Data Structure Program Assignment #5  
(Due: PM: 5:00, April 18, 2022)

**Calculator Design***Instructor: Jiann-Jone Chen*

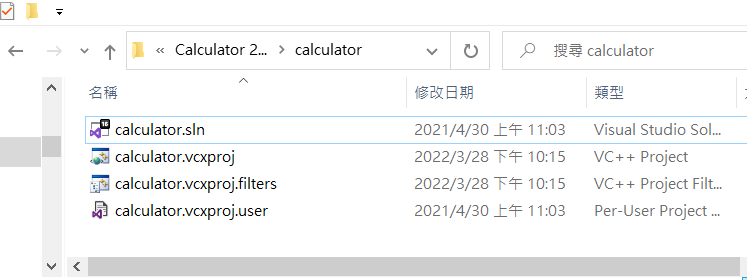
* **Introduction**

In lecture 3, we introduced stacks and queues that can store the same object data in a different order. The stack is used to provide the first-in-last-out function. In this programming homework, you had to reuse the template stack and queue data structure to store tokens extracted from a string of numerical operation expressions. Then the infix to postfix (page52-page57) and evaluation (page 51) functions that we introduced in lecture notes can be used to evaluate the value of an infix expression.

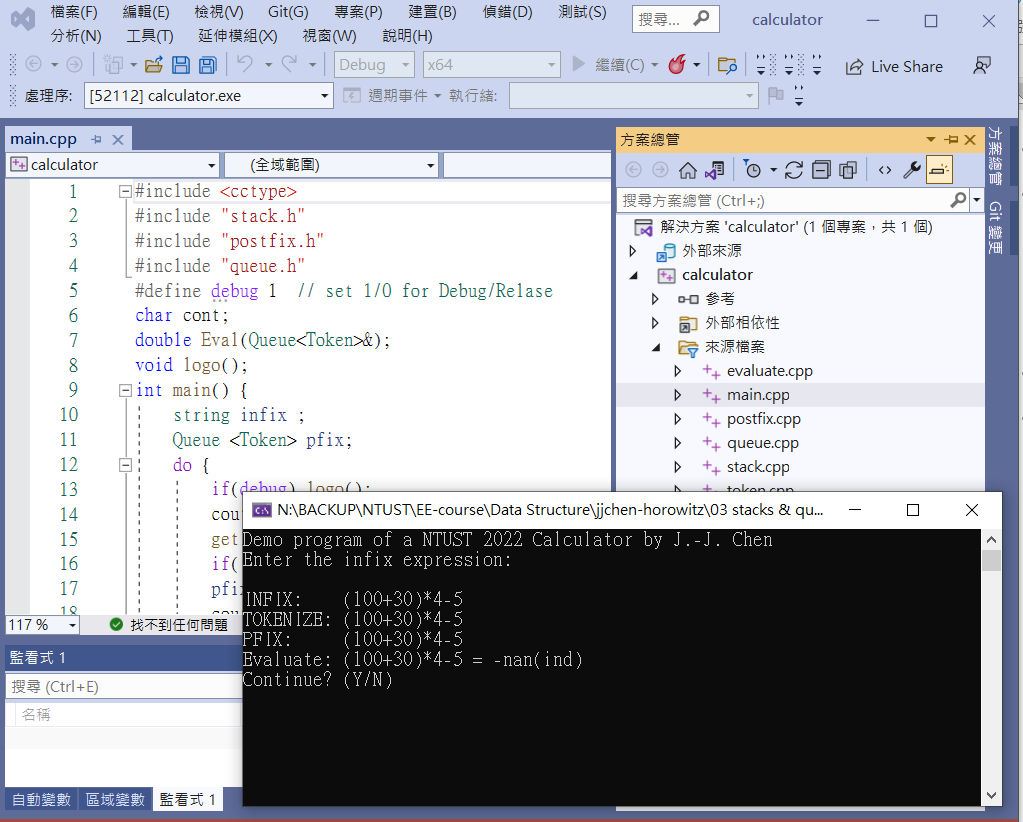
For fairness, you are asked to design the program based on the provided demo project. In addition, you had to use the stack and queue template class that you designed in the previous program homework in this program.

* **Steps**

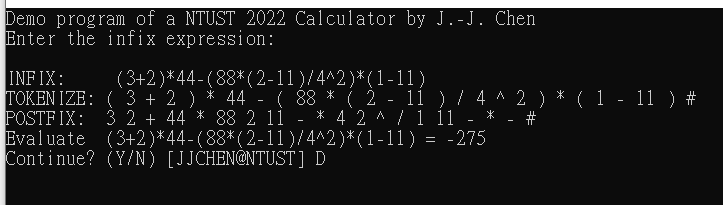
1. A demo project is provided for you to quickly start programming to save your precious study time. Click the calculator.sln by visual studio 2022.



1. Set to Debug Mode and Execute. Key in any infix expression you like, and the result looks like the figure below. Type Y to continue and N to exit. This demo project is provided and you can focus your brainpower on the calculator algorithm design.



1. With your INFIX expression, e, your program should output three parts: (1) TOKENIZ items; (2) POSTFIX: Tokens after the expression ‘e’ is transformed to postfix expression, postfix(string e); (3) Evaluate: The numerical value of your calculator output.



1. When the expression is stored in the string infix, it should be tokenized. For example, when an input expression is 35 + 7, it should be decomposed into three tokens, i.e., 35(operand), ‘+’(operator), and 7(operand). This tokenize function should be designed as:

**#include "queue.h"** // use your own queue.h

**Queue <Token> token = tokenize(string e){ };**

//design the tokenized function

According to the algorithm of our textbook, a ‘#’ symbol is appended as the last token to signal the ending. For example, 35 + 7 should be tokenized into four tokens, i.e., [35] [+] [7] [#].

1. Infix to postfix: (see lecture notes page 56 )  
   **Queue <Token> pfix = postfix(string e);  
   Queue <Token> postfix(string e){ // use your own queue.h for this queue**

**Stack <Token> stack; // use your own stack.h for this stack  
 ….**

**}**

In this postfix function, you had to assign in stack priority (isp) and in-coming priority (icp) for all operators, so that the higher priority operators can be performed before lower priority ones, i.e., icp(‘(‘)=0 and isp(‘(‘)=8. Remember to assign the ‘#’ with the lowest priority 8 and append it at the end of the tokenized postfix expression, so that it can pop out all operators and finish the transformation.

1. Evaluation: (see lecture notes page 51) **double answer = Eval(pfix);**

In this program, you had to use your own template stack class to design and solve this problem.

1. You can start to design your program based on this main file.

#include <cctype>

#include "stack.h"

#include "postfix.h"

#include "queue.h"

#define debug 0 //set 1/0 for Debug/Release mode

char cont;

double Eval(Queue<Token>&);

void logo();

int main() {

string infix ;

Queue <Token> pfix;

system("CLS");

do {

if(debug) logo();

cout << left << setw(10) << "\nINFIX: ";

getline(cin >> ws, infix);

if(!debug) cout << left << infix<<endl;

pfix = postfix(infix);

cout << "\nEvaluate "<< infix << " = " << Eval(pfix) << endl;

cout << "Continue? (Y/N) ";

cin >> cont; cout << (char) toupper(cont)<<endl;

} while (cont == 'Y' || cont == 'y');

return 0;

}

void logo() {

system("CLS");

cout << "Demo program of a NTUST 2022 Calculator by J.-J. Chen\n";

cout << "Enter the infix expression:\n\n";

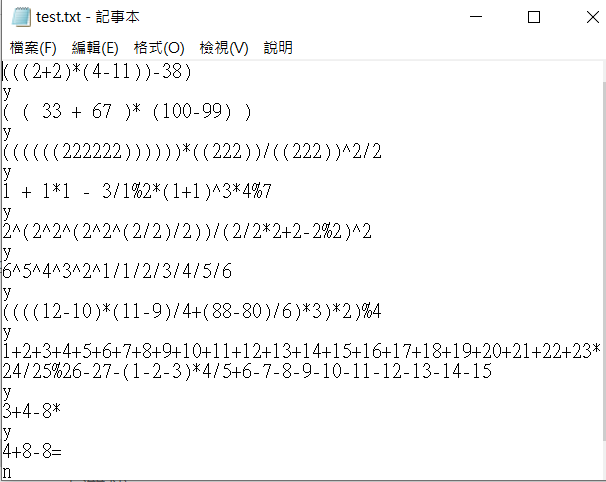
cout << left << setw(10) << "INFIX: ";

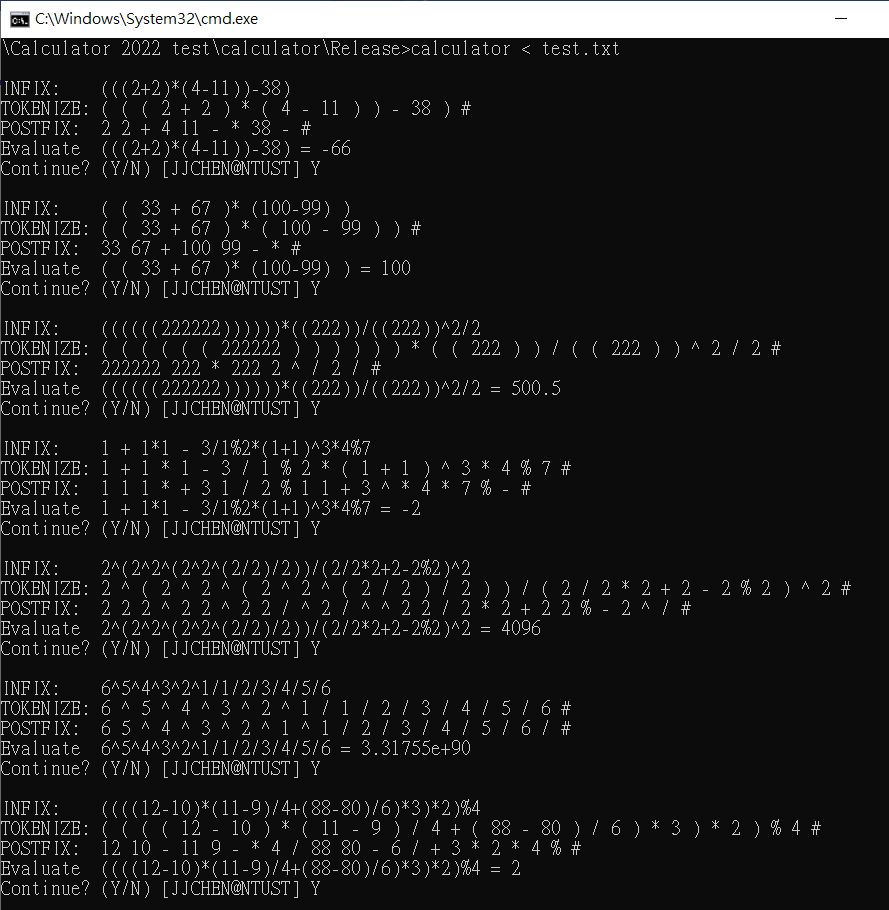
}

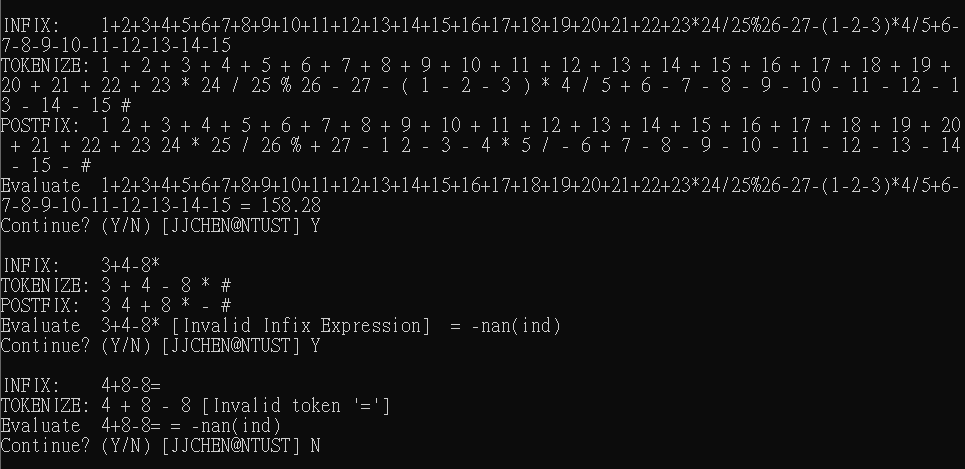
1. Remember to #define debug 1 for debug mode and 0 for Release Mode.



1. An executable calculator.exe and a test.txt file are provided for you to verify the calculator function. Some execution results are shown below.







* **Requirements:**

Some functions are partially finished and you are asked to make the program complete. The execution result is shown in the above figure.

1. You will have to submit the complete project such that the TA can recompile your programs to test correctness.
2. You have to write a short report to describe
   1. What is all about the program?
   2. What functions you have designed to provide a calculator program?
   3. What and how to solve problems during the program design.
   4. How will you improve this program?

* **Criteria**

1. (70%) Basic +, -, \*, / operations are correct
2. (10%) infix expressions that comprise (, ), ^, can be evaluated correctly.
3. (10%) unary operation such as (-3\*5 or 5\*-3) can be recognized and evaluated.
4. (10%) Correctness check of the infix expression. Credits will depend on the correctness check details.

Hint:

1. The **cin >> infix** instruction to read in the expression and it will stop reading any characters after reading a space character, ‘ ‘. For example, the expression 3\*4 + 5 would be read as 3\*4 only. You had better use the instruction below for perfect input:

**getline(cin >> ws, infix);**

1. You had to carefully design copy construction and copy assignment function for template class Queue<Token>.
2. If possible, use dynamic memory allocation for container classes, such as stack and queue class.