

Function Overloading & Template

Lab 11: Sorting Data

Rung-Bin Lin

International Bachelor Program in Informatics
Yuan Ze University

December 21, 2021

Purposes

- **Understand the followings**
 - **Function overloading**
 - **Function template**

Function Overloading & Template

• Function overloading

- C++ enables several functions of the same name to be defined, as long as they have different signatures, i.e., different parameters.
- Overloaded functions are normally used to perform similar operations that involve different program logic on different data types.

• If the program logic and operations are identical for each data type, overloading may be performed more compactly and conveniently by using function templates.

Function Overloading

```
1  // Fig. 5.23: fig05_23.cpp
2  // Overloaded functions.
3  #include <iostream>
4  using namespace std;
5
6  // function square for int values
7  int square( int x )
8  {
9      cout << "square of integer " << x << " is ";
10     return x * x;
11 } // end function square with int argument
12
13 // function square for double values
14 double square( double y )
15 {
16     cout << "square of double " << y << " is ";
17     return y * y;
18 } // end function square with double argument
19
```

Fig. 5.23 | Overloaded square functions. (Part I of 2.)

Function Overloading cont.

```
20  int main()
21  {
22      cout << square( 7 ); // calls int version
23      cout << endl;
24      cout << square( 7.5 ); // calls double version
25      cout << endl;
26  } // end main
```

square of integer 7 is 49
square of double 7.5 is 56.25

Fig. 5.23 | Overloaded square functions. (Part 2 of 2.)

Function Template

```
1 // Fig. 5.25: maximum.h
2 // Definition of function template maximum.
3 template < class T > // or template< typename T >
4 T maximum( T value1, T value2, T value3 )
5 {
6     T maximumValue = value1; // assume value1 is maximum
7
8     // determine whether value2 is greater than maximumValue
9     if ( value2 > maximumValue )
10         maximumValue = value2;
11
12     // determine whether value3 is greater than maximumValue
13     if ( value3 > maximumValue )
14         maximumValue = value3;
15
16     return maximumValue;
17 } // end function template maximum
```

Fig. 5.25 | Function template maximum header file.

Function Template cont2.

```
1 // Fig. 5.26: fig05_26.cpp
2 // Function template maximum test program.
3 #include <iostream>
4 #include "maximum.h" // include definition of function template maximum
5 using namespace std;
6
7 int main()
8 {
9     // demonstrate maximum with int values
10    int int1, int2, int3;
11
12    cout << "Input three integer values: ";
13    cin >> int1 >> int2 >> int3;
14
15    // invoke int version of maximum
16    cout << "The maximum integer value is: "
17         << maximum( int1, int2, int3 );
18
19    // demonstrate maximum with double values
20    double double1, double2, double3;
21
22    cout << "\n\nInput three double values: ";
23    cin >> double1 >> double2 >> double3;
24
```

← **T** now is replaced by **int**.

Function Template cont2.

```
25 // invoke double version of maximum
26 cout << "The maximum double value is: "
27     << maximum( double1, double2, double3 );
28
29 // demonstrate maximum with char values
30 char char1, char2, char3;
31
32 cout << "\n\nInput three characters: ";
33 cin >> char1 >> char2 >> char3;
34
35 // invoke char version of maximum
36 cout << "The maximum character value is: "
37     << maximum( char1, char2, char3 ) << endl;
38 } // end main
```

T now is replaced by **double**.

T now is replaced by **char**.

Input three integer values: 1 2 3
The maximum integer value is: 3

Input three double values: 3.3 2.2 1.1
The maximum double value is: 3.3

Input three characters: A C B
The maximum character value is: C

Fig. 5.26 | Demonstrating function template maximum. (Part 2 of 2.)

LAB 11: Sorting

• Part I (70%)

- Use function overloading to sort a list of integers, real numbers, or strings in descending order. Any sorting function can be used. However, you have to write your own code for sorting. The strings are ordered in their lexicographic order. For example, $A < a$, $a < b$, $aa < ab$, $aa < aaa$, etc. Basically, you can use the operator “>” to compare two strings to determine their lexicographic order. check

https://en.wikipedia.org/wiki/Lexicographic_order for details.

- For this, you have to write three functions using the same name as follows:
void sortFunc(int [], int);
void sortFunc(double [], int);
void sortFunc(string [], int);
- The first parameters in these three functions are respectively an integer array, a double precision number array, and a string array. The second parameter gives the number of elements in the array.

● Part II (30%)

- Implement the following function template for sorting data in **ascending order**.



template <class T>



void sortFuncTempt(T anAry[], int numElm);

The first parameter anAry[] is an array that stores the data being sorted. The second parameter is the number of elements actually stored in the array.

- **NOTE:** A template must be defined before it is used. So it has to be placed at a position before main() function. Also if we have a function template, we should not include its function prototype in the program. Please refer to the example in Fig. 5.25 and Fig. 5.26.

main() Function

- The main function is partly provided to you. The two places each highlighted with a  should be inserted with proper code to generate required output. The code should include both sortFunc() and sortFuncTemp(). The code you need add should be similar to the part highlighted with a .

```
int main ()
{
    int numTest;
    int intList[100];
    double doubleList[100];
    string strList[100];
    int numElm;
    string dataType;
    cin >> numTest;
    for(int k=0; k<numTest; k++){
        cin >> dataType;
        cin >> numElm;
        if(dataType == "int"){
 Place some code here
        }
        else if(dataType == "double"){
 Place some code here
        }
        else {
            for (int m=0; m<numElm; m++)
                cin >> strList[m];
            sortFunc(strList, numElm);
            printArray(strList, numElm);
            sortFuncTemp(strList, numElm);
            printArray(strList, numElm);
        }
    }
    return 0;
}
```

printArray(T [], int)

- This template is provided to you for printing out an array. You can use it as an example for developing the template for sortFuncTempt(...).

```
template <class T>
void printArray(T elmList, int numElm)
{
    for(int i=0; i<numElm; i++){
        if( i%10 == 0 )
            cout << "#";
        cout << " " << elmList[i];
        if( (i+1)%10 == 0 )
            cout << endl;
    }
    if(numElm%10 != 0)
        cout << endl;
}
```

Input & Output

• Input format

- The first line specifies the number of test cases. Starting from the second line, input data for each test case are presented.
- The first line of each test case specifies the data type of elements being read. It is *int* for integer data type, *double* for double precision data type, and *string* for string data type. The second line specifies the number of elements given for the test case. After this, each line gives data being stored in the array. The data for each test case may take more than one line. The number of elements for a test case is at most 100.

• Output format

- The first line of the output for each test case should be **TEST x:** where x is the test case number. Then, the sorted items are presented line by line, each line containing 10 elements.

Sample Input & Output

Input	Output
3	# 5 4 4 4 4 4 3 3 2
int	# 2 1 1 0 -1 -1 -3 -3 -4 -4
21	# -9
4 4 4 4 4 1 2 2 3 3 -1 -3	# -9 -4 -4 -3 -3 -1 -1 0 1 1
0 -9 -4 -3 -4 -1 1 5 4	# 2 2 3 3 4 4 4 4 4 4
double	# 5
14	
0.1 0.5 0.4 0.2 0.4 0.2 -0.1 0.1	# 3.2 1.4 1.2 0.5 0.41 0.4 0.4 0.2 0.2 0.1
0.41 1.2 3.2 -1 -5 1.4	# 0.1 -0.1 -1 -5
string	# -5 -1 -0.1 0.1 0.1 0.2 0.2 0.4 0.4 0.41
14	# 0.5 1.2 1.4 3.2
Bye Ace bike corE core bye fit Fet	
kit Kitty zip Yam ulm via	# zip via ulm kit fit core corE bye bike Yam
	# Kitty Fet Bye Ace
	# Ace Bye Fet Kitty Yam bike bye corE core fit
	# kit ulm via zip

Requirements for Lab & TA Grading

- Should have the following three functions implemented.

void sortFunc(int [], int);

void sortFunc(double [], int);

void sortFunc(string [], int);

These three functions sort data in **descending order**.

- Should have the following template implemented.

template <class T>

void sortFuncTempt(T anAry[], int numElm);

This function template sorts data in **ascending order**.

- Should not add any other functions.
- Should include both sortFunc() and sortFuncTempt() in the inserted code in the main() function.