01/01/24

Lab-3

Q2. Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operator +(plus), -(minus), *(multiply), / (divide) and ^(power).

⇒ Input -

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
char stack[100];
int top=-1, size;
void push(char item)
{
  if (top >= size-1)
  {
    printf("\n Stack overflow.");
  }
  else
  {
    top = top+1;
    stack[top] = item;
  }
}
```

```
char pop()
{
  char item;
  if (top <0)
  {
    printf ("\n Stack Underflow \n");
  }
  else
  {
    item = stack [top];
    top = top -1;
    return (item);
  }
}
int is_operator (char symbol)
{
  if (symbol == '^' || symbol == '*' ||
      symbol == '/' || symbol == '+' ||
      symbol == '-')
  {
    return 1;
  }
  else
  { return 0; } }
```

```
int precendence (char symbol)
{
 if (symbol == '^')
 {
  return (3);
 }
 else if (symbol == '*' || symbol == '/')
 {
  return (2);
 }
 else if (symbol == '+' || symbol == '-')
 {
  return (1);
 }
 else
 {
  return (0);
 }
}
void InfixToPostfix (char infix_exp[], char postfix_
                     exp[])
{
 int i, j;
 char item;
```

```
char x;
push('(');
strcat(infix_exp, ")");
i=0;
j=0;
item=infix_exp[i];
while(item!='\0')
{
if(item=='(')
{
   push(item);
}
else if(isdigit(item)||isalpha(item))
{
   postfix_exp[j]=item;
   j++;
}
else if(is_operator(item)==1)
{
   x=pop();
   while(is_operator(x)==1&& precedence(x)
   >=precedence(item))
{
```

```
        postfix_exp[j] = x;
        j++;
        x = pop();
    }
    push(x);
    push(item);
}
else if (item == ')')
{
    x = pop();
    while x != '('
    {
        postfix_exp[j] = x;
        j++;
        x = pop();
    }
}
else
{
    printf("\n Invalid infix Expression.\n");
    exit(1);
}
i++;
item = infix_exp[i];
```

```
    postfix_exp[j]='\0';
}
main()
{
    char infix[100], postfix[100];
    printf("\n Enter size of stack");
    scanf("%d", &size);
    printf("Assume the infix expression contains
            single letter variables and single
            digit constants only.\n");
printf("\n Enter Infix Expression: ");
    scanf("%s", infix);
    InfixToPostfix(infix, postfix);
    printf("Postfix Expression:");
    printf("%s", postfix);
}
```

Output-

Enter size of Stack 10

Assume the infix expression contains single letter variables and single digit constants only.

Enter Infix Expression: a/f-p+d*y12

Postfix Expression: af/p-dy2^*+

Q3. Write a program to simulate the working of a queue of integers using an array. Provide the following operations.

a) Insert
b) Delete
c) Display

The program should print appropriate messages for queue empty and queue overflow conditions.

⇒ Input –

```
#include <stdio.h>
#include <conio.h>
#define MAX 10
int queue[MAX];
int front = -1, rear = -1;
void insert (void);
int delete_element (void);
int peek (void);
void display (void);
int main()
{
   int option, val;
   do
   {
      printf ("\n\n ***** MAIN MENU *****");
```

```
printf("\n 1. Insert an element");
printf("\n 2. Delete an element");
printf("\n 3. Peek");
printf("\n 4. Display the queue");
printf("\n 5. Exit");
printf("\n Enter your option : ");
scanf("%d", &option);
switch(option)
{
 Case 1:
      insert();
      break;
 Case 2:
      val = delete_element();
      if (val != -1)
      printf("\n The number deleted is : %d",
                  val);
      break;
 Case 3:
      val = peek();
      if (val != -1)
          printf("\n The first value in
                  queue is %d", val);
 break;
```

```
        case 4:
                display();
                break;
        }
    }while(option!=5);
    getch();
    return 0;
}
void insert()
{
    int num;
    printf("\n Enter the number to be inserted
            in the queue:");
    scanf("%d", &num);
    if (rear = MAX-1)
        printf("\n OVERFLOW");
    else if (front == -1 && rear == -1)
        front = rear = 0;
    else
        rear++;
    queue[rear] = num;
}
int delete_element()
{
```

```
int val;
if (front == -1 || front > rear)
{
  printf ("\n UNDERFLOW");
  return -1;
}
else
{
  val = queue[front];
  front++;
  if (front > rear)
    front = rear = -1;
  return val;
}
}
int peek()
{
  if (front == -1 || front > rear)
  {
    printf ("\n Queue is Empty");
    return -1;
  }
  else
  { return queue[front]; } }
```

```
void display()
{
 int i;
 printf("\n");
 if (front==-1 || front>rear)
    printf("\n QUEUE IS Empty");
 else
 {
    for (i=front; i<=rear; i++)
        printf("\t %d", queue[i]);
 }
}
```

Output-

```
*** MAIN MENU ***
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. Exit
Enter your Option: 1
Enter the number to be inserted in the queue: 7
Enter your option: 2
The number deleted is : 7
Enter your Option: 4
 QUEUE IS EMPTY
```

Q3b. Write a program to simulate the working of a circular queue of integers using an array. Provide the following operations.

a) Insert
b) Delete
c) Display

The program should print appropriate messages for queue empty and queue overflow conditions.

=> Input -

```
#include <stdio.h>
#include <conio.h>
#define MAX 10
int queue [MAX];
int front = -1, rear = -1;
void insert (void);
int delete_element (void);
int peek (void);
void display (void);
int main()
{
  int option, val;
  clrscr();
  do
    {
```

```
printf("\n ***** MAIN MENU *****");
printf("\n 1. Insert an element");
printf("\n 2. Delete an element");
printf("\n 3: Peek");
printf("\n 4. Display the queue");
printf("\n 5. EXIT");
printf("\n Enter your option: ");
scanf("%d", &option);
switch(option)
{
  Case 1:
      insert();
      break;
  Case 2:
      val = delete_element();
      if (val != -1)
      printf("\n The number deleted is %d", val);
      break;
  Case 3:
      val = peek();
      if (val != -1):
          printf("\n The first value is: %d", val);
      break;
```

```
        case 4:
                display();
                break;
        }
} while (option!=5);
  getch();
  return 0;
}
void insert()
{
  int num;
  printf("\n Enter the number to be inserted in
          the queue:");
  scanf("%d", &num);
  if (front==0 && rear==MAX-1)
      printf("\n OVERFLOW");
  else if (front==-1 && rear==-1)
  {
    front=rear=0;
    queue[rear]=num;
  }
  else if (rear==Max-1 && front!=0)
  {
    rear=0; queue[rear]=num;}
```

```
    else
    {
        rear++;
        queue[rear] = num;
    }
}
int delete_element()
{
    int val;
    if (front == -1 && rear == -1)
    {
        printf("\n UNDERFLOW");
        return -1;
    }
    val = queue[front];
    if (front == rear)
        front = rear = -1;
    else
    {
        if front == MAX-1)
            front = 0;
        else
            front++;
    } return val; }
```

```
int peek()
{
  if (front == -1 && rear == -1)
  {
    printf ("\n Queue is Empty");
    return -1;
  }
  else
  {
    return queue[front];
  }
}
Void display()
{
  int i;
  printf ("\n");
  if (front == -1 && rear == -1)
    printf ("\n Queue is empty");
  else
  {
    if (front < rear)
    {
      for (i = front; i <= rear; i++)
        printf ("\t %d", Queue[i]);
    }
```

```
        else
        {
            for(i=front; i<Max; i++)
                printf("\t%d", queue[i]);
            for(i=0; i<=rear; i++)
                printf("\t%d", queue[i]);
        }
    }
}
```

Output -

```
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. Exit
Enter your option: 1
Enter the number to be inserted: 25
Enter your option: 2
The number deleted is: 25
Enter your option: 3
 Queue is empty
Enter your option: 5
```

03/01/2022