(STACK PROGRAMS)

(ALI AKBER BSCS 3RD SS1)

PUSH AND POP.

```
#include <iostream>

using namespace std;

const int MAX=7;

class OrderArray{

        private:

        int arr[MAX];

        int count;

        public:

        OrderArray(){

                count=0;

        }

        int size(){

                return count;

        }

                void insertion(int val){

        int i,j;

        for(i=0;i<count;i++){

                if(arr[i]>val)

                break;

        }

        for(j=count;j>i;j--){
```

```
        arr[j]=arr[j-1];
}
  arr[i]=val;
        count++;
}
int operator[](int index) {
        if (index >= 0 && index < count) {
                return arr[index];
        } else {
                return -1;
        }
}

  int search(int sval){
  int lower;
  int upper;
  int loc;
  while(true){
        loc=(lower+upper)/2;
        if (arr[loc]==sval)
        return loc;
        else if(lower>upper)
        return count;
        else {
```

```
                    if(arr[loc]<sval)
                lower=loc+1;
                else
                upper=loc-1;
                }
            }
        }

            bool deletion(int val){
          int i=search(val);
          if(i==count){
          return false;
        } else{
          for(int j=i;j<count;j++){
                arr[j]=arr[j+1];
                count--;
                return true;
          }
        }
};
};

int main(){
        OrderArray oa;
```

```
    oa.insertion(3);
      oa.insertion(4);
        oa.insertion(2);

            for (int i = 0; i < oa.size(); i++) {
            cout << oa[i] << " ";
    }
    cout << endl;
          if(oa.search(30)!=oa.size())
            cout << "SEARCH SUCCESSFUL! NUMBER FOUND"<<endl;
            else
                    cout<<"SEARCH UNSUCCESSFUL! NUMBER NOT FOUND";

                    cout<<endl;

                      if(oa.deletion(3))
                    cout<<"VALUE DELETED SUCCESSFULLY"<<endl;
                    else
                      cout<<"VALUE DELETE UNSUCCESSFULL"<<endl;
  return 0;
}
```

INFIX TO POSTFIX.

```
#include <iostream>
```

```
#include <string>

using namespace std;

const int MAX=20;

class Stack{

        private:

        char items[MAX];

        int top;

        public:

        Stack(){

                top=-1;

        }

        bool isempty(){

                if(top==-1){

                        return true;

                } else{

                        return false;

                }

        }

        char Stacktop(){

                return items[top];

        }

        void push(char ch){

                if(top==MAX-1){

                        cout<<"Overflow"<<endl;

                        exit(1);
                } else{
                        items[++top]=ch;
                }
        }
        char pop(){
                    if(top==-1){
                        cout<<"Underflow"<<endl;
                          exit(1);
                    }
                    return items[top--];
        }
        bool precedence(char top,char symb)
        {
                if(top=='(' || symb=='(')
                return false;
                if(symb==')')
                return true;
                  if(symb=='$')
                  return false;
                if(top =='$')
                  return true;
                  if((symb=='*' || symb=='/') && (top=='*' || top=='/'))
                  return true;
                    if((symb=='+' || symb=='-') && (top=='+' || top=='-'))
                  return true;
```

```
                else{
                      return false;
                }
          }
};
int main(){
          Stack stk;
          string infix,postfix;
          int i;
          cout<<"Enter infix expression"<<endl;
          cin>>infix;

          for(i=0;i<infix.length();i++){
                char symb=infix[i];
                if(symb>='A' && symb<='Z')
                postfix.append(1,symb);
                else{
                      while(!stk.isempty() && stk.precedence(stk.Stacktop(),symb)){
                            char topsymb=stk.pop();
                            postfix.append(1,topsymb);
                      }      if(stk.isempty() || symb!=')')
                      stk.push(symb);
                      else{
                            stk.pop();
                      }
```

```
        }
    }
    while (!stk.isempty()){
        char topsymb=stk.pop();
        postfix.append(1,topsymb);
    }
    cout<<"Postfix = "<<postfix<<endl;
}
```

EVALUATION OF POSTFIX.

```
#include <iostream>
#include <math.h>
using namespace std;
const int MAX=20;
class Stack{
    private:
    int items[MAX];
    int top;
    public:
    Stack(){
        top=-1;
    }
    void push(int val){
        if(top==MAX-1){
            cout<<"Overflow"<<endl;
            exit(1);
        } else{
                items[++top]=val;
        }
}
int pop(){
            if(top==-1){
                cout<<"Underflow"<<endl;
                  exit(1);
            }
            return items[top--];
}

    int calculate(int op1,int op2,char opt){
        switch(opt){
                case '+':
                return op1+op2;
                break;
                  case '-':
                return op1-op2;
                break;
                  case '*':
                return op1*op2;
                break;
                  case '/':
                return op1/op2;
```

```
                        break;
                          case '$':
                        return pow(double(op1),(op2));
                        break;
                        default:
                        cout<<"Invalid Option"<<endl;
                }
        }
};

int main(){
        Stack stk;
        string postfix;
        int i;
        int op1,op2,r;
        cout<<"Enter postfix expression"<<endl;
        cin>>postfix;

        for(i=0;i<postfix.length();i++){
          char symb=postfix[i];
          if(symb>='0'&& symb<='9'){
                stk.push(symb-'0');
          } else{
                op2=stk.pop();
                  op1=stk.pop();
                r=stk.calculate(op1,op2,symb);

                stk.push(r);

            }

        }

        cout<<"Value is "<<stk.pop()<<endl;


      return 0;

}
```

INFIX TO PREFIX.

```
//INFIX TO PREFIX

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

const int MAX = 20;

class Stack {

private:

        char items[MAX];

        int top;

public:

        Stack() {

                top = -1;

        }

        bool isEmpty() {

                return top == -1;
        }

        char stackTop() {

                return items[top];

        }

        void push(char ch) {

                if (top == MAX - 1) {

                        cout << "Overflow" << endl;

                        exit(1);

                } else {

                        items[++top] = ch;

                }

        }

        char pop() {

                if (top == -1) {

                        cout << "Underflow" << endl;

                        exit(1);

                }

                return items[top--];

        }


        bool precedence(char top, char symb) {

                if (top == ')' || symb == ')')

                        return false;

                if (symb == '(')

                        return true;
```

```
            if (symb == '$')

                  return false;

            if (top == '$')

                  return true;

            if ((symb == '*' || symb == '/') && (top == '*' || top == '/'))

                  return true;

            if ((symb == '+' || symb == '-') && (top == '+' || top == '-'))

                  return true;

            else {

                  return false;

            }

      }

};

string infixToPrefix(string infix) {

      reverse(infix.begin(), infix.end());

      string prefix = "";

      Stack stk;


      for (int i = 0; i < infix.length(); i++) {

            char symb = infix[i];


            if ((symb >= 'A' && symb <= 'Z') || (symb >= 'a' && symb <= 'z') || (symb >= '0' && symb <=
'9')) {

                  prefix += symb;

            } else {

                  while (!stk.isEmpty() && stk.precedence(stk.stackTop(), symb) && stk.stackTop() != '(') {
```

```
                char topsymb = stk.pop();

                prefix += topsymb;

            }

            if (stk.isEmpty() || symb != '(') {

                stk.push(symb);

            } else {

                stk.pop();

            }

        }

    }

    while (!stk.isEmpty()) {

        char topsymb = stk.pop();

        prefix += topsymb;

    }

    reverse(prefix.begin(), prefix.end());

    return prefix;

}

int main() {

    string infix, prefix;

    cout << "Enter infix expression: ";

    cin >> infix;

    prefix = infixToPrefix(infix);

    cout << "Prefix = " << prefix << endl;

    return 0;

}
```

```
//Enter infix expression:     (a-b/c)*(a/k-l)

//Prefix = */-abc-/akl
```