**LAB 8**

Q.1 **Implemet a circular queue by writing a menu drivenn program with function like**

* **INSERTION**
* **DELETION**
* **DISPLAY**

#include <stdio.h>

#define MAX 5

int cqueue\_arr[MAX];

int front = -1;

int rear = -1;

void insert(int item)

{

if ((front == 0 && rear == MAX - 1) || (front == rear + 1))

{

printf("Queue Overflow ");

return;

}

if (front == -1)

{

front = 0;

rear = 0;

}

else

{

if (rear == MAX - 1)

rear = 0;

else

rear = rear + 1;

}

cqueue\_arr[rear] = item;

}

void deletion()

{

if (front == -1)

{

printf("Queue Underflow");

return;

}

printf("Element deleted from queue is : %d \n", cqueue\_arr[front]);

if (front == rear)

{

front = -1;

rear = -1;

}

else

{

if (front == MAX - 1)

front = 0;

else

front = front + 1;

}

}

void display()

{

int front\_pos = front, rear\_pos = rear;

if (front == -1)

{

printf("Queue is empty");

return;

}

printf("Queue elements : \n");

if (front\_pos <= rear\_pos)

while (front\_pos <= rear\_pos)

{

printf("%d ", cqueue\_arr[front\_pos]);

front\_pos++;

}

else

{

while (front\_pos <= MAX - 1)

{

printf("%d ", cqueue\_arr[front\_pos]);

front\_pos++;

}

front\_pos = 0;

while (front\_pos <= rear\_pos)

{

printf("%d ", cqueue\_arr[front\_pos]);

front\_pos++;

}

}

printf("\n");

}

int main()

{

int choice, item;

do

{

printf("\n1.Insertion");

printf("\n2.Deletion");

printf("\n3.Display");

printf("\n4.Quit");

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

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Input the element for insertion in queue : \n");

scanf("%d", &item);

insert(item);

break;

case 2:

deletion();

break;

case 3:

display();

break;

case 4:

break;

default:

printf("Wrong choice");

}

} while (choice != 4);

return 0;}

**OUTPUT**

1.Insertion

2.Deletion

3.Display

4.Quit

Enter your choice :

1

Input the element for insertion in queue :

2

1.Insertion

2.Deletion

3.Display

4.Quit

Enter your choice :

1

Input the element for insertion in queue :

3

1.Insertion

2.Deletion

3.Display

4.Quit

Enter your choice :

1

Input the element for insertion in queue :

4

1.Insertion

2.Deletion

3.Display

4.Quit

Enter your choice :

3

Queue elements :

2 3 4

1.Insertion

2.Deletion

3.Display

4.Quit

Enter your choice :

4

Q2.**Implement a Queue ADT**

#include <stdio.h>

#include <stdlib.h>

struct Queue

{

int q[100];

int front, rear;

};

void initialise(struct Queue \*q)

{

q->front = -1;

q->rear = -1;

}

void insert(struct Queue \*q1, int n)

{

if ((q1->front == 0 && q1->rear == 99) || (q1->front == q1->rear + 1))

{

printf("Overflow\n");

return;

}

else

{

q1->front = 0;

q1->rear = ((q1->rear) + 1) % 100;

q1->q[q1->rear] = n;

}

}

int delete (struct Queue \*q1, int \*n)

{

if ((q1->front == -1 && q1->rear == -1))

{

return 0;

}

else if (q1->front == q1->rear)

{

q1->front = -1;

q1->rear = -1;

printf("The list is empty\n");

}

else

{

n = &q1->q[q1->front];

q1->front = ((q1->front + 1)) % 100;

}

return \*n;

}

void display(struct Queue \*q1)

{

int i;

if ((q1->front == q1->rear) || (q1->front == -1 && q1->rear == -1))

{

printf("The list is empty\n");

}

else

{

i = q1->front;

do

{

printf("%d", q1->q[i]);

i = (i + 1) % 100;

} while (i != ((q1->rear) + 1));

}

}

int main()

{

struct Queue q;

int n, ch, val, prev;

printf("Enter 1 to Insert\n");

printf("Enter 2 to Delete\n");

printf("Enter 3 to display\n");

printf("Enter 4 to exit\n");

initialise(&q);

while (1)

{

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

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter size of Queue :\n");

scanf("%d", &n);

printf("Enter the elements :\n");

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

{

scanf("%d", &val);

insert(&q, val);

}

printf("The elements are: \n");

display(&q);

break;

case 2:

prev = delete (&q, &prev);

if (prev != 0)

{

printf("The deleted item : %d\n", prev);

}

else

printf("Underflow!!\n");

break;

case 3:

printf("The elements are: \n");

display(&q);

break;

case 4:

printf("Exiting...\n");

exit(0);

default:

printf("Wrong choice!!\n");

continue;

}

}

return 0;

}

**OUTPUT**

Enter 1 to Insert

Enter 2 to Delete

Enter 3 to display

Enter 4 to exit

Enter your choice :

1

Enter size of Queue :

4

Enter the elements :

2

3

4

5

The elements are:

2345

Enter your choice :

2

The deleted item : 2

Enter your choice :

3

The elements are:

345

Enter your choice :

4

Exiting…

**Q3.//WAP to evaluate a given postfix expression**

#include <stdio.h>

#include <ctype.h>

int stack[20];

int top = -1;

void push(int x)

{

stack[++top] = x;

}

int pop()

{

return stack[top--];

}

int main()

{

char exp[40];

char \*e;

int n1, n2, n3, num;

printf("Enter the expression : ");

scanf("%s", &exp);

e = exp;

while (\*e != '\0')

{

if (isdigit(\*e))

{

num = \*e - 48;

push(num);

}

else

{

n1 = pop();

n2 = pop();

switch (\*e)

{

case '+':

{

n3 = n1 + n2;

break;

}

case '-':

{

n3 = n2 - n1;

break;

}

case '\*':

{

n3 = n1 \* n2;

break;

}

case '/':

{

n3 = n2 / n1;

break;

}

}

push(n3);

}

e++;

}

printf("\n The result of expression %s = %d \n \n", exp, pop());

return 0;

}

**OUTPUT**

Enter the expression : 254+\*

The result of expression 254+\* = 18

Q4.//**Write a program to covert a given infix expression to its equivalent postfix expression**

#include<stdio.h>

#include<ctype.h>

char stack[100];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

return 0;

}

int main()

{

char exp[100];

char \*e, x;

printf("Enter the expression : ");

scanf("%s",exp);

printf("\n");

e = exp;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c ",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c ", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c ",pop());

push(\*e);

}

e++;

}

while(top != -1)

{

printf("%c ",pop());

}return 0;

}

**OUTPUT**

Enter the expression : (2+3)\*(6-5)\*(8\*5)

2 3 + 6 5 - \* 8 5 \* \*

Q5.**//Wap to implement two stack to an array to minimaze overflow in the stack..**

#include <stdio.h>

#define SIZE 20

int array[SIZE];

int top1 = -1;

int top2 = SIZE;

void push1(int data)

{

if (top1 < top2 - 1)

{

top1++;

array[top1] = data;

}

else

{

printf("Stack is full");

}

}

void push2(int data)

{

if (top1 < top2 - 1)

{

top2--;

array[top2] = data;

}

else

{

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

}

}

void pop1()

{

if (top1 >= 0)

{

int popped\_element = array[top1];

top1--;

printf("%d is being popped from Stack 1\n", popped\_element);

}

else

{

printf("Stack is Empty \n");

}

}

void pop2()

{

if (top2 < SIZE)

{

int popped\_element = array[top2];

top2--;

printf("%d is being popped from Stack 1\n", popped\_element);

}

else

{

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

}

}

void display\_stack1()

{

int i;

for (i = top1; i >= 0; --i)

{

printf("%d ", array[i]);

}

printf("\n");

}

void display\_stack2()

{

int i;

for (i = top2; i < SIZE; ++i)

{

printf("%d ", array[i]);

}

printf("\n");

}

int main()

{

int ar[SIZE];

int i;

int num\_of\_ele;

printf("We can push a total of 20 values\n");

for (i = 1; i <= 10; ++i)

{

push1(i);

printf("Pushed in Stack 1 is %d\n", i);

}

for (i = 11; i <= 20; ++i)

{

push2(i);

printf("Pushed in Stack 2 is %d\n", i);

}

display\_stack1();

display\_stack2();

printf("Pushing Value in Stack 1 is %d\n", 11);

push1(11);

num\_of\_ele = top1 + 1;

while (num\_of\_ele)

{

pop1();

--num\_of\_ele;

}

pop1();

return 0;

}

**OUTPUT**

**We can push a total of 20 values**

**Pushed in Stack 1 is 1**

**Pushed in Stack 1 is 2**

**Pushed in Stack 1 is 3**

**Pushed in Stack 1 is 4**

**Pushed in Stack 1 is 5**

**Pushed in Stack 1 is 6**

**Pushed in Stack 1 is 7**

**Pushed in Stack 1 is 8**

**Pushed in Stack 1 is 9**

**Pushed in Stack 1 is 10**

**Pushed in Stack 2 is 11**

**Pushed in Stack 2 is 12**

**Pushed in Stack 2 is 13**

**Pushed in Stack 2 is 14**

**Pushed in Stack 2 is 15**

**Pushed in Stack 2 is 16**

**Pushed in Stack 2 is 17**

**Pushed in Stack 2 is 18**

**Pushed in Stack 2 is 19**

**Pushed in Stack 2 is 20**

**10 9 8 7 6 5 4 3 2 1**

**20 19 18 17 16 15 14 13 12 11**

**Pushing Value in Stack 1 is 11**

**Stack is full10 is being popped from Stack 1**

**9 is being popped from Stack 1**

**8 is being popped from Stack 1**

**7 is being popped from Stack 1**

**6 is being popped from Stack 1**

**5 is being popped from Stack 1**

**4 is being popped from Stack 1**

**3 is being popped from Stack 1**

**2 is being popped from Stack 1**

**1 is being popped from Stack 1**

**Stack is Empty**

RISHIKESH

2105734

CSE32