**V Venkateswarlu Gupta**

**1602-19-733-119**

**CSE-B**

**Data Structures Lab**

Week – 5

***Prelab Questions***

**1. Design ADT of queue data structure.**

* Queue overflow
* Insertion into a queue
* Queue underflow
* Deletion of element from the queue
* Display the queue

**2. What are the ends through which insertion and deletion are performed in a queue?**

Insertion (enqueue) is performed at rear end.

Deletion (dequeue) is performed at front end.

**3. What condition should be checked when you delete elements from the queue using arrays.**

Is-Empty condition should be checked. front==-1 and rear==-1

**4. Write a condition to check for inserting an element into a queue implemented using linked list.**

node \*front,\*rear,\*temp;

If(front==NULL && rear==NULL){

t=(node \*)malloc(sizeof(node));

temp->data=x;

temp->next=NULL;

front=rear=t;

}else{

t->data=x;

t=rear->next;

t->next=NULL;

rear=temp;

}

**5. List the applications of queues.**

* Customer service centers follow first come first serve.
* Operating System operations
* In recognizing palindromes.

**6.Peudocode to do insertion at ends of queue.**

void Insert(int x){

if(rear==(size-1)) {

write(“Queue is ***FULL***”);

return;

}

if(front==-1 && rear==-1){

front++;

}

arr[++rear]=x;

}

**7.Give step by step procedure to evaluate postfix expression abcd\*-+ using stack.**

* Push a on to the stack.
* Push b onto the stack.
* Push c onto the stack.
* Push d onto the stack.
* Encounters \*, pop c, d outside and perform c\*d and place c\*d onto the stack.
* Encounters, pop c\*d, b outside and perform b-c\*d and place b-c\*d on to the stack.
* Encounters +, pop b-c\*d, a outside and perform a+b-c\*d and place it on to the stack.
* The final value which stack contain is the value of the postfix expression a+b-c\*d.

**Pre Lab Programs**

**Implementing queues using linked list.**

#include<stdio.h>

#include<stdlib.h>

typedef *struct* node

{

*int* data;

*struct* node \*link;

}node;

node \*front=NULL,\*rare=NULL;

*void* enqueue()

{

    node \*t=(node \*)malloc(sizeof(node));

    printf("Enter data : ");

    scanf("%d",&t->data);

    t->link=NULL;

    if(front==NULL&&rare==NULL)

    {

        front=t;

        rare=t;

    }

    else

    {

        rare->link=t;

        rare=t;

    }

}

*void* dequeue()

{

*int* data;

*struct* node \*t;

    if(front==NULL)

        printf("Queue is empty\n");

    data=front->data;

    if(front==rare)

    {

        front=NULL;

        rare=NULL;

    }

    t=front;

    front=front->link;

    free(t);

    printf("Removed from Queue : %d\n",data);

}

*void* display()

{

    node \*t;

    t=front;

    if(front==NULL){

        printf("No elements to display\n");

        return;

    }

    while(t->link!=NULL)

    {

        printf("%d->",t->data);

        t=t->link;

    }

    printf("%d\n",t->data);

}

*void* main()

{

*short* repeat;

    do

    {

*short* choice;

        printf(" 1 enqueue\t 2 dequeue\t 3 display\t # exit : ");

        scanf("%hi",&choice);

        switch(choice)

        {

            case 1:enqueue();

                   break;

            case 2:dequeue();

                   break;

            case 3:display();

                   break;

            default:exit(1);

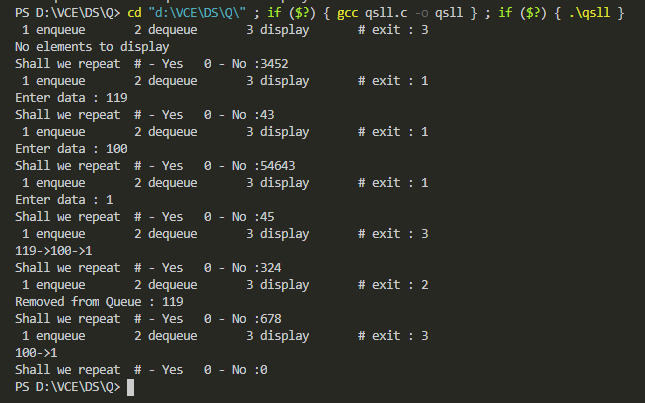
        }

        printf("Shall we repeat\t # - Yes   0 - No :");

        scanf("%hi",&repeat);

    }while(repeat);

}



**Implementing queues using Arrays.**

#include<stdio.h>

#include<stdlib.h>

*int* front=-1;

*int* rear=-1;

#define n 50

*int* a[n];

*void* enqueue(){

*int* x;

    printf("Enter element : ");

    scanf("%d",&x);

    if(rear==n-1){

        printf("Queue is full\n");

    }else if(front==-1 && rear==-1){

        front++;

    }

    a[++rear]=x;

}

*int* dequeue(){

*int* x;

    if(front==-1 && rear==-1){

        printf("No elements to delete\n");

    }else if(front==rear){

        front=-1;

        rear=-1;

    }else{

        x=a[front];

        front++;

        return x;

    }

}

*void* display(){

    printf("Elements in the Q\n");

    if(front==-1){

        printf("Queue is empty\n");

    }else{

        for(*int* i=front;i<=rear;i++){

        printf("%d\t",a[i]);

        }

    }

    printf("\n");

}

*void* main()

{

*int* x;

*short* repeat;

    do

    {

*short* choice;

        printf(" 1 enqueue\t 2 dequeue\t 3 display\t # exit : ");

        scanf("%hi",&choice);

        switch(choice)

        {

            case 1:enqueue();

                   break;

            case 2:x=dequeue();

                   printf("%d is deleted\n",x);

                   break;

            case 3:display();

                   break;

            default:exit(1);

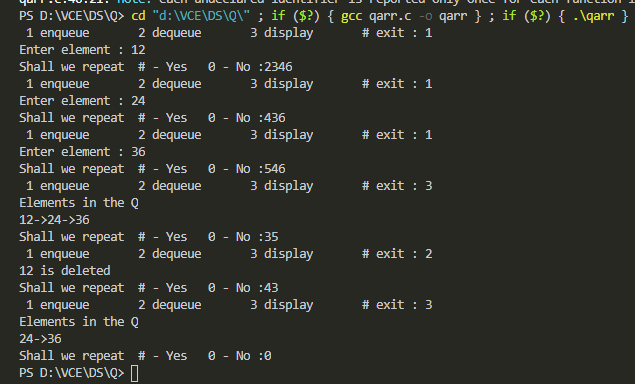
        }

        printf("Shall we repeat\t # - Yes   0 - No :");

        scanf("%hi",&repeat);

    }while(repeat);

}



***Lab Programs:***

**Postfix evaluation**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef *struct* node{

*int* data;

*struct* node \*link;

}node;

node \*h=NULL;

*void* push(*int* *x*){

    node \*t;

    t=(node \*)malloc(sizeof(node \*));

    t->data=*x*;

    t->link=NULL;

    if(h==NULL)

        h=t;

    else{

        t->link=h;

        h=t;

    }

}

*int* pop(){

*int* x;

    node \*t;

    if(h==NULL)

        printf("No elements to pop!\n");

    else if(h->link==NULL){

        x=h->data;

        t=h;

        h=NULL;

    }else{

        x=h->data;

        t=h;

        h=h->link;

    }

    free(t);

    return x;

}

*void* display(){

    node \*q;

    q=h;

    if(q==NULL) return ;

    printf("Displaying\n");

    while(q->link!=NULL){

        printf("%d->",q->data);

        q=q->link;

    }

    printf("%d\n",q->data);

}

*void* main(){

    h=NULL;

*char* exp[128];

    printf("Enter the Postfix Expression : ");

    scanf("%s",exp);

    for(*int* i=0;i<strlen(exp)||h==NULL;i++){

        if(exp[i]=='+'||exp[i]=='-'||exp[i]=='\*'||exp[i]=='/'){

*int* n2,n1;

            n2=pop();

            n1=pop();

            if(exp[i]=='+'){

                push(n1+n2);

            }else if(exp[i]=='-'){

                push(n1-n2);

            }else if(exp[i]=='\*'){

                push(n1\*n2);

            }else if(exp[i]=='/'){

                push(n1/n2);

            }

        }else{

*int* x=exp[i];

            push(x-48);

        }

    }

    printf("Result : %d",pop());

}

Infix to Postfix

#include<string.h>

#include<stdio.h>

#include<ctype.h>

*int* top=-1;

*int* stack[100];

*void* push(*char* *x*){

    if (top>=100){

        printf("Stack is full\n");

    }else{

      stack[++top]=*x*;

    }

}

*char* pop(){

    if(top==-1){

        printf("No elements to pop out\n");

    }else{

        return stack[top--];

    }

}

*int* precedence(*char* *x*){

    if(*x*=='('||*x*==')'){

        return 0;

    }else if(*x*=='+'||*x*=='-'){

        return 1;

    }else if(*x*=='\*'||*x*=='/'){

        return 2;

    }else if(*x*=='^'){

        return 3;

    }

    return -1;

}

*int* main(){

*char* exp[128];

*int* c,size,i;

    printf("Enter the infix expression\n");

    scanf("%s",exp);

*char* d;

    size=strlen(exp);

    while(i<size){

        if(isalnum(exp[i]))

            printf("%c",exp[i]);

        else if(exp[i]=='(')

            push('(');

        else if(exp[i]==')'){

            while((d=pop())!='('){

                printf("%c",d);

            }

        }else{

            while(top!=-1&&precedence(stack[top])>=precedence(exp[i])){

                d=pop();

                printf("%c",d);

            }

            push(exp[i]);

        }

        i++;

    }

    while(top!=-1){

        printf("%c",pop());

    }

}

Paranthesis Count:

#include<stdio.h>

#include<string.h>

*void* main(){

*char* ch[128];

*int* sz,open=0,close=0;

    printf("Enter the Expression \n");

    scanf("%s",ch);

    sz=strlen(ch);

    for(*int* i=0;i<sz;i++){

        if(ch[i]=='('){

            open++;

        }else if(ch[i]==')'){

            close++;

        }

    }

    if(open==close){

        printf("Matched\n");

    }else{

        printf("Not Matched\n");

    }

}