Assignment 04

1. Define 'stack'.

Ans. A stack is a non primitive linear data structure. It is an ordered list in which add and delete operations are done only from one end, known as top of stack (TOS). A stack is also called a Last-in-First-out (LIFO) list is the last element is the first one in the list to be taken out from it.

2. Describe the applications of stack.

Ans. There are three applications in stack.

[i] Infix notation : Operation between operands as A+B
[ii] Prefix notation : Operation between operands as +AB
[iii] Postfix notation : Operation between operands as AB+

3. Write a program to convert the infix expression to postfix expression.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 20
char stack[20];
int top = -1;
int isEmply()
{
   return top == -1;
}
int isFull()
{
   return top == MAX-1;
}
char peek()
{
```

```
    return stk[top];
}
char pop()
{
    if ( isEmpty())
        return -1;
    char ch = stk[top];
    top--;
    return (ch);
}
void push (char oper)
{
    if (isFull())
        printf ("Stack Full !!");
    else
    {
        top++;
        stk[top] = oper;
    }
}
int checkIfOperand (char ch)
{
    return (ch>='a' && ch<='z') || (ch>='A' && ch<='Z');
}
int precedence (char ch)
{
    switch (ch)
    {
      case '+':
      case '-':
          return 1;
```

```
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
    }
    return -1;
}
int convertInfixToPostfix(char *exp)
{
    int i, j;
    for (i=0, j=-1; expression[i]; i++)
    {
        if (checkIfOperand(expression[i]))
            expression[j++] = expression[i];
        else if(expression[i] == '(')
            push(expression[i]);
        else if (expression[i] == ')')
        {
            while (!isEmpty() && peek() != '(')
                expression[j++] = pop();
            if (!isEmpty() && peek() != '(')
                return -1;
            else
                pop();
        }
        else
        {
            while (!isEmpty() && precedence(expression[i])
                <= precedence(peek()))
                expression[j++] = pop();
            push(expression[i]);
```

```
    }
  }
  while (!isEmpty())
      expression[j++] = pop();
  expression[j++] = '\0';
  printf("%s", expression);
}
int main()
{
  char expression[] = "((p+(q*r))-s)";
  convertInfixToPostfix(expression);
  return 0;
}
```

4. Write a program to evaluate the postfix expression.

Program:

```
#include <stdio.h>
int stack[20];
int top = -1;
void push(int x)
{
   stack[++top] = x;
}
int pop()
{
   return stack[top--];
}
int main()
{
   char exp[20];
   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;
            }
            cas '/':
            {
```

```
            n3 = n2 / n1;
            break;
        }
    }
    push(n3);
}
i++;
}
printf("\n The result of expression %s = %d\n\n", exp, pop());
return 0;
}
```

[b]

5. Convert the following infix expression to postfix form:

[a] $(((a+b)/c)+((d-e)*f))\#$

Symbol	Stack	Postfix Array
initial	#	—
(	#(	—
(	#((	—
(	#(((	—
a	#(((	a
+	#(((+	a
b	#(((+	ab
)	#((	ab+
/	#((/	ab+
c	#((/	ab+c
)	#(	ab+c/
+	#(+	ab+c/
(	#(+(	ab+c/
(	#(+((	ab+c/

a	# (+((	a b + c / d
-	# (+(-	a b + c / d
e	# (+((	a b + c / d e
)	# (+(	a b + c / d e -
*	# (+(*	a b + c / d e -
f	# (+(*	a b + c / d e - f
)	# (+	a b + c / d e - f *
)	#	a b + c / d e - f * +

Ans

[b] (x + (y % z) - p / (q * r)) #

Symbol	Stack	Postfix Array
initial	#	—
(	# (	—
x	# (	x
+	# (+	x
(	# (+(	x
y	# (+(	x y
%	# (+(%	x y
z	# (+(%	x y z
)	# (+	x y z %
-	# (-	x y z % +
p	# (-	x y z % + p
/	# (-/	x y z % + p
(	# (-/(	x y z % + p
q	# (-/(	x y z % + p q
*	# (-/(*	x y z % + p q
r	# (-/(*	x y z % + p q r
)	# (-/	x y z % + p q r *
)	#	x y z % + p q r * / -

6. Evaluate the following expression using stack

[a] 6, 7, 12, +, 70, 4, *, -, / #

Symbol	Stack
6	6
7	6, 7
12	6, 7, 12
+	6, 19
70	6, 19, 70
4	6, 19, 70, 4
*	6, 19, 280
-	6, -261
/	0

[b] 25, 35, -, 20, 10, +, 30, +, *

Symbol	Stack
25	25
35	25, 35
-	-10
20	-10, 20
10	-10, 20, 10
+	-10, 30
30	-10, 30, 30
+	-10, 60
*	-600

[c] True, False, &&, True, True, &&, || #

Symbol	Stack
True	True
False	True, False
&&	False
True	False, True
True	False, True, True
&&	False, True
||	True

Ans

7. Convert the following infix expression to prefix form
a + b % c - d * e + f * g - h / i

Ans. Reverse: i / h - g * f + e * d - c % b + a #

Symbol	Stack	~~Stack~~ Postfix Array
initial	#	-
i	#	i
/	# /	i
h	# /	ih
-	# -	ih/
g	# -	ih/g
*	# - *	ih/g
f	# - *	ih/gf
+	# +	ih/gf*-
e	# +	ih/gf*-e
*	# + *	ih/gf*-e
d	# + *	ih/gf*e
-	# -	ih/gf*-ed
c	# -	ih/gf*-ed*+
%	# - %	ih/gf*-ed*+c
b	# - %	ih/gf*-ed*+c
+	# +	ih/gf*-ed*+cb
a	# +	ih/gf*-ed*+cb%-
		ih/gf*-ed*+cb%-a

| #= | ih/gf*-ed*+cb*-a+

Reverse: +a-*bc+*de-*fg/hi Ans

8. Consider a stack of size 6. Push 10, 15, 18 and 12. Then perform the following operation and show the operation diagrammatically in stack:

[a] Push 7

```
# define MAX 6
int stack [MAX];
int top = -1
void push (int item)
{
   int item = 7;
   if (top == MAX-1)
   {
      printf ("Stack overflow");
      return;
   }
   top++;
   stack [top] = item;
}
```

[b] Pop

```
# define MAX 6
int stack [MAX];
   int top = -1;
   int pop (int *s, int top)
   {
      int temp = s[top];
      if (top == MAX-1)
      {
         printf ("Stack overflow");
      }
```

```
    else
    {
      top--;
    }
    return (top);
}
```

[c] Pop

```
#define MAX 6
int stack [MAX];
int top = -1;
int pop (int *s, int top)
{
    int temp = s[top]
    if (top == MAX-1)
    { printf ("Stack underflow")
    }
    else
    {
      top--;
    }
    return (top);
}
```

[d] pop

```
#define MAX 6
int stack [MAX];
int top = -1;
int pop (int *s, int top)
{
    int temp = s[top];
    if (top == MAX-1)
    { printf ("Stack underflow")
    }
    else
    {
      top--;
    }
    return (top);
}
```

[e] Pop

```
#define MAX 6
int stack[MAX];
int top = -1;
int pop(int *s, int top)
{
    int temp = s[top];
    if (top == MAX-1)
    {
        printf("Stack Underflow");
    }
    else
    {
        top--;
    }
    return(top);
}
```

[f] Push 9

```
#define MAX 6
int stack[MAX];
int top = -1;
void push(int item)
{
int item = 9;
if(top == MAX-1)
{
    printf("Stack Overflow");
    return;
}
top++
stack[top] = item;
}
```

[g]

[h]

[g] Push 8

```
#define MAX 6
int stack[MAX];
int top=-1;
void push(int item)
{
    int item = 8;
    if (top == MAX-1)
    {
        printf("Stack Overflow");
        return;
    }
    top++;
    stack[top] = item;
}
```

[h] Pop

```
#define MAX 6
int stack[MAX];
int top = -1;
int pop(int *s, int top)
{
    int temp = s[top];
    if (top == MAX-1)
    {
        printf("Stack Underflow");
    }
    else
    {
        top--;
    }
    return(top);
}
```

Assignment - 05

1.

```
# include <stdio.h>
void fun(int *);
int main()
{
   int i = 10, *p = &i;
   fun(p++);
}
void fun(int *p)
{
   printf("%d\n", *p)
}
```

Ans. [a] 10

2.

```
# include<stdio.h>
int main()
{
 int i = 97, *p = &i;
 fun(&p);
 printf("%d", *p);
 return 0;
}
void fun(int **p)
{ int j = 2;
 *p = &j;
 printf("%d", **p);
}
```

ns. [a] 2 2

```
# include <stdio.h>
int main()
{ int i = 11;
 int *p = &i;
```

```
    fun(&p);
    printf("%d",*p);
}
void fun(int *const *b)
{
    int j = 10;
    *b = &j;
    printf("%d",**b);
}
```

Ans - [a] compile time error

4. What is output?

```
int * fun()
{
    int A=10;
    return(&A);
}
int main()
{
    int *p;
    p = fun();
    printf("%p int", p);
    printf("%d\n", *p);
    return 0;
}
```

Ans - Segmentation fault

5. What is the output

```
int * fun()
{
    static int A = 10;
    return(&A);
}
int main()
{
    int *p;
    p = fun();
```

```
    printf("%p\n", p);
    printf("%d\n", d);
    return 0;
}
```

Ans - 10

6. What is the output?

```
#include <stdio.h>
void print(int *arr, int m, int n)
{
    int i, j;
    for(i=0; i<m; i++)
    {
        for(j=0; j<n; j++)
        {
            printf("%d", *((arr + i*n)+j));
        }
    }
}
int main()
{
    int arr[][3]={{1,2,3},{4,5,6},{7,8,9}};
    int m=3, n=3;
    print((int*)arr, m, n);
    return 0;
}
```

Ans. 1 2 3 4 5 6 7 8 9

7. What is the output?

```
#include <stdio.h>
int main()
{
    int arr[5]={1,2,3,4,5};
    int p, q, r;
    p = ++arr[1];
    q = arr[1]++;
```

```
r = arr[p++];
printf("%d,%d,%d", p,q,r);
return 0;
}
```

Ans - 4, 3, 4

8.
```
# include <stdio.h>
void main()
{
  char c[] = "GATE 2011";
  char *p = c;
  printf("%s", p+p[3]-p[1]);
}
```

Ans. [C] 2011

9.
```
# include <stdio.h>
int main()
{
  char str[10] = "98765", *p;
  p = str+1;
  *p = '0';
  printf("%s", str);
}
```

Ans. [D] 90765

10.
```
# include <stdio.h>
int main()
{char s1[] = "Hello";
 char s2[] = "Hello";
 if (s1 = s2)
  printf("same");
 else
  printf("Not same");
 return 0;
}
```

Ans - Same

11. Let us suppose that a queue can store maximum seven elements with the following data.
front = 3, rare = 5
queue ::
What will happen after ADD Q operation take place.

Ans - [a] front = 3, rare = 6
queue :: —, —, —, R, S, P, Q, —

12. Let us suppose that a circular queue can store maximum element with the following data
front = 4, rare = 6
circular queue :: —, —, —, —, R, S, P
What will happen after ADD Q and ADD T operations take place respectively?

Ans - [c] fron = 4, rare = 1
circular queue :: Q, T, —, —, R, S, P

13. If the elements 'L', 'M', 'N' and 'O' are added in a queue and are removed one by one, then in which order will they be deleted?

Ans. [a] L M N O

14. Define the following terms:

[i] Queue :- A queue is an abstract data type. It is a non primitive linear data structure. It is an ordered list in which elements are added at the rare end and deleted only from the front end.

[ii] Circular queue :- A circular queue is a non-primitive linear data structure which follows the first in first out principle

[iii] Deque :- A double ended queue the elements can be added or deleted from both the ends. There are two Types of Deques:

[a] Input Restricted Deque
[b] Output Restricted Deque

15. Write a menu driven program of a queue with add, delete and display functionalities.

```
# include <stdio.h>
# define MAX 20
# includ <stdlib.h>
int queue_array[MAX];
int rear = -1;
int front = -1;
void insert()
{
   int add item;
   if (rear == MAX-1)
      printf("Queue overflow");
   else
   {
      if (front == -1)
         front = 0;
      printf("\n Enter element:");
      scanf("%d", &add_item);
      printf("\n %d is inserted in queue\n", add_item);
      rear = rear + 1;
      queue_array[rear] = add_item;
   }
}
void delete()
{
   if (front == -1 || front > rear)
   {
      printf("Queue Overflow\n");
      return;
   }
```

```
    else
    {
        printf ("\n Element deleted from queue is: %d\n",
                queue_array[front]);
        front= front +1;
    }
}
void display ()
{
    int i;
    if (front == -1)
        printf ("Queue is empty");
    else
    {
        printf ("\n Queue is: ");
        for (i = front; i <= rear; i++)
            printf("%d", queue_array[i];
        printf ("\n");
    }
}
int main
{
    printf (" Perform Operations on queue \n");
    printf ("\t Menu");
    printf ("1. Insert element \n");
    printf ("2. Delete element \n");
    printf ("3. Display queue \n");
    printf ("4. Exit \n");
    int ch;
    while(1)
    {
        printf ("choose operation: ");
        scanf ("%d", &ch);
```

```
switch(ch){
{
  case 1:
      insert();
      break;
  case 2:
      delete();
      break;
  case 3:
      display();
      break;
  case 4:
      exit(1)
  default:
      printf("Invalid operation \n");
  }
}
return 0;
}
```

16. Write a menu driven program of a circular queue with add, delete and display functionalities.

```
# 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;
    }
    squeue_arr[rear]=item;
}
void deletion()
{
    if (front==-1)
    {
        printf("Queue underflown");
        return;
    }
    printf("Element deletefrom queue is :%d", squeue_arr
                                        [front]);
    if(front==rear)
    {
        front=-1;
        rear=-1;
    }
    else
    {
        if(front==MAX-1)
        front=0;
        else
        front=front+1;
    }
}
```