Fundamental Computer Programming - C++ Lab(II)

Lab 3: Stack Class for Checking Arithmetic Expression

03/11/2021

Rung-Bin Lin

International Bachelor Program in Informatics
Yuan Ze University

Purposes of this Lab

- **Get you familiar with constructors with arguments.**
- **Get you familiar with separating class definition, class implementation, and application program (main function and other global functions) into different files.**

Time Class Definition for Software Reuse

```
// Fig. 9.10: Time.h, known as a header file that stores the declaration of a class
#ifndef TIME_H    // These two lines and the last line are called preprocessor wrapper.
#define TIME_H    // The class declaration is enclosed between ifndef and endif.
                  // This prevents a header file from being included many times.
// Time abstract data type definition
class Time
{
public:
   Time( int = 0, int = 0, int = 0 ); // default constructor with default argument values
   void setTime( int, int, int ); // set hour, minute, second
   void printUniversal(); // output time in universal-time format
   void printStandard(); // output time in standard-time format
private:
   int hour; // 0 - 23 (24-hour clock format)
   int minute; // 0 - 59
   int second; // 0 - 59
}; // end class Time

#endif
```

Member Functions (Time.cpp)

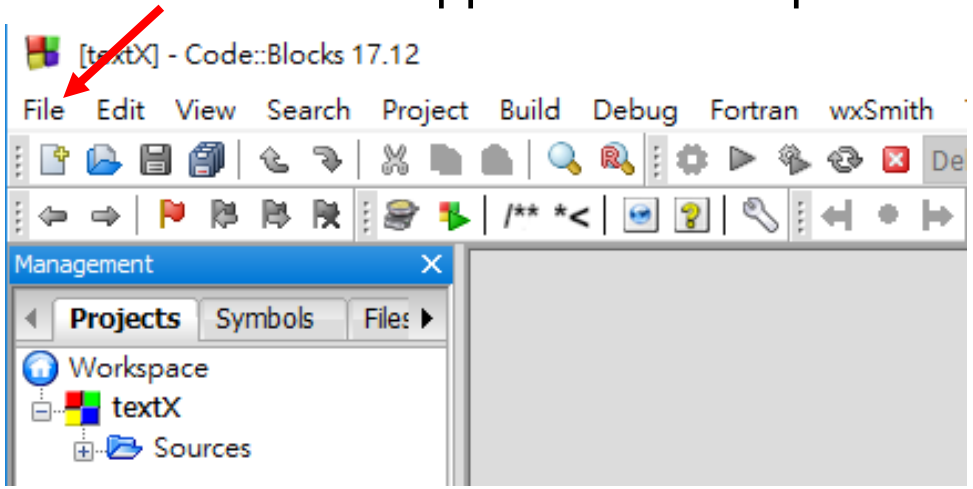
```
// Fig. 9.11: Time.cpp
#include “Time.h”  // you need to know the difference from <Time.h>. See page 416
Time::Time(int hr, int min, int sec){
   setTime(hr, min, sec);
}
void Time::setTime( int h, int m, int s ) {
   hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
   minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
   second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
} // end function setTime
void Time::printUniversal() {
   cout << setfill( '0' ) << setw( 2 ) << hour << ":"
      << setw( 2 ) << minute << ":" << setw( 2 ) << second;
} // end function printUniversal
void Time::printStandard() {
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ) << ":"
      << setfill( '0' ) << setw( 2 ) << minute << ":" << setw( 2 )
      << second << ( hour < 12 ? " AM" : " PM" );
} // end function printStandard
```

Main() Function

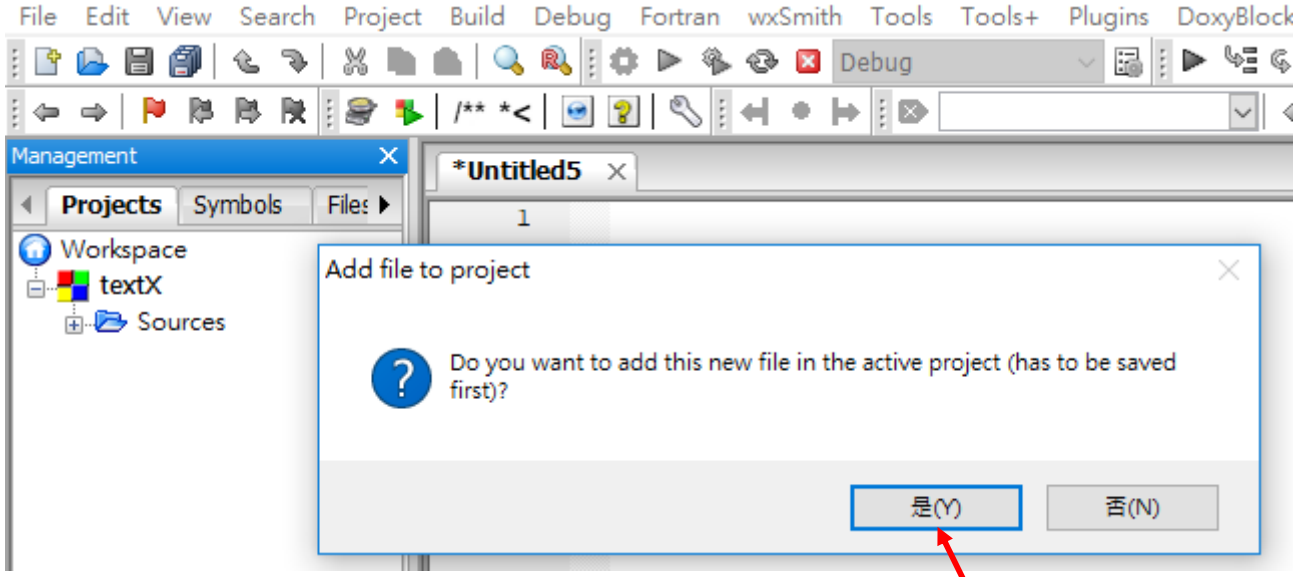
```
// Stored as a file called main.cpp

Time t1; // all arguments defaulted
   Time t2( 2 ); // hour specified; minute and second defaulted
   Time t3( 21, 34 ); // hour and minute specified; second defaulted
   Time t4( 12, 25, 42 ); // hour, minute and second specified
   Time t5( 27, 74, 99 ); // all bad values specified
   cout << "Constructed with:\n\nt1: all arguments defaulted\n  ";
   t1.printUniversal(); // 00:00:00
   cout << "\n  ";
   t1.printStandard(); // 12:00:00 AM

   cout << "\n\nt2: hour specified; minute and second defaulted\n  ";
   t2.printUniversal(); // 02:00:00
   cout << "\n  ";
   t2.printStandard(); // 2:00:00 AM

   cout << "\n\nt3: hour and minute specified; second defaulted\n  ";

   t3.printUniversal(); // 21:34:00
   cout << "\n  ";
   t3.printStandard(); // 9:34:00 PM

   cout << "\n\nt4: hour, minute and second specified\n  ";
   t4.printUniversal(); // 12:25:42
   cout << "\n  ";
   t4.printStandard(); // 12:25:42 PM

   cout << "\n\nt5: all invalid values specified\n  ";
   t5.printUniversal(); // 00:00:00
   cout << "\n  ";
   t5.printStandard(); // 12:00:00 AM
   cout << endl;
} // end main
```

Creating .h and .cpp Files with CodeBlock

- Click “file” on the upper left of the panel.

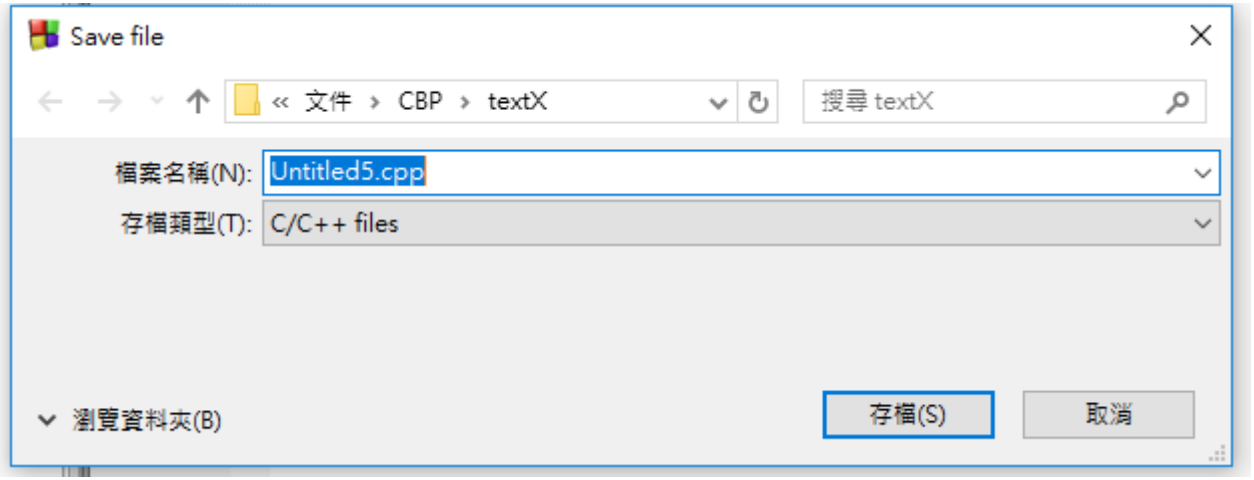


- Select “new”, then select “empty file”

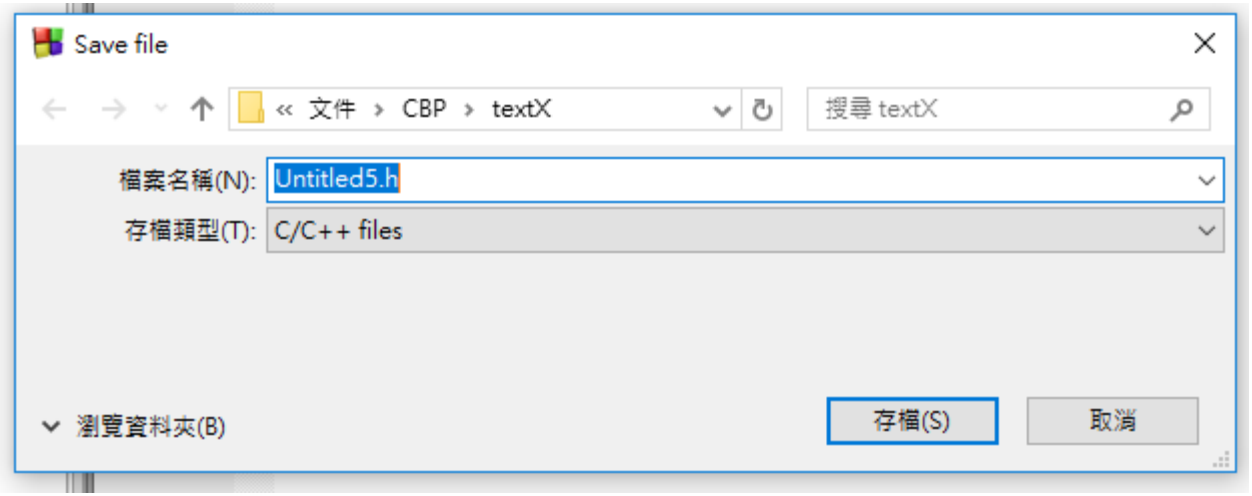


Creating .h and .cpp Files with CodeBlock (cont.)

- Saving a file with extension .cpp will create a source file

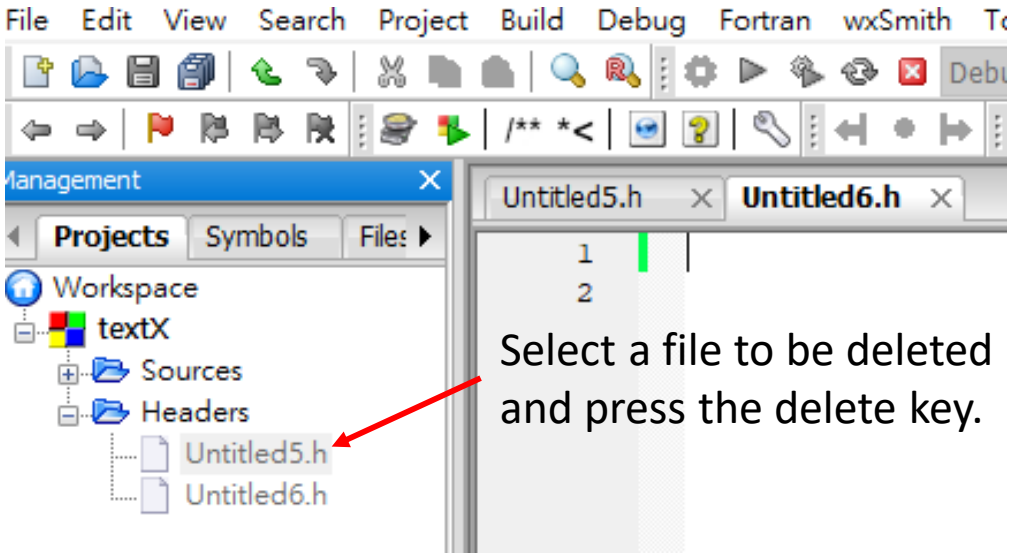


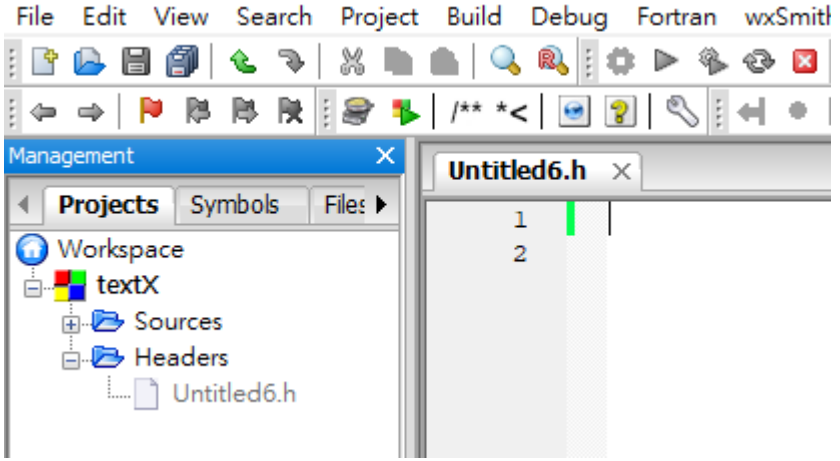
- Saving a file with extension .h will create a header file.



Deleting a file from a Project

- Select the file and press the “delete” on the keyboard.





After pressing the delete key.

Lab 3: Stack Class

- **Create a Stack class of characters**
  - **with the following private data**
    - ✓ int stackSize; // size of a stack
    - ✓ int stackPtr; // the number of characters stored in the stack. It should be also used as an // index into a stack entry where the top most element is located in the stack.
    - ✓ char *charStack; // a character pointer that points to a character array which should be // created dynamically using *new* statement.
  - **with the following member functions**
    - ✓ Stack(int =1, int = 0, char * = NULL);
      // constructor with an argument that will be used to initialize the stack with a given size.
      // When a stack is initially created, the private data members should be initialized.
    - ✓ void push(char); // Push a character onto the top of a stack.
    - ✓ char pop(); // Return the topmost element from a stack
    - ✓ void reset(); // Reset stackPtr to 0. This says that the stack is reset.
    - ✓ bool empty(); // Return true if a stack is empty
    - ✓ bool full(); // Return true if a stack is full

- You should implement an algorithm as a global function **checkExpression(string, Stack &)** to check whether a given arithmetic expression such as (12+3*30)+(20) is correct. The first parameter will hold an expression. The second parameter will be a stack used to process an expression.
  - A correct expression is defined as an expression which contains the same number of (‘s and)’s. That is, if there is a (, the expression should have a corresponding). For example, the above expression is a correct one. The expression)2+3(is incorrect, but (2+3) is correct. Certainly, 2+3 is also correct. An algorithm for solving this problem can be found in Example 12.3 on page 324 in the book, Foundations of Computer Science, Behrouz Forouzan, fourth edition. For your convenience, this algorithm is also given below. Note that this algorithm may not be able to produce the output correctly as shown in the Input & Output example.

Algorithm 12.2: Example 12.3

Algorithm: CheckingParentheses (expression)

Purpose: Check the pairing of parentheses in an expression
Pre: Given the expression to be checked
Post: Error messages if unpaired parentheses are found
Return: None

```
{   stack (S)
    while (more character in the expression) {
        Char ← next character
        if (Char = '(')
            push (S, Char)
        else if ( Char = ')' ) {
                if (empty (S))      // More )'s  than ('s
                    print (unmatched opening parenthesis)
                else
                    pop (S, x)
                }
     } // End while
     if (not empty (S))             // More ('s than )'s
     {
          print (a closing parenthesis not matched)
      }
     return
}
```

Input & Output Formats

- Input format
  - The first line contains a number *m* which specifies the size of a stack being created. The second line specifies the number of expressions that will be checked. Then, it is followed by *n* lines where each line contains an expression.
- Output format
  - If an expression is correct, print “correct” onto a line. If there is more (‘s than)’s being inspected during checking the expression from left to right , print “more (‘s”, otherwise print more “more)’s”. If a stack is not large enough to hold data, you should print “stack full” and terminate checking of this expression. So, the output should have *n* lines where a line contains either correct, incorrect, or stack full. **Note that an escape character \ must be used for printing out ‘ in “more (‘s”.**

Input & Output Example

Input	Output
4	
8	
2*32	**correct**
(12+3*30)+(20)	**correct**
)2+3(	**more)'s**
6+(((((20+2)*5)+2)*6)+7)	**stack full**
2*5(7+23)4()	**correct**
(4+(2*5(7+23))*5	**more ('s**
123	**Correct**
6+((((20+2)*5)+2)*6)+7)	**more)'s**

Note that the test case)2+3(is said to have more)'s than ('s because we need one more (to match a) during inspecting this expression.

Key Points for **Grading**

- The private data members and function members should be exactly the same as that specified in this lab.
- The output should be correct.
- The member function void checkExpression(string, Stack &) should be used for determining the correctness of an expression.
- The program should be able to handle more than one expression at a time.
- The program should be separated into a .h file for class declaration, a .cpp file for class implementation, and a .cpp file for main (application) program.