VG101 Lab Manual

Lab 8

Instructor: Dr. Zhu Yifei
TA: Cao Hangrui
TA: Shan Haoxuan
TA: Wu Qinhang
TA: Xu Muchen

Table of Contents

Lab 8 Manual

Workflow

Content	Approx. Time
Warmup: Basic Syntax Exercise	30 mins
Expression evaluation	60 mins
Break	10 mins
Practical Exercise	50 mins

Expression evaluation

In this lab, we will mainly focus on one question: **expression evaluation**. We will deal with a mathematical expression and calculate its corresponding output. Sample inputs that we're going to deal with are listed below:

```
-1 - (9 + 7 * (2/3) ) / (2 - (3 - 4/5))
1/3/3*2-(1-2-(5-9-(-5-10)))
```

It's hard for us to treat such a complex string directly, so we will separate it into several tasks. **If you meet difficulties, you can have a view on possible solutions mentioned in the last part.**

Task 1: Basic a+b

Input a formula, which includes two numbers and one operator (+,-,*,/). Give the answer for the formula. Consider all results are integer.

```
Input: 1-1
Output: 0
Input: 12/4
Output: 3
```

Hint: You can use atoi to change a string into a number in C.

Task 2: Multi-operator operation

This time, the formula contains multiple operators and they may be various kinds. Again, output result.

```
Input: -1+4*5/2-3
Output: 16
Input: 12/2/3+2*3*2
Output: 14
```

Hint: Consider the priority order of operators

Task 3: Elimate Spaces

A quite easy procedure. Input a string with spaces and output it after deleting all spaces.

Task 4: Brackets

Now, the formula contains brackets. Start from only one operator '-'.

```
Input: -1-(4-7)-6
Output: -4
Input: 0-((5-(6-9))-15)-1
Output: -8
```

Task 5: Complete Question

Then, consider formula with all operators and finish the question.

Solution 1: Recursion

Design the function below.

```
int evalVg101(char str[],int left_marker,int right_marker);
```

Each time, only deal with the operator that should be calculated last. For example, the `#` place in the formula below

```
2/3/4
***#*

1+2*3/4
*#*****

1-2*3*4-2
*******#*
```

In your function, you may have something like

```
if( str[i] == '+' )
    return evalVg101(str,left_marker,i-1) +  evalVg101(str,i+1,right_marker);
```

For brackets, you should skip the part covered by brackets when there're still some operators outside the brackets. When there's no operators outside, and brackets exists, you can then recursively call with dealing string between '`left_marker+1`' and '`right_marker-1`'. (It may be helpful to write the formula on a paper and draw out the process)

To improve efficiency, you can firstly do pretreatment with the brackets, matching '`(`' with '`)`'. A Pseudo Code can be:

```
int bracketPosition[MAXSIZE];
int bracketStack[MAXSIZE];
int bracketStackHeight = 0;

for i = 0 ~ strlen(str) - 1
    if( str[i] == '(' )
        bracketStack[bracketStackHeight++] = i;
    else if( str[i] == ')' )
        bracketPosition[i] = bracketStack[bracketStackHeight-1];
        bracketPosition[bracketStack[bracketStackHeight-1]] = i;
        bracketStackHeight--;
    else
        continue;
```

Solution2 : Reverse Polish Notation Method

This methods can be more efficient than the previous method. However, it requires your deeper understanding about calculation order.

You can skip this method if you find it difficult to understand.

Before we start to tell the method, we need to introduce some knowledge, including "Binary Tree ", "Infix Expression", "Postfix Expression".

Binary Tree

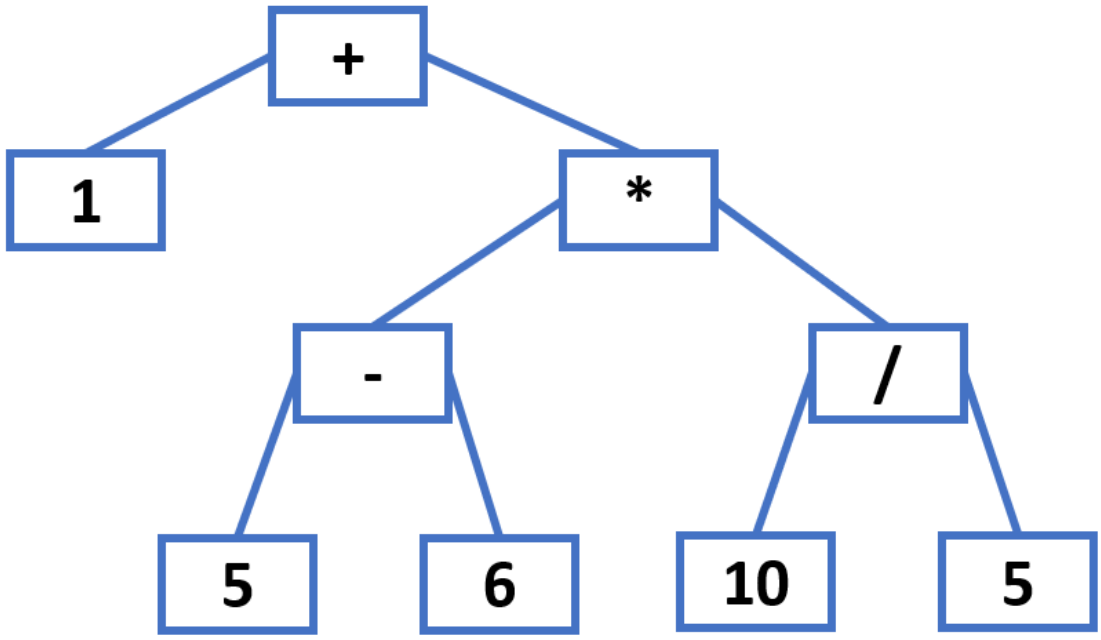
A binary tree contains a tree structure, contains the following possible structures:

1. nothing
2. a piece of data that we care about, together with the information about 2 other binary tree nodes(referred as children).

Basically we will define a tree structure as a struct in C, for example:

```
typedef struct binaryTree{
    // it can be any other form of data, /like int, float or data type you
defined
    char data;
    struct binaryTree *leftchild;
    struct binaryTree *rightchild;
}
```

Also for each tree, there should be a root node pointing to the root of the tree. Like in the following tree shown below, the root node points to char `'+'`, and the root node has two child, the left child is the tree struct containing `'1'` and the right child containing `'*'`. For the tree node containing `'1'`, it does not have any child nodes. But for `'*'` it has child nodes `'-'` and `'/'`.



Infix Expression

For the tree showed above, its infix expression is

```
1+((5-6)*(10/5))
```

More general, infix expression is the most common expression we usually use in life.

Postfix Expression

For the tree showed above, its postfix expression is

```
1 5 6 - 10 5 / * +
```

To calculate a postfix expression, you can read from left to right. When reading an operator, calculate the two numbers on its left, and replace the two numbers with one number representing the result. Repeat the process until read the whole string. A Pseudo Code can be:

```
int lastNumber = 0;
int numberStack[MAXSIZE];
for i = 0 ~ length(str)-1
    if( isoperator(str[i]) )
        numberStack[lastNumber-1] = \
        calc(numberStack[lastNumber-1], \
        numberStack[lastNumber],str[i]);
        lastNumber--;
```

```
    else if( isdigit(str[i]) )
        find the end of this number, deote as j
        numberStack[++lastNumber] = eval(str,i,j);
        i = j;
    else
        continue;
int ans = numberStack[0];
```

Visit elements in a tree

An example:

To visit elements in a tree structure, there are three methods and their corresponding pseudo codes are shown as follows:

```
function preorder(tree node){
    if(tree is not empty)
    {
        operationon(root of the data);
        preorder(left child of the tree);
        preorder(right child of the tree);
    }
}
```

```
function inorder(tree node){
    if(tree is not empty)
    {
        inorder(left child of the tree);
        operationon(root of the data);
        inorder(right child of the tree);
    }
}
```

```
function postorder(tree node){
    if(tree is not empty)
    {
        postorder(left child of the tree);
        postorder(right child of the tree);
        operationon(root of the data);
    }
}
```

Change a infix to postfix

A Pseudo Code can be: (**Please note that this code is simply showing you a possible structure and doesn't address all detailed issues**)

```
outputStack[]
OperatorStack[]

for i = 0 ~ end
    if( isnumber(str[i]) )
```

```
        add number to the tail of outputStack
    else if( isoperator(str[i]) )
        while( OperatorStack is not empty and not '(' and \
            operator before str[i] has higher priority ) )
                output the tail of OperatorStack to \
                the tail of OperatorStackoutputStack
        add str[i] to the tail of OperatorStackoutputStack

    else if ( str[i] == '(')
        add to the tail of OperatorStack

    else if ( str[i] == ')')
        find the '(' in Stack and \
        add all operators between two brackets into outputStack
```

Grading Rubric

Criteria	Weight	Available Time	Due Time	Entry
Attendance	30%	**4:00pm**, July.3	**11:59am**, July.4 (**noon**)	Canvas Assignment
In-lab quiz	70%	**9:00pm**, July.3	**11:59pm**, July.5	Canvas Quiz

- For the attendance score, you need to submit your code for **exercises** in the **worksheet** on Canvas. We won't judge the correctness of your code. You'll earn full credits as long as you've tried these exercises.