Function Overloading & Template

Lab 11: Finding Objects at 1/3 List

Rung-Bin Lin

International Bachelor Program in Informatics
Yuan Ze University

December 5, 2023

Purposes

- Understand the followings
  - Function overloading (5.18)
  - Function template (5.19)

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 with function templates.

Function Overloading

```cpp
// Fig. 5.23: fig05_23.cpp
// Overloaded functions.
#include <iostream>
using namespace std;

// function square for int values
int square( int x )
{
   cout << "square of integer " << x << " is ";
   return x * x;
} // end function square with int argument

// function square for double values
double square( double y )
{
   cout << "square of double " << y << " is ";
   return y * y;
} // end function square with double argument

```

Fig. 5.23 | Overloaded `square` functions. (Part 1 of 2.)

Function Overloading cont.

```cpp
int main()
{
    cout << square( 7 ); // calls int version
    cout << endl;
    cout << square( 7.5 ); // calls double version
    cout << endl;
} // 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

```cpp
// Fig. 5.25: maximum.h
// Definition of function template maximum.
template < class T >  // or template< typename T >
T maximum( T value1, T value2, T value3 )
{
   T maximumValue = value1; // assume value1 is maximum

   // determine whether value2 is greater than maximumValue
   if ( value2 > maximumValue )
      maximumValue = value2;

   // determine whether value3 is greater than maximumValue
   if ( value3 > maximumValue )
      maximumValue = value3;

   return maximumValue;
} // end function template maximum
```

Fig. 5.25 | Function template `maximum` header file.

Function Template cont2.

```
// Fig. 5.26: fig05_26.cpp
// Function template maximum test program.
#include <iostream>
#include "maximum.h" // include definition of function template maximum
using namespace std;

int main()
{
   // demonstrate maximum with int values
   int int1, int2, int3;

   cout << "Input three integer values: ";
   cin >> int1 >> int2 >> int3;

   // invoke int version of maximum
   cout << "The maximum integer value is: "
      << maximum( int1, int2, int3 );

   // demonstrate maximum with double values
   double double1, double2, double3;

   cout << "\n\nInput three double values: ";
   cin >> double1 >> double2 >> double3;

```

T now is replaced by **int.**

Fig. 5.26 | Demonstrating function template `maximum`. (Part 1 of 2.)

Function Template cont2.

```
    // invoke double version of maximum
    cout << "The maximum double value is: "
       << maximum( double1, double2, double3 );

    // demonstrate maximum with char values
    char char1, char2, char3;

    cout << "\n\nInput three characters: ";
    cin >> char1 >> char2 >> char3;

    // invoke char version of maximum
    cout << "The maximum character value is: "
       << maximum( char1, char2, char3 ) << endl;
} // 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: Finding Objects at 1/3

- **Part I (70%)**
  - Use function overloading to find the 1/3 objects from a list of *n* objects which can be all integers, all real numbers, or all strings. A 1/3 object of a list of n objects is the object that there are exactly $\lceil n/3 \rceil - 1$ objects smaller than it. You should also count the number of 1/3 objects. If there is no 1/3 object, treat the maximum object in the list as the 1/3 objects. The strings should be ordered in their lexicographic order (check https://en.wikipedia.org/wiki/Lexicographic_order for details).
  - For example, the 1/3 object of 3, 5, 5, 6, 2, 8, 8 is 5 whereas it is 4 if the list is 3, 4, 10, 5, 6, 2, 8, 8. Yet another example, the 1/3 object of 1, 12, 12, 13, 2, 2, 13 is 13 because there is no 1/3 object. For this, you have to write three functions using the same name as follows:

```
int& find13(int [ ], int, int &);
double& find13(double [ ], int, int&);
string& find13(string [ ], int, 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. The third parameter is the number of 1/3 objects in the array (list). The returned object should be the 1/3 object.

Part II (30%)

- Implement the following function template for find13.

```
template <class T>
T find13Tempt(T anAry[], int numElm, int &num13Obj);
```

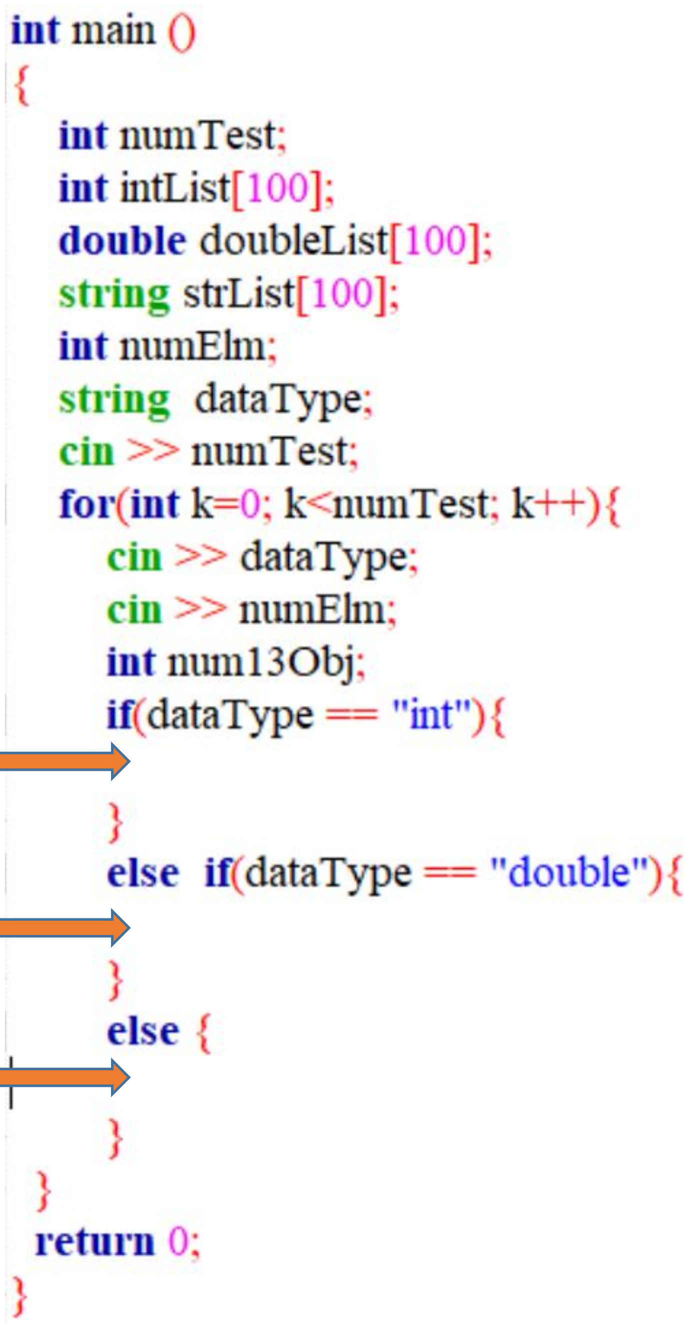
The first parameter anAry[] is an array that stores the objects being processed. The second parameter is the number of objects actually stored in the array. The third parameter is the number of 1/3 objects. The returned value should be the 1/3 object of a given list.

- NOTE: A template must be defined before it is used. So it must 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 three places each highlighted with a → should be inserted with proper code to generate required output. The code should include both find13(…) and find13Tempt(…).

Insert the required code in each of these three sections. You should not use any more variables.



```
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;
        int num13Obj;
        if(dataType == "int"){

        }
        else  if(dataType == "double"){

        }
        else {

        }
    }
  return 0;
}
```

Input & Output

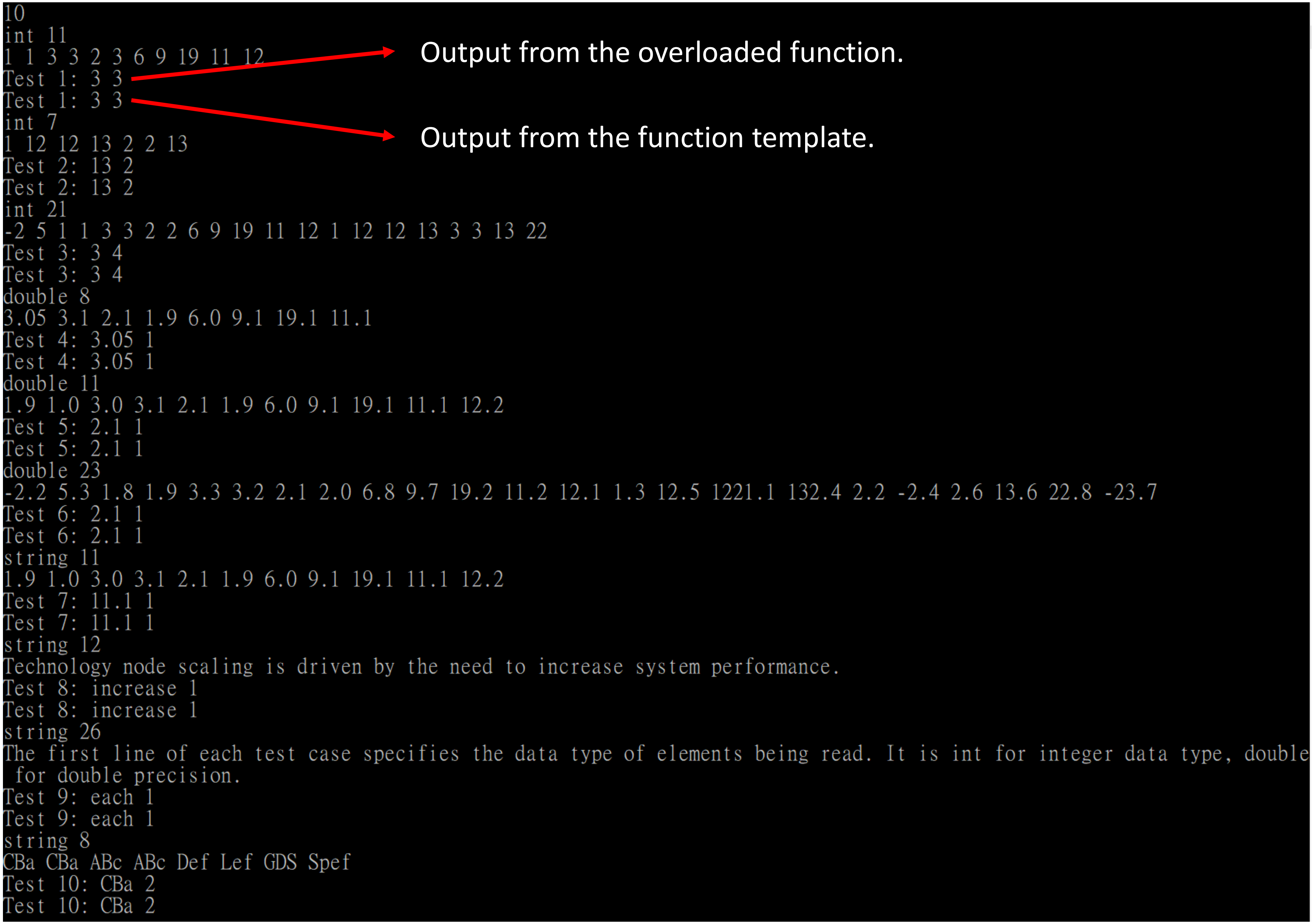
- **Input format**
  - **The first line specifies the number of test cases. Starting from the second line, input data for each test case are presented.**
  - **The first line of each test case has two items. The first item specifies the data type of elements being read. It is *int* for integer data type, *double* for double precision data type, and *string* for string data type. The second item specifies the number of elements given for the test case. After this, each line gives the list of data. 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 output for each test case has two lines. The first line presents the output generated by overloaded function. It should be Test x: where x is the test case number. Then, the 1/3 object is presented. It is then the number of 1/3 objects. The second line has the same format as the first line, which is the output generated by the corresponding function template.**

Sample Input

Sample Input

```
10
int 11
1 1 3 3 2 3 6 9 19 11 12
int 7
1 12 12 13 2 2 13
int 21
-2 5 1 1 3 3 2 2 6 9 19 11 12 1 12 12 13 3 3 13 22
double 8
3.05 3.1 2.1 1.9 6.0 9.1 19.1 11.1
double 11
1.9 1.0 3.0 3.1 2.1 1.9 6.0 9.1 19.1 11.1 12.2
double 23
-2.2 5.3 1.8 1.9 3.3 3.2 2.1 2.0 6.8 9.7 19.2 11.2 12.1 1.3 12.5 1221.1 132.4 2.2 -2.4 2.6 13.6 22.8 -23.7
string 11
1.9 1.0 3.0 3.1 2.1 1.9 6.0 9.1 19.1 11.1 12.2
string 12
Technology node scaling is driven by the need to increase system performance.
string 26
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.
string 8
CBa CBa ABc ABc Def Lef GDS Spef
```

Sample Input & Output



```
10
int 11
1 1 3 3 2 3 6 9 19 11 12
Test 1: 3 3
Test 1: 3 3
int 7
1 12 12 13 2 2 13
Test 2: 13 2
Test 2: 13 2
int 21
-2 5 1 1 3 3 2 2 6 9 19 11 12 1 12 12 13 3 3 13 22
Test 3: 3 4
Test 3: 3 4
double 8
3.05 3.1 2.1 1.9 6.0 9.1 19.1 11.1
Test 4: 3.05 1
Test 4: 3.05 1
double 11
1.9 1.0 3.0 3.1 2.1 1.9 6.0 9.1 19.1 11.1 12.2
Test 5: 2.1 1
Test 5: 2.1 1
double 23
-2.2 5.3 1.8 1.9 3.3 3.2 2.1 2.0 6.8 9.7 19.2 11.2 12.1 1.3 12.5 1221.1 132.4 2.2 -2.4 2.6 13.6 22.8 -23.7
Test 6: 2.1 1
Test 6: 2.1 1
string 11
1.9 1.0 3.0 3.1 2.1 1.9 6.0 9.1 19.1 11.1 12.2
Test 7: 11.1 1
Test 7: 11.1 1
string 12
Technology node scaling is driven by the need to increase system performance.
Test 8: increase 1
Test 8: increase 1
string 26
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.
Test 9: each 1
Test 9: each 1
string 8
CBa CBa ABc ABc Def Lef GDS Spef
Test 10: CBa 2
Test 10: CBa 2
```

Output from the overloaded function.
Output from the function template.

Requirements for Lab & TA Grading

- **Should not sort the list of objects.**
- Should have the following three functions implemented.

```
int find13(int [ ], int, int&);
double find13(double [ ], int, int &);
string find13(string [ ], int, int &);
```

- Should have the following template implemented.

```
template <class T>
T find13Tempt(T anAry[], int numElm, int &num13Obj);
```

- Should not add any other functions.
- Should include both find13() and find13Tempt() in the inserted code sections of the main() function.