Q.6 WAP to convert an infix expression to a postfix expression using stacks.

```
#include<iostream>
#include <cctype>
#include<stack>
#include<string>
using namespace std;
//for checking operator precedence
int precedence (char c){
  if(c=='+'||c=='-') return 1;
  if(c=='*'||c=='/') return 2;
  if(c=='^') return 3;
  return 0;
}
//for checking given char is an operator or not
bool isOperator(char c){
return (c=='+'||c=='-'||c=='*'||c=='/'||c=='^');
}
int main(){
  stack<char> st; //stack creation
  string infix,postfix="";
  cout << "enter the infix expression : ";
  getline(cin,infix);  //taking expression input
  cout << "entered expression : "<< infix <<endl;
  for(int i=0;i<infix.length();i++){
     char c=infix.at(i);
     if(isalnum(c)){
      postfix+=c;   //if char is alphanumeric
     }
```

```
    if(c=='('){
     st.push(c);       //if given char  is  (
    }else if(c==')'){      //if given char is )
     while (!st.empty()&& st.top()!='('){     //until top is (  keep adding to post fix and removing from top;
     postfix+=st.top();
     st.pop();
}
     st.pop();      //remove ( from stack
}
else if(isOperator(c)){
     //if given char is operator
   while(!st.empty()&& precedence(st.top())>=precedence(c)){
    //if operator precendence < that of stack top add stack top to prefix and remove from stack
    postfix+=st.top();
   st.pop();}
   st.push(c); // al last add to stack
 }}
while(!st.empty()){
  postfix+=st.top(); // until stack is empty add to postfix and pop from stack
  st.pop();
}
cout << "after conversion from infix :"<<postfix<<endl;
  return 0;
}
```

OUTPUT :-

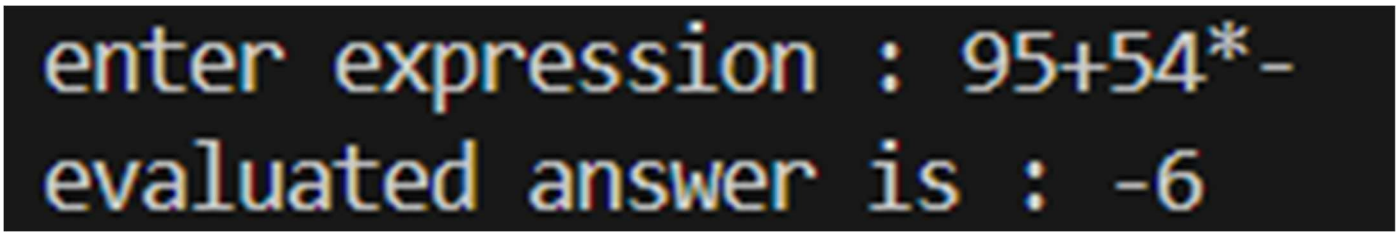
```
enter the infix expression : (a+b)-(c*d)
entered expression : (a+b)-(c*d)
after conversion from infix :ab+cd*-
```

Q.7 WAP to evaluate a postfix expression using stacks.

```
#include<iostream>
#include<string>
#include<stack>
#include<cmath>
using namespace std;
int main(){
  cout << "enter postfix expression : ";
string exp;
getline(cin,exp); // input of postfix expression
  stack<int> st;
  for(int i=0;i<exp.length();i++){
    char c=exp.at(i);
    if(isdigit(c)){
      st.push(c-'0') ;  //if is num then covert it to int and push to satck
    }
    else{
      int val2=st.top();st.pop();   //is operator take val2 ,then val1   then operate val1 to val2 and push for more operations
      int val1=st.top();st.pop();
        switch(c){
          case '+':
          st.push(val1+val2);
          break;
          case '-':
          st.push(val1-val2);
          break;
          case '/':
          st.push(val1/val2);
```

```
        break;
        case '*':
        st.push(val1*val2);
        break;
        case '^':
        st.push(pow(val1,val2));
        break;
       }
   }
 }
 cout << "evaluated answer is : "<<st.top();   //output of final result
 return 0;
}
```

OUTPUT:-



Q.8 Write a recursive function for Tower of Hanoi problem.

```
#include<iostream>

using namespace std;

void towerOfHanoi(int n,char source,char helper,char destination){

  if(n==1){

    cout << "move disk from "<<source<< " to " <<destination<<endl;

    return ;

  }

  towerOfHanoi(n-1,source ,destination,helper);

  cout << "move disk form "<<source <<" to "<<destination<<endl;

  towerOfHanoi(n-1,helper,source,destination);

}

int main(){

  int x;

  cout << "enter no. of disk : ";

  cin>>x;

  towerOfHanoi(x,'A','B','C');


  return 0;

}
```

OUTPUT:-

```
enter no. of disk : 3
move disk from A to C
move disk form A to B
move disk from C to B
move disk form A to C
move disk from B to A
move disk form B to C
move disk from A to C
```

Q.9 WAP to implement insertion and deletion operation in a queue using linear array

```
#include <iostream>
#define SIZE 5 // Define the size of the queue.
using namespace std;
class Queue {
private:
  int arr[SIZE];
  int front, rear;
public:
  Queue() : front(-1), rear(-1) {}
  // Function to insert an element into the queue
  void enqueue(int value) {
    if (rear == SIZE - 1) {
      cout << "Queue Overflow! Cannot insert " << value <<  endl;
     return;
    }
    if (front == -1) {
     front = 0; // Set front to 0 if inserting the first element  }
    arr[++rear] = value;
     cout << "Inserted " << value << " into the queue." <<  endl;  }
  // Function to delete an element from the queue
  void dequeue() {
    if (front == -1 || front > rear) {
      cout << "Queue Underflow! No elements to delete." <<  endl;
     return;}
     cout << "Deleted " << arr[front++] << " from the queue." <<  endl;
    if (front > rear) {
     front = rear = -1; // Reset the queue if it becomes empty }}
```

```
  // Function to display the elements of the queue
  void display() {
    if (front == -1 || front > rear) {
       cout << "Queue is empty." <<  endl;
      return; }
     cout << "Queue elements: ";
    for (int i = front; i <= rear; i++) {
       cout << arr[i] << " ";}
   cout <<  endl;}};
int main() {
  Queue q;
  int choice, value;
  do {
     cout << "\nQueue Operations:\n";
     cout << "1. Enqueue\n";
     cout << "2. Dequeue\n";
     cout << "3. Display\n";
     cout << "4. Exit\n";
     cout << "Enter your choice: ";
     cin >> choice;
    switch (choice) {
    case 1:
       cout << "Enter the value to insert: ";
       cin >> value;
      q.enqueue(value);
      break;
    case 2:
      q.dequeue();
      break;
```

```
    case 3:
      q.display();
      break;
    case 4:
       cout << "Exiting..." <<  endl;
      break;
    default:
       cout << "Invalid choice! Please try again." <<  endl;
    }
  } while (choice != 4);
  return 0;
}
```

OUTPUT:-

```
Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 5
Inserted 5 into the queue.

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 4
Inserted 4 into the queue.

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 5 4

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 5 from the queue.
```

```
Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 4

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting...
```

Q.10 WAP a menu driven program to perform following insertion operation in a linked list :

i. insertion at beginning

ii. insertion at end

iii. insertion after a given node

iv. traversing a linked list

```
#include<iostream>
using namespace std;
class node{
   public :
   int data;
   node *next;
   node(int n){
      data=n;
      next=NULL;}
};

class LinkedList{
   private :
   node *head,*tail;
   public :
   LinkedList(){
      tail=head=NULL;
   }
   void pushFront(int a){
      node *newNode=new node(a);
      if(head==NULL){
         head=newNode;
```

```
        tail=newNode;
        return;
    }
    newNode->next=head;
    head=newNode;
}
void pushEnd(int x){
    node *newNode=new node(x);
    if(tail==NULL){
        head=tail=newNode;
        return;
    }
    tail->next=newNode;
    tail=newNode;
}
void pushAt(int ind,int val){
    node *temp=head;
    if(ind==0){
        cout << "invalid postion : must be greater than or equal 1"<<endl;
    }
    if(ind==1){
        pushFront(val);
        return;
    }
    for(int i=1;i<ind-1;i++){
        if(temp==NULL){
            cout << "invalid position "<<endl;
            break;
        }
```

```
        temp=temp->next;
      };
      if(temp!=NULL){
        node *newNode=new node(val);
        newNode->next=temp->next;
        temp->next=newNode;
      }
    }
    void printLL(){
      node *temp=head;
      if(head==NULL){
        cout << "empty linked list"<<endl;
        return;
      }
      while(temp!=NULL){
        cout<<temp->data<<" -> ";
        temp=temp->next;
      }
      cout<<" null"<<endl;
    }
};

int main(){
    LinkedList ll;
    int choice,value,position;
    while(1){
      cout << "1. to insert at beginning "<<endl;
      cout << "2. to insert at end"<<endl;
      cout << "3. to insert at a given node"<<endl;
```

```
cout << "4. to traverse through the ll"<<endl;
cout << "5. to exit"<<endl;
cin >>choice;

switch(choice){
   case 1:
   cout <<"enter value to push at beginning "<<endl;
   cin >>value;
   ll.pushFront(value);
   break;

   case 2:
   cout <<"enter value to push at end "<<endl;
   cin >>value;
   ll.pushFront(value);
   break;
   case 3 :
   cout <<"enter position to push at  "<<endl;
   cin >>position;
   cout <<"enter value to push at end "<<endl;
   cin >>value;
   ll.pushAt(position,value);
   break;
   case 4:
   ll.printLL();
   break;
   case 5:
   return 0;
   break;
```

```
        default :
        cout <<"invalid selection"<<endl;
      }
   }
   return 0;
}
```

OUTPUT:

```
1. to insert at beginning
2. to insert at end
3. to insert at a given node
4. to traverse through the ll
5. to exit
1
enter value to push at beginning
5
1. to insert at beginning
2. to insert at end
3. to insert at a given node
4. to traverse through the ll
5. to exit
4
5 ->  null
1. to insert at beginning
2. to insert at end
3. to insert at a given node
4. to traverse through the ll
5. to exit
2
enter value to push at end
8
1. to insert at beginning
2. to insert at end
3. to insert at a given node
4. to traverse through the ll
5. to exit
4
8 -> 5 ->  null
1. to insert at beginning
2. to insert at end
3. to insert at a given node
4. to traverse through the ll
5. to exit
3
enter position to push at
2
enter value to push at end
6
```

```
1. to insert at beginning
2. to insert at end
3. to insert at a given node
4. to traverse through the ll
5. to exit
4
8 -> 6 -> 5 ->  null
1. to insert at beginning
2. to insert at end
3. to insert at a given node
4. to traverse through the ll
5. to exit
5
```

Q.10 SWAP a menu driven program to perform following insertion operation in a linked list :

i. delete from beginning

ii. delete from end

iii. delete after a given node

iv. traversing a linked list

```
#include<iostream>
using namespace std;
class node{
   public :
   int data;
   node *next;
   node(int n){
      data=n;
      next=NULL;
   }
};

class LinkedList{
   private :
   node *head,*tail;
   public :
   LinkedList(){
      tail=head=NULL;
   }
   void pushFront(int a){
      node *newNode=new node(a);
```

```
    if(head==NULL){
        head=newNode;
        tail=newNode;
        return;
    }
    newNode->next=head;
    head=newNode;
}
void pushEnd(int x){
    node *newNode=new node(x);
    if(tail==NULL){
        head=tail=newNode;
        return;
    }
    tail->next=newNode;
    tail=newNode;
}

void printLL(){
    node *temp=head;
    if(head==NULL){
        cout << "empty linked list"<<endl;
        return;
    }
    while(temp!=NULL){
        cout<<temp->data<<" -> ";
        temp=temp->next;
    }
    cout<<" null"<<endl;
```

```
}
void popFront(){
    if(head==NULL){
        cout <<"empty linked list"<<endl;
        return;
    }
    node *temp=head;
    head=temp->next;
    cout << "deleted "<<temp->data<<endl;
    delete temp;
}
void popEnd(){
    if(tail==NULL){
        cout << "empty linked list"<<endl;
        return;
    }
    node *temp=head;
    while(temp->next->next!=NULL){
        temp=temp->next;
    }
    cout <<"deleted "<<temp->next->data<<endl;
    delete temp->next;
    temp->next=NULL;
}
void popAt(int position){
    node *temp=head;
    if(head==NULL){
        cout << "empty linked list"<<endl;
        return;
```

```
        }
        if(position==1){
            popFront();
            return ;
        }
        for(int i=1;i<position-1;i++){
            if(temp->next==NULL){
                cout << "invalid position"<<endl;
            }
            temp=temp->next;

        }
        if(temp!=NULL){
            node *superTemp=temp->next;
           temp->next =temp->next->next;
           cout << "deleted "<<superTemp->data<<endl;
           delete superTemp;

        }
    }
};

int main(){
    LinkedList ll;
    ll.pushFront(1);
    ll.pushFront(2);
    ll.pushFront(3);
     ll.pushFront(4);
    ll.pushFront(5);
```

```
ll.pushFront(6);
 ll.pushFront(7);
ll.pushFront(8);
ll.pushFront(9);

int choice,position;
while(1){
   cout << "1. delete from beginning "<<endl;
   cout << "2. to delete from end"<<endl;
   cout << "3. to delete after a given node"<<endl;
   cout << "4. to traverse through the ll"<<endl;
   cout << "5. to exit"<<endl;
   cin >>choice;

   switch(choice){
      case 1:

      ll.popFront();
      break;

      case 2:
      ll.popEnd();
      break;
      case 3 :
      cout <<"enter position to pop "<<endl;
      cin >>position;
      ll.popAt(position);
      break;
      case 4:
```

```
        ll.printLL();
        break;
        case 5:
        return 0;
        break;
        default :
        cout <<"invalid selection"<<endl;
      }
   }
   return 0;
}
```

OUTPUT:

```
1. delete from beginning
2. to delete from end
3. to delete after a given node
4. to traverse through the ll
5. to exit
4
9 -> 8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 ->  null
1. delete from beginning
2. to delete from end
3. to delete after a given node
4. to traverse through the ll
5. to exit
1
deleted 9
1. delete from beginning
2. to delete from end
3. to delete after a given node
4. to traverse through the ll
5. to exit
4
8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 ->  null
1. delete from beginning
2. to delete from end
3. to delete after a given node
4. to traverse through the ll
5. to exit
2
deleted 1
1. delete from beginning
2. to delete from end
3. to delete after a given node
4. to traverse through the ll
5. to exit
4
8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 ->  null
```

```
1. delete from beginning
2. to delete from end
3. to delete after a given node
4. to traverse through the ll
5. to exit
3
enter position to pop
5
deleted 4
1. delete from beginning
2. to delete from end
3. to delete after a given node
4. to traverse through the ll
5. to exit
4
8 -> 7 -> 6 -> 5 -> 3 -> 2 ->  null
1. delete from beginning
2. to delete from end
3. to delete after a given node
4. to traverse through the ll
5. to exit
5
PS D:\DSA\share>
```

Q.12 WAP to implement push pop operation in stack using liked list

```
#include<iostream>
using namespace std;
class node{
   public :
   int data;
   node *next;
   node(int n){
      data=n;
      next=NULL;
   }
};
node *head=NULL;
 void push(int a){
      node *newNode=new node(a);
      if(head==NULL){
         head=newNode;
         return;
      }
      newNode->next=head;
      head=newNode;
   }
    void pop(){
      if(head==NULL){
         cout <<"empty Stack list"<<endl;
         return;
      }
      node *temp=head;
      head=temp->next;
```

```
        cout << "deleted "<<temp->data<<endl;
        delete temp;
    }

     void printStack(){
        node *temp=head;
        if(head==NULL){
            cout << "empty linked list"<<endl;
            return;
        }
        cout << "printing stack"<<endl;
        while(temp!=NULL){
            cout<<temp->data<<endl;
            temp=temp->next;
        }
    }
int main(){
    push(10);
    push(20);
    push(50);
    printStack();
    pop();
    printStack();
return 0;
}
```

OUTPUT:

```
printing stack
50
20
10
deleted 50
printing stack
20
10
```