VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Fall Semester 2020-2021

CSE2002- Theory of Computation & Compiler Design

Digital Assignment-1

Name: Arshdeep Singh Bhatia

Reg no: 19BCB0086

Contents

Question addressed

Write a program (C/Python) to implement a simple desk calculator using operator precedence parsing.

Important Instructions:

(1) Program should be dynamic.

(2) Make your assumptions and mention them with submission.

(3) Construct the grammar and provide it with your submission.

(4) Output of the program should contain:

(a) Precedence table (showing the precedence of only those operators (not all), which are present in given input string). Example: if the input expression is 2+3*4-5, which is needs to be parse than the precedence table should be:

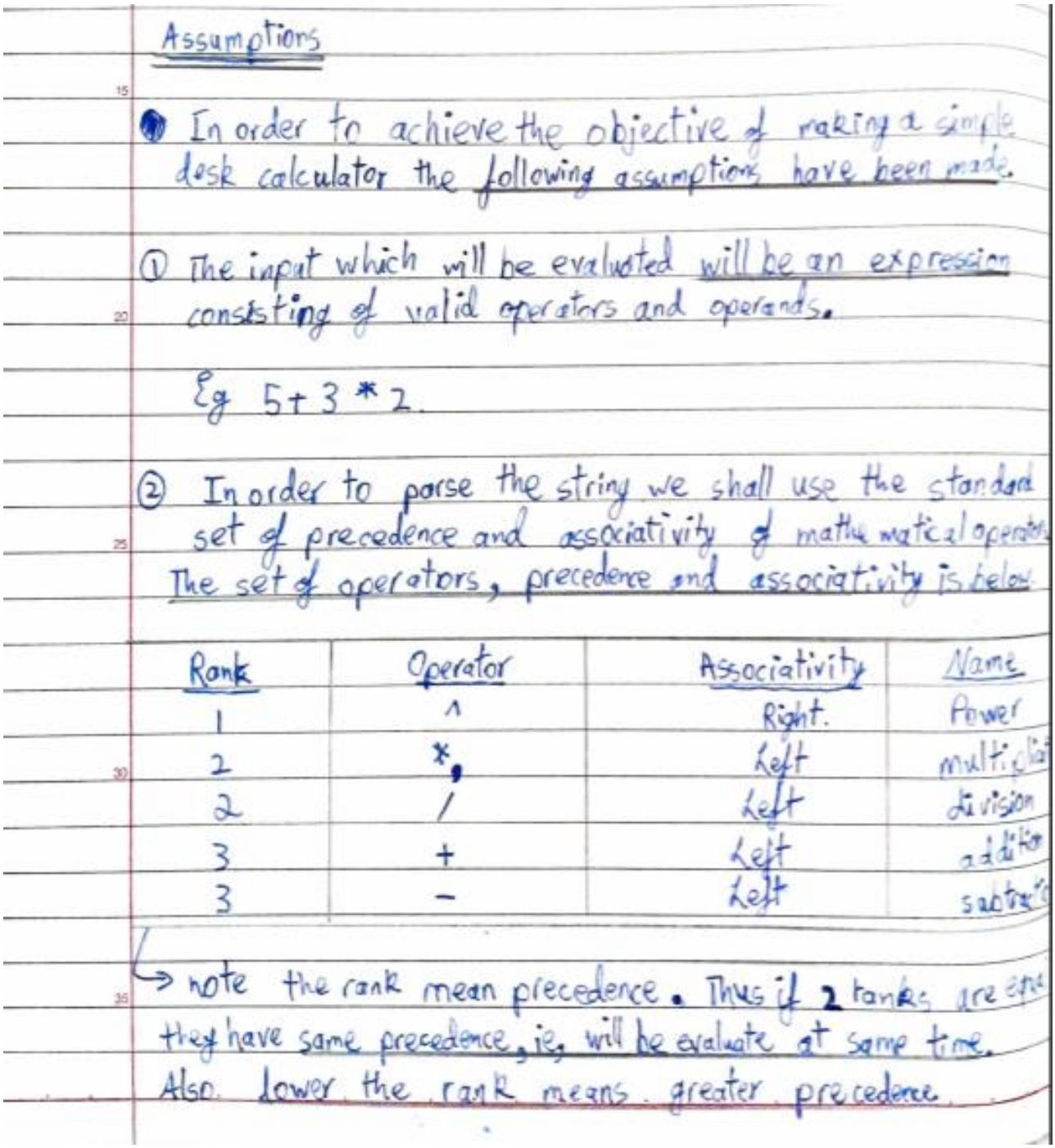
	+	*	-
+		<	>
*	>		>
-	<	<	

(b) Output of the calculator (on providing valid mathematical expression as an input to the program. E.g. if input is 2+2 to the program then output of the program should be 4)

(c) Error, if the input string is not valid.

(5) Submit your program, assumptions and grammar by **Nov 4, 2020**.

Assumptions (as picked from da submitted on 24th October 2020)



Assumptions

● In order to achieve the objective of making a simple desk calculator the following assumptions have been made.

① The input which will be evaluated will be an expression consisting of valid operators and operands.

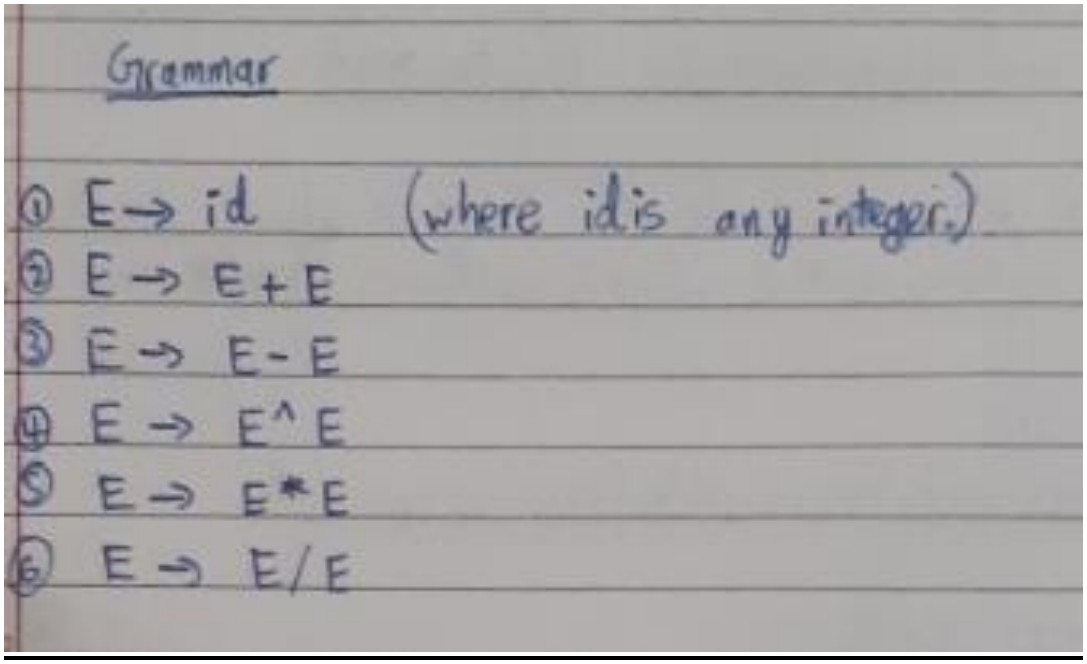
Eg 5+3*2.

② In order to parse the string we shall use the standard set of precedence and associativity of mathematical operators. The set of operators, precedence and associativity is below.

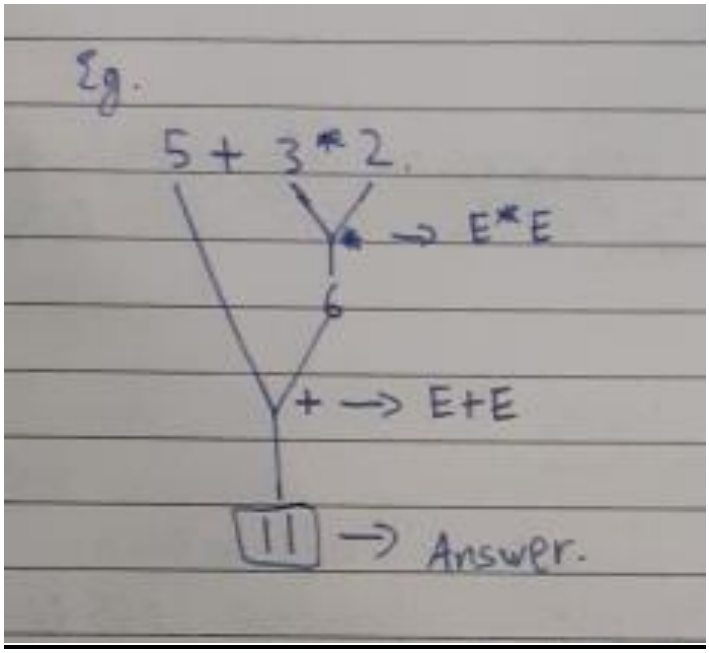
Rank	Operator	Associativity	Name
1	^	Right.	Power
2	*,	Left	multiplication
2	/	Left	division
3	+	Left	addition
3	−	Left	subtraction

→ note the rank mean precedence. Thus if 2 ranks are equal they have same precedence, ie, will be evaluate at same time. Also lower the rank means greater precedence.

Grammar
(production rules)



Example to illustrate evaluation of the code



CODE (written in C++)

```
#include <bits/stdc++.h>
using namespace std;
string wordsk[20];
int wordi=0;

bool isNumber(string s)
{
   for (int i = 0; i < s.length(); i++)
      if (isdigit(s[i]) == false)
         return false;

   return true;
}

void goodstring(string str)
{
  string word = "";
  for (auto x : str)
  {
     if (x == ' ')
     {
        wordsk[wordi]=word;
        wordi++;
        word = "";
     }
     else
     {
        word = word + x;
     }
  }
wordsk[wordi]=word;
wordi++;
}

bool a=false,b=false,c=false,d=false,e=false;
int precedence(char op){
           if(op == '+'||op == '-')
           return 1;
           if(op == '*'||op == '/')
           return 2;
           if (op == '^')
   return 3;

           return 0;
}
```

```
float applyOp(float a, float b, char op){

	switch(op){
		case '+': return a + b;
		case '-': return a - b;
		case '*': return a * b;
		case '/': return a / b;

		case '^': return pow(a,b);
	}
}


float evaluate(string tokens){
	int i;

	// stack to store integer values.
	stack <float> values;

	// stack to store operators.
	stack <char> ops;

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

		// Current token is a whitespace,
		// skip it.
		if(tokens[i] == ' ')
			continue;

		else if(isdigit(tokens[i])){
			int val = 0;

			// There may be more than one
			// digits in number.
			while(i < tokens.length() &&
						isdigit(tokens[i]))
			{
				val = (val*10) + (tokens[i]-'0');
				i++;
			}

			values.push(val);
		}

		// Current token is an operator.
		else
		{
```

```
                        // While top of 'ops' has same or greater
                        // precedence to current token, which
                        // is an operator. Apply operator on top
                        // of 'ops' to top two elements in values stack.
                        while(!ops.empty() && precedence(ops.top())
                                                        >= precedence(tokens[i])){
                                float val2 = values.top();
                                values.pop();

                                float val1 = values.top();
                                values.pop();

                                char op = ops.top();
                                ops.pop();

                                values.push(applyOp(val1, val2, op));

                        }

                        // Push current token to 'ops'.
                        ops.push(tokens[i]);
                }
        }

        // Entire expression has been parsed at this
        // point, apply remaining ops to remaining
        // values.
        while(!ops.empty()){
                float val2 = values.top();
                values.pop();

                float val1 = values.top();
                values.pop();

                char op = ops.top();
                ops.pop();

                values.push(applyOp(val1, val2, op));
        }

        // Top of 'values' contains result, return it.
        return values.top();
}

void valid(string wordsk[20])
{
   int cnt=0;
   while (wordsk[cnt]!= "\0")
```

```
    {
        cnt++;
    }

    if (!isNumber(wordsk[cnt-1]))
    {
        cout<<"wrong string"<<endl;
        exit(100);
    }
    for (int k=1;k<cnt;k+=2)
    {
        if (wordsk[k]=="+" || wordsk[k]=="-" || wordsk[k]=="*" || wordsk[k]=="/" || wordsk [k]=="^")
        {
            continue;
        }
        else
        {
            cout<<"wrong string"<<endl;
            exit(100);
        }
    }
    for (int k=0;k<cnt;k+=2)
    {
        if (!isNumber(wordsk[k]))
        {
            cout<<"error";
            exit(100);
        }
    }

}

void printing(char ar[6],int i,int k)
{
    cout<<"\t";
    cout << ar[k];
        for (int l=1;l<i;l++)
        {
            if (precedence(ar[k])>precedence(ar[l]))
            {
                cout <<" "<< '>' ;
            }
            else if (precedence(ar[k])==precedence(ar[l]) && precedence(ar[l])!=3)
            {
                cout<<" "<< '>';
            }
            else if (precedence(ar[k])< precedence(ar[l]))
            {
```

```
            cout<<" "<<'<';
          }
          else if (precedence(ar[k])==precedence(ar[l]) && precedence(ar[l])==3)
          {
            cout<<" "<<'<';
          }
        }
        cout << endl;
}

int main()
{
   cout << "enter a expression \nin a manner that the operators and operands are seperated by a
single space \nand only ^,*,/,+,- are allowed"<< "\n";
   string tokens;
   getline(cin,tokens);

   goodstring(tokens);
   valid(wordsk);

   cout<< endl;
   for (int i=0;i<tokens.length();i++)
   {
      switch (tokens[i])
      {
         case '+':a=true;
         break;
         case '-':b=true;
         break;
         case '*':c=true;
         break;
         case '/':d=true;
         break;
         case '^':e=true;
         break;
      }
   }
   //cout<< a <<" "<< b << " " << c << " " << d << endl;

   char ar[6];
   int i=1;

   if (a==true)
   {
      ar[i]='+';
      i++;
   }
   if (b==true)
```

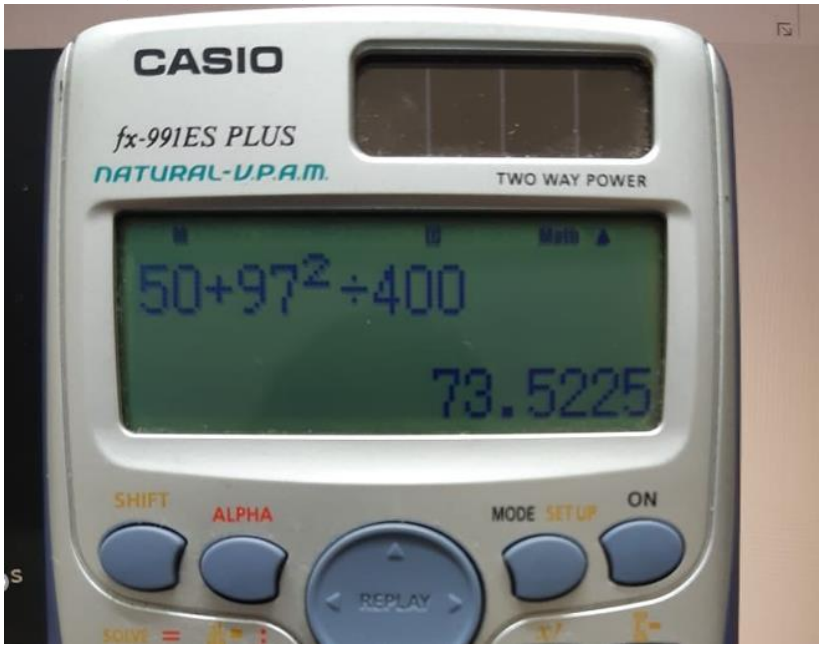
```
    {
       ar[i]='-';
       i++;
    }
    if (c==true)
    {
       ar[i]='*';
       i++;
    }
    if (d==true)
    {
       ar[i]='/';
       i++;
    }
    if (e==true)
    {
       ar[i]='^';
       i++;
    }
cout<<"OPERATOR PRECEDENCE FOR THE GIVEN EXPRESSION"<<endl;
cout<<"\t";
for (int j=0;j<i;j++)
    cout << ar[j]<< " ";
    cout<<endl;

for (int k=1;k<i;k++)
    {
       printing(ar,i,k);
    }
cout<<endl<<endl;
cout<<"_____"<<endl<<endl<<endl;

    cout<<"   "<<"Answer of  "<<tokens;
    cout<<" = "<< evaluate(tokens)<< endl;
    cout<<endl;
cout<<"_____"<<endl<<endl<<endl;
    return 0;
}
```

Output(5+97^2/400)

```
"C:\Users\arshd\Documents\CODEBLOCKS FILES\op_table.exe"
enter a expression
in a manner that the operators and operands are seperated by a single space
and only ^,*,/,+,- are allowed
50 + 97 ^ 2 / 400

OPERATOR PRECEDENCE FOR THE GIVEN EXPRESSION
          + / ^
        + > < <
        / > > <
        ^ > > <


_____


   Answer of  50 + 97 ^ 2 / 400 = 73.5225

_____


Process returned 0 (0x0)   execution time : 28.997 s
Press any key to continue.
```

Note

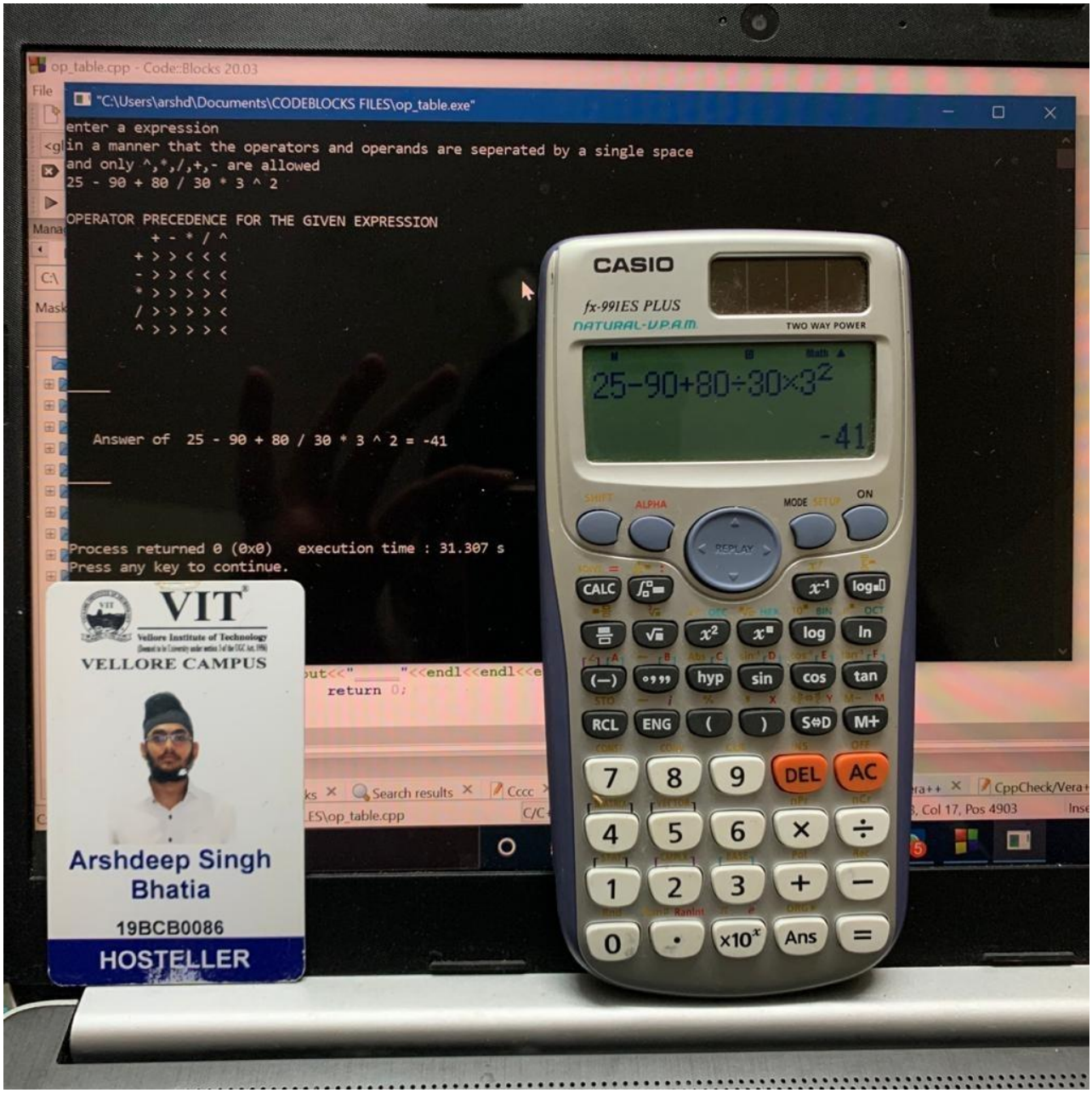
1. the output precedence table is dynamically generated
2. the string only contains '+','/' and '^' and similarly the table only prints those values
3. in order to verify the operator precedence the same question has been executed in a calculator(casio fx-991ES)



Another example (25 – 90 + 80/30 *3^2)

calc= -41 || my code= -41

Also dynamic operator precedence table is only plotted for -,+,/,*,^



Error(on entry of wrong string)

Eg:5 + 9 -

```
enter a expression
in a manner that the operators and operands are seperated by a single space
and only ^,*,/,+,- are allowed
5 + 9 -
wrong string

Process returned 100 (0x64)   execution time : 6.314 s
Press any key to continue.
```

Eg:+ 9 / -

```
enter a expression
in a manner that the operators and operands are seperated by a single space
and only ^,*,/,+,- are allowed
+ 9 / -
wrong string

Process returned 100 (0x64)   execution time : 10.632 s
Press any key to continue.
```