Lab 1 Report

Name: Hau Tao

Lab number: 24193 - CSE 330- Data Structures

Winter 2016

1. Status

I completed 100% of the lab including conversion infix-to-postfix expressions and evaluate these expressions

2. Complexity analysis

1. Converting infix to postfix expression

Time complexity	Storage complexity	function
O(n)	O(1)	main
O(n)	O(1)	prec

2. Evaluation expression

Time complexity	Storage complexity	function
O(n)	O(1)	main
O(n)	O(1)	prec
O(n)	O(1)	evaluation_stack
O(n)	O(1)	get_operand

3. Source Code

1. For the first part of the lab, I implemented the algorithm to convert infix to postfix expression. Here is my code

```
/*****************************************************************************
* Hau Tao
* lab1.cpp
* 01/17/2016
* This program implement infix-to-postfix expression conversion
* Valid operands are single digits and characters: 0-9 a-z A-Z
* For example,   a + b - c          translates to   a b + c -
*                a + b * c          translates to   a b c * +
*          (a + 2) / (5 - d)         goes to   a 2 + 5 d - /
*          a + ((b - c) * d) / e   to   a b c - d * e / +
* Valid operators are: + - * / ( )
* Highest precedence:  * /
* Lowest precedence:   + -
* ( has lowest precedence on the stack and highest precedence outside of stack.
* ) never goes on stack.
* Bottom of the stack has the lowest precedence than any operator.
* Use a prec() function to compare the precedence of the operators based
* on the above rules.
*****************************************************************************/
#include <iostream>
#include <stack>
using namespace std;
int prec(char );
main()
{
        char in ;
  stack <int> my_stack;
  cout <<"Enter your input " << endl;
  cin >> in ;
  while(!cin.eof()){
        if((in >='a' && in <= 'z' )||(in >= '0' && in <= '9')|| (in >= 'A' && in <= 'Z'))
```

```
                cout << in;
        else {
                if(in == '(')
                        my_stack.push(in);
                else if (in == ')'){
                        if(!my_stack.empty() && my_stack.top() != '('  ){
                                cout <<char( my_stack.top());
                                my_stack.pop();
                        } else
                                cout <<"Error: No matching '('";

                }
                else if (in =='*'|| in =='/'|| in =='+'|| in=='-'){
                        if(my_stack.empty() || prec(my_stack.top()) < prec(in))
                                my_stack.push(in);
                        else {
                                cout << char(my_stack.top());
                                my_stack.pop();
                                my_stack.push(in);
                        }
                }
                else
                        cout << "Error input:";
        }
        cin >> in;
  }

  // Print out the stack when stopping input
  while(!my_stack.empty()){
        if(my_stack.top()!= '('){
                cout << char(my_stack.top());
                my_stack.pop();
        } else
                my_stack.pop();
  }
  cout << endl;

}

// Check precedence
int prec(char in )
{
  if(in =='*'|| in =='/')
        return 2;
  else if (in =='+'|| in =='-')
        return 1;
  else
        return 0;
```

```
}
```

2. For the second part , I added one more stack to evaluate expressions. Here is my code:

```
/*********************************************************************************************
 * Hau Tao
 * lab1_extension.cpp
 * 01/17/2016
 * This program implement infix-to-postfix expression conversion and evaluate them
 * The algorithm to convert infox to postfix expression
 * Valid operands are single digits and characters: 0-9 a-z A-Z
 * Valid operators are: + - * / ( )
 * Highest precedence:  * /
 * Lowest precedence:   + -
 * ( has lowest precedence on the stack and highest precedence outside of stack.
 * ) never goes on stack.
 * Bottom of the stack has the lowest precedence than any operator.
 * Use a prec() function to compare the precedence of the operators based
 * on the above rules.
 * The algorithm to evaluate the expression
 * Scanning operands in postfix expressions and push into the new stack
 * until meeting up the operator, take 2 operands from new stack, pop them out and
 * Evaluate them, the result of this valuation will be pushed back to the stack .
 * Continuing this process until no more operators
 *********************************************************************************************/
#include <iostream>
#include <stack>
using namespace std;
int prec(char );
char evaluation_stack(char , char , char );
void  get_operand(stack<int> & evaluation, stack <int> & my_stack);
main()
{
        char in, operand_1, operand_2 , result;
   stack <int> my_stack;
   stack <int> evaluation;
   cout <<"Enter your input " << endl;
   cin >> in ;
   while(!cin.eof()){
        if((in >='a' && in <= 'z' )||(in >= '0' && in <= '9')|| (in >= 'A' && in <= 'Z')){
                cout << in;
                evaluation.push(in-'0');
        } else {
                if(in == '(')
                        my_stack.push(in);
                  else if (in == ')'){
                        if(!my_stack.empty() && my_stack.top() != '('  ){
                                cout <<char( my_stack.top());
                                  get_operand(evaluation, my_stack);
                                  my_stack.pop();
                        }
                        else
                                cout <<"Error: No matching '('";
                  }
```

```
                else if (in =='*'|| in =='/'|| in =='+'|| in=='-'){
                        if(my_stack.empty() || prec(my_stack.top()) < prec(in))
                                my_stack.push(in);
                        else {
                                cout << char(my_stack.top());
                                get_operand(evaluation, my_stack);
                                my_stack.pop();
                                my_stack.push(in);
                        }
                }
                else
                        cout << "Error input:";
         }
        cin >> in;
   }

   // print out the stack when stopping input
   while(!my_stack.empty()){
        if(my_stack.top()!= '('){
                cout << char(my_stack.top());
                get_operand(evaluation, my_stack);
                my_stack.pop();
        } else
                my_stack.pop();
   }
   cout << endl;
   cout <<"Evaluation result: ";
   while(!evaluation.empty()){
        cout << evaluation.top();
        evaluation.pop();
   }
   cout << endl;

}

// Check precedence
int prec(char in )
{
   if(in =='*'|| in =='/')
        return 2;
   else if (in =='+'|| in =='-')
        return 1;
   else
        return 0;

}
// Evaluate 2 operands from stack

char evaluation_stack(char operators, char operand_1, char operand_2)
{

   if(operators == '*')
        return (operand_1 * operand_2) ;
   else if (operators == '/')
        return (operand_1 / operand_2)  ;
   else if (operators == '+')
        return (operand_1 + operand_2);
   else if (operators == '-')
```

```
            return (operand_1 - operand_2);

}
// Take 2 operands from the stack and evaluate them
void  get_operand(stack<int> & evaluation, stack <int> & my_stack)
{
   char operand_1, operand_2, result;
   operand_2 = evaluation.top();
   evaluation.pop();
   operand_1 = evaluation.top();
   evaluation.pop();
   result = evaluation_stack(char(my_stack.top()), operand_1, operand_2);
   evaluation.push(result);

}
```

4. Sample Run.

1. The sample run for conversion from infix to postfix expression

```
Script started on Mon 18 Jan 2016 09:55:08 PM PST
]0;hau@hau-Lenovo-Y50-70: ~/Desktop/CSE 330/lab1hau@hau-Lenovo-Y50-
70:~/Desktop/CSE 330/lab1$ g++ lab1.cpp
.]0;hau@hau-Lenovo-Y50-70: ~/Desktop/CSE 330/lab1hau@hau-Lenovo-Y50-
70:~/Desktop/CSE 330/lab1$ ./a.out
Enter your input
a+b
ab+
]0;hau@hau-Lenovo-Y50-70: ~/Desktop/CSE 330/lab1hau@hau-Lenovo-Y50-
70:~/Desktop/CSE 330/lab1$ ./a.out
Enter your input
a*b+c
ab*c+
]0;hau@hau-Lenovo-Y50-70: ~/Desktop/CSE 330/lab1hau@hau-Lenovo-Y50-
70:~/Desktop/CSE 330/lab1$ ./a.out
Enter your input
a+b*c
abc*+
]0;hau@hau-Lenovo-Y50-70: ~/Desktop/CSE 330/lab1hau@hau-Lenovo-Y50-
70:~/Desktop/CSE 330/lab1$ (a[K[K./a.out
Enter your input
```

```
(a+b)*(c+d)
ab+cd+*
]0;hau@hau-Lenovo-Y50-70: ~/Desktop/CSE 330/lab1hau@hau-Lenovo-Y50-
70:~/Desktop/CSE 330/lab1$ exit

Script done on Mon 18 Jan 2016 09:56:04 PM PST
```

2. The sample run for evaluating expressions

```
Script started on Mon 18 Jan 2016 10:27:22 PM PST
]0;hau@hau-Lenovo-Y50-70: ~/Desktop/CSE 330/lab1hau@hau-Lenovo-Y50-70:~/Desktop/CSE
330/lab1$ g++ lab1_extension.cpp
]0;hau@hau-Lenovo-Y50-70: ~/Desktop/CSE 330/lab1hau@hau-Lenovo-Y50-70:~/Desktop/CSE
330/lab1$ ./a.out
Enter your input
1+2
12+
Evaluation result: 3
]0;hau@hau-Lenovo-Y50-70: ~/Desktop/CSE 330/lab1hau@hau-Lenovo-Y50-70:~/Desktop/CSE
330/lab1$ ./a.out
Enter your input
1*2+5
12*5+
Evaluation result: 7
]0;hau@hau-Lenovo-Y50-70: ~/Desktop/CSE 330/lab1hau@hau-Lenovo-Y50-70:~/Desktop/CSE
330/lab1$ ./a.out
Enter your input
2+3*2
232*+
Evaluation result: 8
]0;hau@hau-Lenovo-Y50-70: ~/Desktop/CSE 330/lab1hau@hau-Lenovo-Y50-70:~/Desktop/CSE
330/lab1$ ./a.out
Enter your input
(1+2)/(1+1)^[[D     0)
12+10+/
Evaluation result: 3
]0;hau@hau-Lenovo-Y50-70: ~/Desktop/CSE 330/lab1hau@hau-Lenovo-Y50-70:~/Desktop/CSE
330/lab1$ exit

Script done on Mon 18 Jan 2016 10:28:56 PM PST
```