PRACTICE 4

PRACTICE 4.1.1

AIM: Write a C program on stack and its operations using arrays

PROGRAM:

```
#include<stdio.h>
int stack[5],i,top=-1,ele;
void push()
{
        if(top==4)
        printf("stack is overflow");
        else
        {
                top++;
                printf("enter an element");
                scanf("%d",&ele);
                stack[top]=ele;
        }
}
void pop()
{
        if(top==-1)
        printf("Stack is undeflow");
        else
        {
                ele=stack[top];
                printf("Deleted element is %d",ele);
                top--;
        }
}
void display()
{
        for(i=top;i>=0;i--)
        printf("%d\n",stack[i]);
}
int main()
{
        while(1)
        {
                int ch;
                printf("\n1.push 2.pop 3.display 4.exit\n");
                printf("enter your choice");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:push();
                        break;
                        case 2:pop();
                        break;
                        case 3:display();
                        break;
                        case 4:exit(0);
                }
        }
}
```

OUTPUT:

```
1.push 2.pop 3.display 4.exit
enter your choice1
enter an element10

1.push 2.pop 3.display 4.exit
enter your choice1
enter an element20

1.push 2.pop 3.display 4.exit
enter your choice1
enter an element30

1.push 2.pop 3.display 4.exit
enter your choice3
30
20
10

1.push 2.pop 3.display 4.exit
enter your choice2
Deleted element is 30
1.push 2.pop 3.display 4.exit
enter your choice4
```

PRACTICE 4.1.2

AIM: Write a C program on stack and its operations using LinkedList

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *next;
};
struct node *newnode=NULL;
struct node *top=NULL;
struct node *temp=NULL;
void push();
void pop();
void display();
int ele;
void main()
{
 int ch;
 while(1)
{
 printf("\nstack operations\n");
 printf("1.push 2.pop 3.display 4.exit\n");
 printf("Enter your choice\n");
 scanf("%d",&ch);
 switch(ch)
 {
 case 1:push();
 break;
 case 2:pop();
 break;
 case 3:display();
 break;
 default: exit(0);
 break;
 }
 }
}
void push()
{
 newnode=(struct node*)malloc(sizeof(struct node));
 printf("Enter data\n");
 scanf("%d",&ele);
 if(top==NULL)
 {
 newnode->data=ele;
 newnode->next=NULL;
 top=newnode;
 }
 else
 {
 newnode->data=ele;
 newnode->next=top;
 top=newnode;
 }
}
void pop()
{
 if(top==NULL)
 printf("Stack is empty");
 else
 {
 temp=top;
 ele=top->data;
 printf("Deleted element is %d\n",ele);
 top=top->next;
 temp->next=NULL;
 free(temp);
 }
}
void display()
{
 if(top==NULL)
 printf("Stack is empty");
 else
 {
```

```
 printf("The stack elements are:\n");
 temp=top;
 while(temp!=NULL)
 {
 printf("%d->", temp->data);
 temp=temp->next;
 }
 printf("NULL");
 }
}
```

OUTPUT:

```
stack operations
1.push 2.pop 3.display 4.exit
Enter your choice
1
Enter data
10

stack operations
1.push 2.pop 3.display 4.exit
Enter your choice
1
Enter data
20

stack operations
```

```
1.push 2.pop 3.display 4.exit
Enter your choice
1
Enter data
30

stack operations
1.push 2.pop 3.display 4.exit
Enter your choice
3
The stack elements are:
30->20->10->NULL
stack operations
1.push 2.pop 3.display 4.exit
Enter your choice
2
Deleted element is 30

stack operations
1.push 2.pop 3.display 4.exit
Enter your choice
4
```

PRACTICE 4.2

AIM: Write a C program to evaluate a postfix expression

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#define MAX 20
void push(int);
int pop();
int evaluationofpostfix(char*);
int stack[MAX], top = -1;
void push(int n)
{
   stack[++top] = n;
}
int pop()
{
   return stack[top--];
}
int evaluationofpostfix(char *ex)
{
   int d = 0, s = 0, i;
   for (i = 0; ex[i] != '\0'; i++)
        {
      if (isdigit(ex[i]))
         d++;
        else if (ex[i] == '+' || ex[i] == '-' || ex[i] == '*' || ex[i] == '/' || ex[i] == '%'
|| ex[i] == '^')

            s++;

    }

    if (d <= s)

        {

        printf("Invalid Expression\n");

        exit(0);

    }


for (i = 0; ex[i] != '\0'; i++)

{

if(isdigit(ex[i]) || ex[i] == '+' || ex[i] == '-' || ex[i] == '*' || ex[i] == '/' || ex[i] ==
'%') {

            if (isdigit(ex[i]))

                push(ex[i] - '0');

            else {

                int x1, x2;

                x1 = pop();

                x2 = pop();


                switch (ex[i]) {

                    case '+': push(x2 + x1); break;

                    case '-': push(x2 - x1); break;

                    case '*': push(x2 * x1); break;

                    case '/': push(x2 / x1); break;

                    case '%': push(x2 % x1); break;
                }
            }
        } else {
            printf("Invalid expression\n");
            exit(0);
        }
    }

    if (top != 0) {
        printf("Invalid expression\n");
        exit(0);
    } else {
        return stack[top];
    }
}

int main()
 {
    char exp[MAX];
    printf("Enter a valid postfix expression: ");
    gets(exp);
    printf("Result of postfix evaluation is %d\n", evaluationofpostfix(exp));
    return 0;
}
```

OUTPUT:

```
Enter a valid postfix expression: 234+*
Result of postfix evaluation is 14
```

PRACTICE 4.3.1

AIM: Write a C program on Queue and its operations using arrays

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int queue[SIZE], f = -1, r = -1, ele;
void enq();
void deq();
void display();
int main()
{
    int ch;
    while (1)
          {
        printf("\n1.Enq 2.Deq 3.Display 4.Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch (ch)
                    {
            case 1: enq(); break;
            case 2: deq(); break;
            case 3: display(); break;
            case 4: exit(0);
            default: printf("Invalid choice\n");
        }
    }
    return 0;
}

void enq()
{
    if (r == SIZE - 1)
          {
        printf("Queue is Full\n");
    }
           else
          {
        printf("Enter an element: ");
        scanf("%d", &ele);
        if (f == -1) f = 0;  // Only set f=0 for the first insertion
        queue[++r] = ele;
    }

}

void deq()
{
    if (f == -1 || f > r)
          {
        printf("Queue is Empty\n");
    }
          else
          {
        ele = queue[f];
        f++;
        printf("Deleted Element is %d\n", ele);
        if (f > r) f = r = -1; // Reset after last element is removed
    }
}

void display()
 {
          int i;
    if (f == -1 || f > r)
          {
        printf("Queue is Empty\n");
    } else
          {
        printf("Queue elements: ");
        for(i = f; i <= r; i++)
            printf("%d ", queue[i]);
        printf("\n");
    }
```

```
}
```

OUTPUT:

```
1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 1
Enter an element: 10

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 1
Enter an element: 20

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 1
Enter an element: 30

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 3
Queue elements: 10 20 30

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 2
Deleted Element is 10

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 4
```

PRACTICE 4.3.2

AIM: Write a C program on Queue and its operations using LinkedList

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node* next;
};
struct Node *front = NULL;
struct Node *rear = NULL;
void enq();
void deq();
void display();
int main()
{
   int ch;
   while (1)
        {
      printf("\n1.Enq 2.Deq 3.Display 4.Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &ch);
      switch (ch)
               {
         case 1: enq(); break;
```

```
            case 2: deq(); break;
            case 3: display(); break;
            case 4: exit(0);
            default: printf("Invalid choice\n");
        }
    }
    return 0;
}

void enq() {
    int ele;
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory not allocated\n");
        return;
    }
    printf("Enter an element: ");
    scanf("%d", &ele);
    newNode->data = ele;
    newNode->next = NULL;

    if (rear == NULL)
          {
        front = rear = newNode;
    }
          else
          {
        rear->next = newNode;
        rear = newNode;
    }
}

void deq() {
    if (front == NULL)
          {
        printf("Queue is Empty\n");
        return;
    }
    struct Node *temp = front;
    printf("Deleted Element is %d\n", front->data);
    front = front->next;
    free(temp);

    if (front == NULL)
        rear = NULL;
}

void display()
{
    struct Node* temp = front;
    if (front == NULL) {
        printf("Queue is Empty\n");
        return;
    }

    printf("Queue elements: ");
    while (temp != NULL)
        {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

OUTPUT:

```
1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 1
Enter an element: 10

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 1
Enter an element: 20

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 1
Enter an element: 30

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 3
```

```
Queue elements: 10 20 30

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 2
Deleted Element is 10

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 4
```

PRACTICE 4.4

AIM: Write a C program on Circular Queue and its operations using arrays

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

int queue[SIZE], f = -1, r = -1, ele;

void enq();
void deq();
void display();

int main() {
    int ch;
    while (1) {
        printf("\n1.Enq 2.Deq 3.Display 4.Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1: enq(); break;
            case 2: deq(); break;
            case 3: display(); break;
            case 4: exit(0);
            default: printf("Invalid choice\n");
        }
    }
    return 0;
}

void enq() {
    // Check if the queue is full
    if ((f == 0 && r == SIZE - 1) || (r + 1 == f)) {
        printf("Queue is Full\n");
    } else {
        printf("Enter an element: ");
        scanf("%d", &ele);

        if (f == -1) {
            f = r = 0;
        } else {
            r++;
            if (r > SIZE - 1)
                r = 0;
        }
        queue[r] = ele;
    }
}

void deq() {
    if (f == -1) {
        printf("Queue is Empty\n");
    } else {
        ele = queue[f];
        printf("Deleted Element is %d\n", ele);
        if (f == r) {
            // Only one element was there
            f = r = -1;
        } else {
            f++;
            if (f > SIZE - 1)
                f = 0;
        }
    }
}

void display() {
    if (f == -1) {
        printf("Queue is Empty\n");
    } else {
        printf("Queue elements: ");
        int i = f;
        while (1) {
            printf("%d ", queue[i]);
            if (i == r)
                break;
            i++;
            if (i > SIZE - 1)
                i = 0;
        }
        printf("\n");
    }
}
```

OUTPUT:

```
1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 1
Enter an element: 10

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 1
Enter an element: 20

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 1
Enter an element: 30

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 1
Enter an element: 40

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 1
Enter an element: 50

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 3
Queue elements: 10 20 30 40 50

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 2
Deleted Element is 10

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 1
Enter an element: 60

1.Enq 2.Deq 3.Display 4.Exit
Enter your choice: 3
Queue elements: 20 30 40 50 60
```

```
1.Enq 2.Deq 3.Display 4.Exit

Enter your choice:4
```