

CSCI 140 PA 5 Submission

Due Date: 09/30/2021 Late (date and time): _____

Name(s): Nero Li

Exercise 1 -- need to submit source code and I/O

-- check if completely done ☒; otherwise, discuss issues below

Source code below:

```
/* Program: PA_5_exercise_1
   Author: Nero Li
   Class: CSCI 220
   Date: 09/30/2021
   Description:
       You must use either existing C++ stack class or Java Stack class
to solve
       the "Balancing Symbols" problem. The symbols are (), [], and {},
and each
       opening symbol must have a corresponding closing symbol as well as
in correct
       order. Ignore operands and arithmetic operators since they are not
relevant
       to our problem. You can assume each token is separated by spaces.
```

I certify that the code below is my own work.

Exception(s): N/A

```
*/

#include <iostream>
#include <stack>

using namespace std;

bool func(char str[])
{
    stack<char> braces;
    for (int i = 0; str[i] != '\0'; ++i)
    {
        switch (str[i])
        {
            case '(':
            case '[':
            case '{':
```

```

        braces.push(str[i]);
        break;
    case ')':
        if (braces.top() == '(')
        {
            braces.pop();
        }
        else
        {
            return false;
        }
        break;
    case ']':
        if (braces.top() == '[')
        {
            braces.pop();
        }
        else
        {
            return false;
        }
        break;
    case '}':
        if (braces.top() == '{')
        {
            braces.pop();
        }
        else
        {
            return false;
        }
        break;
    default:
        break;
}

}

if (braces.size() != 0)
{
    return false;
}

return true;
}

int main()
{
    char str1[] = {"{ ( a + b ) * c1 }"};
    char str2[] = {"{ ( a + b ) * c1 ]"};
    char str3[] = {"( ( a + b ) * c1 } / 15 )"};
    char str4[] = {"( ( ( ( ( ( ( [ [ [ { { } } ] ] ] ) ) ) ) ) ) ) )"};
    char str5[] = {"( [ ) ]"};

```

```

    cout << str1 << " - " << (func(str1) ? "valid" : "invalid") << endl;
    cout << str2 << " - " << (func(str2) ? "valid" : "invalid") << endl;
    cout << str3 << " - " << (func(str3) ? "valid" : "invalid") << endl;
    cout << str4 << " - " << (func(str4) ? "valid" : "invalid") << endl;
    cout << str5 << " - " << (func(str5) ? "valid" : "invalid") << endl;

    cout << "Author: Nero Li\n";
    return 0;
}

```


Input/output below:

```

{ ( a + b ) * c1 } - valid
{ ( a + b ) * c1 ] - invalid
( ( a + b ) * c1 } / 15 ) - invalid
( ( ( ( ( ( ( [ [ [ { { } } ] ] ] ) ) ) ) ) ) - valid
( [ ) ] - invalid
Author: Nero Li

```

Exercise 2 -- need to submit source code and I/O

-- check if completely done ; otherwise, discuss issues below

Source code below:

```

/* Program: PA_5_exercise_2
   Author: Nero Li
   Class: CSCI 220
   Date: 09/30/2021
   Description:
       You must define and implement your own Stack class or use the one
from
       the textbook. The Stack class supports standard basic stack
operations,
       and you can implement it with an array or a linked list. You
should create
       a class template Stack in C++ or generic class Stack in Java, but
an
       integer stack would work as well. Set up a function or static
method that
       receives a string representing a postfix expression and it returns
an
       integer result. Your function/method uses a stack to evaluate a
postfix
       expression (see an operand -- push; see an operator -- pop twice,
evaluate,
       then push result).

```

I certify that the code below is my own work.

Exception(s): N/A

```

*/

#include <iostream>
#include <cmath>

using namespace std;

template <typename T>
class Stack
{
    public:
        Stack() {}

        int size()
        {
            return i;
        }

        bool empty()
        {
            if (i == 0)
            {
                return true;
            }
            return false;
        }

        T top()
        {
            if (empty())
            {
                cout << "StackEmpty\n";
                exit(-1);
            }
            return head->n;
        }

        void push(T n)
        {
            Node *cur = new Node;
            cur->n = n;
            cur->next = head;
            head = cur;
            ++i;
        }

        T pop()
        {
            T lost;
            Node *cur;
            if (empty())
            {

```

```

        cout << "StackEmpty\n";
        exit(-1);
    }
    lost = head->n;
    cur = head;
    head = head->next;
    delete(cur);
    --i;
    return lost;
}

private:
    int i{0};
    struct Node
    {
        T n;
        Node *next;
    } *head{NULL};
};

int func(string str)
{
    Stack<int> expression;
    int value{0};

    for (size_t i = 0; i < str.size(); ++i)
    {
        if (str[i] >= '0' && str[i] <= '9')
        {
            value *= 10;
            value += str[i] - '0';
        }
        else if (str[i] == '+')
        {
            int a{expression.pop()};
            int b{expression.pop()};
            expression.push(b + a);
            ++i;
        }
        else if (str[i] == '-')
        {
            int a{expression.pop()};
            int b{expression.pop()};
            expression.push(b - a);
            ++i;
        }
        else if (str[i] == '*')
        {
            int a{expression.pop()};
            int b{expression.pop()};
            expression.push(b * a);
            ++i;
        }
    }
}

```

```

    }
    else if (str[i] == '/')
    {
        int a{expression.pop()};
        int b{expression.pop()};
        expression.push(b / a);
        ++i;
    }
    else if (str[i] == '^')
    {
        int a{expression.pop()};
        int b{expression.pop()};
        expression.push(pow(b, a));
        ++i;
    }
    else
    {
        expression.push(value);
        value = 0;
    }
}

return expression.pop();
}

int main()
{
    cout << func("17 2 3 + / 13 -") << endl;
    cout << func("5 2 3 ^ *") << endl;
    cout << func("2 3 2 ^ ^") << endl;

    cout << "Author: Nero Li\n";
    return 0;
}

```

Input/output below:

```

-10
40
512
Author: Nero Li

```

Answer for Question 1:

There are two steps to handle an expression like $\{(a+25)*c1\}$:

- First, we need to make sure the expression is valid.
 - Check all the braces by a brace stack, push all the begin brace, and when we see the end brace, check the top of the brace stack. If they match, do pop for brace stack. We need to make sure after reading the string, the brace stack is empty.
 - Then, ask the user to input a number when we see a letter in the string. Replace the letter and following characters until the operator or the space and change it to the number that the user input.
- Second, we need to change it from infix to postfix.
 - Use a for loop to check each character in infix string.
 - If the character is a number, make sure next character is not number and create an int value to store this number. When we find a new number after that, value $\times 10 +$ this number. After this process, connect the number onto the postfix string.
 - If the character is an operator, check the operator stack is empty. If it is, push the operator. If it is not, check the operator stack top is not prior to the operator we are looking right now. If it is, pop the operator and connect it onto the postfix string until no prior operator on the top. If it isn't, push the operator into the operator stack.
 - If we see the begin brace operator, push it into the operator stack and it has the highest priority.
 - If we see the end brace operator, pop all the operator inside the operator stack and connect them to the postfix string until the top operator is the begin brace operator. Then pop the begin brace operator but do not connect it.

- After reading the infix string, pop all the operator and connect them onto the postfix string.
- Finally, calculate it as a postfix expression.
 - Use a for loop to check each character in infix string.
 - When we see the number, push it into the number stack.
 - When we see the operator, pop two numbers from the number stack and calculate them. Push the answer into the number stack.
 - After reading the postfix string, pop the number from the number stack, and it will be the answer we want to see for the expression.

Answer for Question 2:

The running time for my postfix evaluation should be $O(n)$. In my function, there is only one for-statement and the running time for this if-statement will depend on how long the postfix equation is. No other nested loop shown so the running time should be $O(n)$ and n is the size of the string, including the spaces.

Extra Credit

Source code below:

```
/* Program: PA_5_extra_credit
   Author: Nero Li
   Class: CSCI 220
   Date: 09/30/2021
   Description:
       Set up a function or static method that receives a string
representing an
       infix expression and it returns an equivalent postfix expression.
Your
       function/method uses a stack to convert an infix expression to a
postfix
       expression.
```

I certify that the code below is my own work.

Exception(s): N/A

```
*/
```

```
#include <iostream>
```

```
using namespace std;
```

```
template <typename T>
```

```
class Stack
```

```
{
```

```
    public:
```

```
        Stack() {}
```

```
        int size()
```

```
        {
```

```
            return i;
```

```
        }
```

```
        bool empty()
```

```
        {
```

```
            if (i == 0)
```

```
            {
```

```
                return true;
```

```
            }
```

```
            return false;
```

```
        }
```

```
        T top()
```

```
        {
```

```
            if (empty())
```

```
            {
```

```

        cout << "StackEmpty\n";
        exit(-1);
    }
    return head->n;
}

void push(T n)
{
    Node *cur = new Node;
    cur->n = n;
    cur->next = head;
    head = cur;
    ++i;
}

T pop()
{
    T lost;
    Node *cur;
    if (empty())
    {
        cout << "StackEmpty\n";
        exit(-1);
    }
    lost = head->n;
    cur = head;
    head = head->next;
    delete(cur);
    --i;
    return lost;
}

private:
    int i{0};
    struct Node
    {
        T n;
        Node *next;
    } *head{NULL};
};

string func(string infix)
{
    Stack<char> symbol;
    string postfix{""};

    for (size_t i = 0; i < infix.size(); ++i)
    {
        if (infix[i] >= '0' && infix[i] <= '9')
        {
            postfix += infix[i];
            if (infix[i + 1] == ' ' || infix[i + 1] == '\\0')

```

```

        {
            postfix += ' ';
        }
    }
    else if (infix[i] == '(' || infix[i] == '[' || infix[i] == '{')
    {
        symbol.push(infix[i++]);
    }
    else if (infix[i] == ')')
    {
        while (symbol.top() != '(')
        {
            postfix += symbol.pop();
            postfix += ' ';
        }
        symbol.pop();
    }
    else if (infix[i] == ']')
    {
        while (symbol.top() != '[')
        {
            postfix += symbol.pop();
            postfix += ' ';
        }
        symbol.pop();
    }
    else if (infix[i] == '}')
    {
        while (symbol.top() != '{')
        {
            postfix += symbol.pop();
            postfix += ' ';
        }
        symbol.pop();
    }
    else if (infix[i] == '^')
    {
        symbol.push(infix[i++]);
    }
    else if (infix[i] == '*' || infix[i] == '/')
    {
        while (!symbol.empty() && (symbol.top() == '^' || symbol.top()
== '*' || symbol.top() == '/'))
        {
            postfix += symbol.pop();
            postfix += ' ';
        }

        symbol.push(infix[i++]);
    }
    else if (infix[i] == '+' || infix[i] == '-')
    {

```

```

        while (!symbol.empty() && (symbol.top() == '^' || symbol.top()
== '*' || symbol.top() == '/' || symbol.top() == '+' || symbol.top() == '-'
'))
        {
            postfix += symbol.pop();
            postfix += ' ';
        }

        symbol.push(infix[i++]);
    }

}

while (!symbol.empty())
{
    postfix += symbol.pop();
    postfix += ' ';
}

return postfix;
}

int main()
{
    cout << func("17 / ( 2 + 3 ) - 13") << endl;
    cout << func("5 * 2 ^ 3") << endl;
    cout << func("2 ^ 3 ^ 2") << endl;

    cout << "Author: Nero Li\n";
    return 0;
}

```

Input/output below:

```

17 2 3 + / 13 -
5 2 3 ^ *
2 3 2 ^ ^
Author: Nero Li

```