Experiment 7
Infix To Postfix Conversion and Evaluation

Date: 18-10-2020

Aim: To receive and infix expression and find the corresponding postfix expression and evaluate it.

Data Structure Used : Stack, Arrays

Algorithm for Conversion from infix to postfix

Input: Arithmetic expression E in infix notation with a right parenthesis at the end of the expression. A Stack with an opening parenthesis at the top, In-Stack precedence and incoming precedence of operators used.
Output: Corresponding postfix expression
Data Structure: Stacks
Operations Used : symbol() to a read the symbol from the expression

Steps:

```
Step 1: Top = -1, Push('(')
Step 2:while(Top>-1) do
        Step 1: item = E.symbol()
        Step 2: x = pop()
        Step 3:   case item=operand:
                        Step 1: Push(x)
                        Step2 : output(item)

                  case item=')':
                        Step 1:while x !=')' do
                                Step 1: output(x)
                                Step 2: x = Pop()
                        Step 2 :end while

                  case isp(x)>=icp(item):        //If the operator in the stack has a higher precedenc
                        Step 1: while isp(x)>=icp(item) do  //pop items from the stack until
                                Step 1: output(x)             //an item with lower percedence
                                Step 2: x = Pop()             // occurs
                        Step 2: end while
                        Step 3: Push(x)
                        Step 4: Push(item)           //Push the item into the stack

                  case isp(x) < icp(item):           //If the operator in the stack has a lower
                        Step 1 : Push(x)             //precedence, push the item into the
                        Step 2 : Push(item)          //stack

                  default :
                        Step 1 : Print("invalid expression")
Step 3 : EndWhile
Step 4 :Stop
```

Description of the Algorithm:

The algorithm converts infix expression to the corresponding postfix expression, using the stack data structure. When a operand is encountered then it is outputted but on encountering an operator all the operators with precedence higher than it is popped out of the stack and outputted to the postfix expression and on encountering an operator with a lower precedence then the scanned operator is pushed into the array. On encountering an open parenthesis it is push it into the stack and on encountering a closing parenthesis, all elements are popped out of the stack until an opening parenthesis is found.

Algorithm for Evaluation of the postfix Expression

Input: Postfix expression E
Output : Result after the evaluation of the postfix expression
Data Structure: Stacks
Operations used : *symbol()* to read a symbol from the expression, *performOperation(operand1,operand2, operator)*: performs the required mathematical operation denoted by operator

Steps:

```
1.    Step 1 : Start
2.    Step 2 : Top = -1
3.    Step 3 : item = E,symbol()
4.    Step 4 : Push(item)
5.    Step 5 : item = E.symbol()
6.    Step 6 : Push(item)
7.    Step 7 : while(Top > 0)
8.        Step 1: item = E.symbol()
9.        Step 2: if(item.isOperator() ) then
10.           Step 1 : operand2 = Pop()
11.           Step 2 : operand1 = Pop()
12.           Step 3 : result = performOperation(operand1,operand2,item)
13.           Step 4: Push(result)
14.       Step 3: else:
15.           Step 1: Push(item)
16.       Step 4: End While
17.   Step 8 : result = Pop()
18.   Step 9 : print result
```

Description of the algorithm

Elements are read form the postfix expression, if an operand is encountered it is pushed into the stack and if an operator is encountered, two elements are popped form the stack and the corresponding operation is performed. Then the result is pushed into the stack. If there is only one element in the stack then that is the result of the expression

Program Code

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define SIZE 50

//START: Defnition of structures

//Character Stack
typedef struct stacks
{
    int top;
    char stk[SIZE];
}stack;

void push(stack *s,char ch)
{
    if(s->top>=SIZE)
    {
        printf("Stack Overflow Error.EXITING\n");
        exit(0);
    }
    else{
        s->stk[++(s->top)] = ch;
    }
    return;
}

char pop(stack *s)
{
    if(s->top<0)
    {
        printf("Error has occurred. Stack is empty\n");
        exit(0);
        return 0;
    }
    else
    {
        char a = s->stk[s->top];
        (s->top)--;
        return a;
    }
}

//Integer Stack
typedef struct stacksInt
{
    int top;
    int stk[SIZE];
}intStack;

void intPush(intStack *s,int a)
{
    if(s->top>=SIZE)
    {
        printf("Stack Overflow Error.EXITING\n");
        exit(1);
    }
    else{
        s->stk[++(s->top)] = a;
    }
    return;
}

int intPop(intStack *s)
{
    if(s->top<0)
    {
        printf("Error has occurred. Stack is empty\n");
        exit(1);
        return 0;
    }
    else
    {
        int a = s->stk[s->top];
        (s->top)--;
        return a;
    }
}


//START: Utility funtions definition

/* the utility function used are:
 * 1. int verifyExpression(char* exp) --> Checks if the expression is valid, i.e there are correct 
number of operators and operands
 *
 * 2. void printExpression(char* exp) --> Prints the expression
 *
 * 3. int findPrecedence(char a) -->returns the instack precedence of the operator passed
 *
 * 4. void getValues(char *exp, int **values) -->since the expression contains only chrecter variables 
this funciton
 *                                               asks the user for values for each of the charecter
 *
 * 5. int getValue(char c, int** values) -->this funtion is used to find the value of the charecter c 
from the
 *                                          array of values user has entered
 *
 * 6. int evaluate(int a, int b, char c) -->performs the operation c with a as the first operand and b 
as the second operand
 *
 * */


int verifyExpression(char* exp) //Verifies wheather the given expression contains only mathematical 
operations and letters
{
    int i = 0;
    //char *cleared = (char*) malloc(50*(sizeof(char)));
    int operands = 0;
    int operators =0;
    int paranthsis = 0;
    for(i = 0;exp[i]!='\0';i++){
        if(!isalpha(exp[i])){
            if(!(exp[i]=='+'||exp[i]=='-'||exp[i]=='*'||exp[i]=='/'||exp[i]=='^'||exp[i]==')'|| 
exp[i]=='('))
            {
                printf("The Expression must contain only alphbets and mathematical operators \n\n");
                return 0;
            }
            else{
                if(exp[i]=='*'|| exp[i]=='/'|| exp[i]=='-'|| exp[i]=='+'||exp[i]=='^' )
                {
                    operators ++;
                }else if(exp[i]=='('||exp[i]==')'){
                    paranthsis++;
                }
            }
        }
        else{
            operands++;
        }
    }
    if(operators+1 > operands)
    {
        printf("The number of operators for %d operand(s) shoud be: %d but %d found\
n",operands,operands-1,operators);
        exit(1);
    }
    else if(operators+1< operands){
        printf("The number of operands for %d operation(s) must be %d, but %d found\
n",operators,operators+1,operands);
        exit(1);
    }

    if (paranthsis%2!=0)
    {
        printf("ERROR!!!!! The Expression contains an incomplete paranthesis\n");
        exit(1);
    }
    return 1;
}

void printExpression(char* exp) //Prints the given experssion
{
    for(int i = 0;exp[i]!='\0';i++)
        printf("%c ",exp[i]);
    printf("\n");
}

int findPrecedence(char a)
{
    switch(a)
    {
        case '+':
        case '-':return 2;
                 break;
        case '*':
        case '/':return 4;
                 break;
        case '^':return 5;
                 break;
        case '(':return 0;
                 break;
        default: printf("Invalid Expression \n");
                 exit(0);
                 return -1;
    }

}

void getValues(char *exp, int **values)
{
    int i = 0;int j = 0;
    int alreadyScanned = 0;


    for(i = 0; exp[i]!='\0';i++)
    {
        alreadyScanned = 0;
        if(isalpha(exp[i])){
            for(j =0;values[j][0]!=0;j++){
                if(exp[i]== values[j][0]){
                    alreadyScanned = 1;
                }
            }
            if(!alreadyScanned){
                values[j][0] = exp[i];
            }
        }
    }

    printf("Enter the values of ");
    for(j = 0; values[j][0]!=0;j++){
        printf("%c, ",(char)values[j][0]);
    }
```

```
        printf("\b\b: ");

    fflush(stdout);
    fflush(stdin);

    for(j=0;values[j][0]!=0;j++)
    {
        scanf("%d%*c",&values[j][1]);
    }

    /*
    printf("\nThe values entered are :\n");
    for(j = 0; values[j][0]!=0;j++){
        printf("%c - %d\n",(char)values[j][0],values[j][1]);
    }*/
}

int getValue(char c, int** values)
{
    int i=0;
    while(values[i][0]!=c)
    {
        i++;
    }
    return values[i][1];
}

int evaluate(int a, int b, char c)
{
    int i =0;
    int res=1;
    switch(c)
    {
        case '+':return a+b;
        case '-':return a-b;
        case '*':return a*b;
        case '/':return a/b;
        case '^':while(i++<b) res *=a;
                 return res;
        default : printf("Such an charecter is not found Exiting\n");
                  exit(1);

    }
}
//END: Defenition of utility funtions


//START: Convertion to postfix Algorithm
char* convertToPostFix(char* str)
{
    int i=0,j=0;
    int isp,icp;
    int operand;

    char stkItem;

    stack *s = (stack*) malloc(sizeof(stack));
    s->top = -1;
    push(s,'(');

    for(;str[i]!='\0';i++);
    str[i] = ')';
    str[i+1] = '\0';

    // printExpression(str);   //for debugging purposes

    char *postfixExp = (char*) malloc(i*sizeof(char));
    for(i=0;str[i]!='\0';i++)
    {
        /*printf("Iterarion %d\nItem Read = %c ",i+1,str[i]);*/ //for debugging purposes
        operand = 0;
        switch(str[i])
        {
            case '+':;
            case '-':icp = 1;
                    break;

            case '*':
            case '/':icp = 3;
                    break;

            case '^':icp = 6;
                    break;

            case '(':icp = 9;
                    break;

            case ')':icp = 0;
                    break;

            default :postfixExp[j] = str[i];
                     j++;
                     operand = 1;
                     break;
        }
        if(!operand)
        {
            stkItem = pop(s);
            //printf("stkItem = %c",stkItem);  //for debugging purposes

            if(str[i]!=')')
            {
                isp = findPrecedence(stkItem);
                while(isp>=icp)
                {
                    if(s->top == -1)
                    {
                        printf("Invalid Expression\n");
                        exit(1);
                    }
                    postfixExp[j]=stkItem;
                    j++;
                    stkItem = pop(s);
                    isp = findPrecedence(stkItem);
                }
                push(s,stkItem);
                push(s,str[i]);
            }
            else
            {
                while(stkItem!='(')
                {
                    postfixExp[j] = stkItem;
                    j++;
                    stkItem = pop(s);
                }
            }
        }
        /*printf("\n-----------------------------------"); //for debugging purposes
        printf("\n");*/
    }
    postfixExp[j] = '\0';
    return postfixExp;

}
//END: Convertion to postfix Algorithm

//STRAT: Evatuation algorithm
```

```
int evaluatePostfix(char *exp)
{
    int **values = (int**) malloc(SIZE*sizeof(int*));
    intStack *s = (intStack*)malloc(sizeof(intStack));
    s->top = -1;
    int operand1,operand2;

    int i;
    for(i = 0; i<SIZE;i++)
    {
        values[i] = (int*) calloc(2,sizeof(int));
    }

    getValues(exp,values);

    for(i=0;exp[i]!='\0';i++)
    {
        switch(exp[i])
        {
            case '+':
            case '-':
            case '*':
            case '/':
            case '^':
                     operand2 = intPop(s);
                     operand1 = intPop(s);
                     intPush(s,evaluate(operand1,operand2,exp[i]));
                     break;

            default: intPush(s,getValue(exp[i],values));
                     break;
        }
    }
    free(values);
    return intPop(s);
}

//END : Evaluation algorithm


int main()
{
    char *exp = (char*) malloc(50*(sizeof(char)));
    char *pexp;
    char c;

        printf("Infix to postfix conversion and evaluation\n");
        printf("=========================================\n");
        printf("Enter the infix expression for \"20+30\" as \"a+b\" without any spaces\n");
        printf("between the characters and then later enter the values of a and b when asked\n\n");

    printf("Enter the Infix Expression : ");
    scanf("%[^\n]",exp);
    while((c = getchar()) != '\n' && c != EOF);
    if(verifyExpression(exp))
    {
        //printExpression(exp);
        pexp = convertToPostFix(exp);
        printf("\nPostfix Expression is -->  ");
        printExpression(pexp);
        int result = evaluatePostfix(pexp);
        printf("The result is = %d\n",result);
        free(pexp);

    }
    free(exp);
        return 0;
}
```

Sample input 1

```
((A+((B^C)-D))*(E-(A/C)))
3 5 -1 9 20
```

Sample output 1

```
Infix to postfix conversion and evaluation
==========================================
Enter the infix expression for "20+30" as "a+b" without any spaces
between the characters and then later enter the values of a and b when asked

Enter the Infix Expression : ((A+((B^C)-D))*(E-(A/C)))

Postfix Expression is -->  A B C ^ D - + E A C / - *
Enter the values of A, B, C, D, E: 3 5 -1 9 20
The result is = -115
```

Sample input 2

```
A^b^D
-2 2 3
```

Sample output 2

```
Infix to postfix conversion and evaluation
==========================================
Enter the infix expression for "20+30" as "a+b" without any spaces
between the characters and then later enter the values of a and b when asked

Enter the Infix Expression : A^b^D

Postfix Expression is -->  A b D ^ ^
Enter the values of A, b, D: -2 2 3
The result is = 256
```

Sample input 3

```
a-(b+c*c)+e*f
90 -20 4 1 6
```

Sample output 3

```
Infix to postfix conversion and evaluation
==========================================
Enter the infix expression for "20+30" as "a+b" without any spaces
between the characters and then later enter the values of a and b when asked

Enter the Infix Expression : a-(b+c*c)+e*f

Postfix Expression is -->  a b c c * + - e f * +
Enter the values of a, b, c, e, f: 90 -20 4 1 6
The result is = 100
```