Deleted element from front : %d\n",p);

break;

case 4 :

if(opres) {

printf("Wrong Choice.\n");

break;

}

p=deleteRear(&deque);

if(p!=-9999)

printf("Deleted element from rear : %d\n",p);

break;

case 5 :

p=front(&deque);

if(p!=-9999)

printf("Front element : %d\n",p);

break;

case 6 :

p=rear(&deque);

if(p!=-9999)

printf("Rear element : %d\n",p);

break;

case 7 :

display(&deque);

break;

case 8 :

if(isFull(&deque))

printf("Deque is Full.\n");

else

printf("Deque is not Full.\n");

break;

case 9 :

if(isEmpty(&deque))

printf("Deque is Empty.\n");

else

printf("Deque is not Empty.\n");

break;

case 10 :

if(ipres)

ipres=0;

else

ipres=1;

break;

case 11 :

if(opres)

opres=0;

else

opres=1;

break;

case 12 :

free(deque.arr);

exit(0);

default :

printf("Wrong Choice.\n");

}

}

}

//Deque implementation using Doubly Linked List.

#include <stdio.h>

#include <stdlib.h>

typedef struct node//It stores contents of each node

{

int data;

struct node \*prev;

struct node \*next;

}Node;

typedef struct Deque

{

Node \*front;

Node \*rear;

}Deque;

int isEmpty(Deque \*deque)

{

if(deque->front==NULL)

return 1;

return 0;

}

void insertFront(Deque \*deque,int ele)

{

Node \*newnode=(Node \*)malloc(sizeof(\*newnode));

if(newnode==NULL)

{

printf("Overflow.\n");

return;

}

newnode->data=ele;

newnode->prev=newnode->next=NULL;

if(deque->front==NULL)

deque->front=deque->rear=newnode;

else

{

deque->front->prev=newnode;

newnode->next=deque->front;

deque->front=newnode;

}

}

void insertRear(Deque \*deque,int ele)

{

Node \*newnode=(Node \*)malloc(sizeof(\*newnode));

if(newnode==NULL)

{

printf("Overflow.\n");

return;

}

newnode->data=ele;

newnode->prev=newnode->next=NULL;

if(deque->front==NULL)

deque->front=deque->rear=newnode;

else

{

newnode->prev=deque->rear;

deque->rear->next=newnode;

deque->rear=deque->rear->next;

}

}

int deleteFront(Deque \*deque)

{

Node \*temp;

if(isEmpty(deque))

{

printf("Underflow.\n");

return -9999;

}

temp=deque->front;

if(deque->front->next==NULL)

deque->front=deque->rear=NULL;

else

{

deque->front=deque->front->next;

deque->front->prev=NULL;

}

int tmp=temp->data;

free(temp);

return tmp;

}

int deleteRear(Deque \*deque)

{

Node \*temp;

if(isEmpty(deque))

{

printf("Underflow.\n");

return -9999;

}

temp=deque->rear;

if(deque->front->next==NULL)

deque->front=deque->rear=NULL;

else

{

deque->rear=deque->rear->prev;

deque->rear->next=NULL;

}

int tmp=temp->data;

free(temp);

return tmp;

}

int front(Deque \*deque)

{

if(isEmpty(deque))

{

printf("Underflow.\n");

return -9999;

}

return deque->front->data;

}

int rear(Deque \*deque)

{

if(isEmpty(deque))

{

printf("Underflow.\n");

return -9999;

}

return deque->rear->data;

}

void display(Deque \*deque)

{

if(isEmpty(deque))

printf("Deque is Empty.\n");

else

{

printf("Deque looks like.......\n");

if(deque->front->next==NULL)

printf("%d <----FRONT <----REAR\n",deque->front->data);

else

{

printf("%d <----FRONT\n",deque->front->data);

Node \*temp=deque->front->next;

while(temp->next!=NULL)

{

printf("%d\n",temp->data);

temp=temp->next;

}

printf("%d <----REAR\n",deque->rear->data);

}

}

}

void main()

{

int ch,p,v,ipres=0,opres=0;

Deque deque={NULL,NULL};

while(1)

{

printf("------------------------------------------------------\n");

if(ipres)

printf("Input Restriction is ON.\n");

if(opres)

printf("Output Restriction is ON.\n");

if(!ipres)

printf("1. Insert from front.\n");

printf("2. Insert from rear.\n");

printf("3. Delete from front.\n");

if(!opres)

printf("4. Delete from rear.\n");

printf("5. Front.\n");

printf("6. Rear.\n");

printf("7. Display.\n");

printf("8. Check if the Deque is Empty.\n");

printf("9. Toggle Input Restriction.\n");

printf("10. Toggle Output Restriction.\n");

printf("11. Exit.\n");

printf("Enter your choice : \n");

scanf("%d",&ch);

printf("------------------------------------------------------\n");

switch(ch)

{

case 1 :

if(ipres) {

printf("Wrong Choice.\n");

break;

}

printf("Enter the no. to be inserted : ");

scanf("%d",&v);

insertFront(&deque,v);

break;

case 2 :

printf("Enter the no. to be inserted : ");

scanf("%d",&v);

insertRear(&deque,v);

break;

case 3 :

p=deleteFront(&deque);

if(p!=-9999)

printf("Deleted element from front : %d\n",p);

break;

case 4 :

if(opres) {

printf("Wrong Choice.\n");

break;

}

p=deleteRear(&deque);

if(p!=-9999)

printf("Deleted element from rear : %d\n",p);

break;

case 5 :

p=front(&deque);

if(p!=-9999)

printf("Front element : %d\n",p);

break;

case 6 :

p=rear(&deque);

if(p!=-9999)

printf("Rear element : %d\n",p);

break;

case 7 :

display(&deque);

break;

case 8 :

if(isEmpty(&deque))

printf("Deque is Empty.\n");

else

printf("Deque is not Empty.\n");

break;

case 9 :

if(ipres)

ipres=0;

else

ipres=1;

break;

case 10 :

if(opres)

opres=0;

else

opres=1;

break;

case 11 :

//Freeing memory.

while(!isEmpty(&deque))

deleteFront(&deque);

exit(0);

default :

printf("Wrong Choice.\n");

}

}

}