18.1 Function templates

Multiple functions may be nearly identical, differing only in their data types, as below.

Figure 18.1.1: Functions may have identical behavior, differing only in data types.

```
// Find the minimum of three **ints**
int TripleMinInt(int item1, int item2, int item3) {
   int minVal;

   minVal = item1;

   if (item2 < minVal) {
      minVal = item2;
   }
   if (item3 < minVal) {
      minVal = item3;
   }
   return minVal;
}
```

```
// Find the minimum of three **chars**
char TripleMinChar(char item1, char item2, char 
item3) {
   char minVal;

   minVal = item1;

   if (item2 < minVal) {
      minVal = item2;
   }
   if (item3 < minVal) {
      minVal = item3;
   }
   return minVal;
}
```

Writing and maintaining redundant functions that only differ by data type can be time-consuming and error-prone. The language supports a better approach.

A ***function template*** is a function definition having a special type parameter that may be used in place of types in the function.

Figure 18.1.2: A function template enables a function to handle various data types.

```
#include <iostream>
#include <string>
using namespace std;

template<typename TheType>
TheType TripleMin(TheType item1, TheType item2, TheType item3) {
   TheType minVal = item1; // Holds min item value, init to first item

   if (item2 < minVal) {
      minVal = item2;
   }
   if (item3 < minVal) {
      minVal = item3;
   }

   return minVal;
}

int main() {
   int num1 = 55;       // Test case 1, item1
   int num2 = 99;       // Test case 1, item2
   int num3 = 66;       // Test case 1, item3

   char let1 = 'a';     // Test case 2, item1
   char let2 = 'z';     // Test case 2, item2
   char let3 = 'm';     // Test case 2, item3

   string str1 = "zzz";    // Test case 3, item1
   string str2 = "aaa";    // Test case 3, item2
   string str3 = "mmm";    // Test case 3, item3

   // Try TripleMin function with ints
   cout << "Items: " << num1 << " " << num2 << " " << num3 << endl;
   cout << "Min: " << TripleMin(num1, num2, num3) << endl << endl;

   // Try TripleMin function with chars
   cout << "Items: " << let1 << " " << let2 << " " << let3 << endl;
   cout << "Min: " << TripleMin(let1, let2, let3) << endl << endl;

   // Try TripleMin function with strings
   cout << "Items: " << str1 << " " << str2 << " " << str3 << endl;
   cout << "Min: " << TripleMin(str1, str2, str3) << endl << endl;

   return 0;
}
```

```
Items: 55 99 66
Min: 55

Items: a z m
Min: a

Items: zzz aaa mmm
Min: aaa
```

The function return type is preceded by `template<typename TheType>`, where TheType can be any identifier. That type is known as a ***type parameter*** and can be used throughout the function for any parameter types, return types, or local variable types. The identifier is known as a template parameter, and may be various reference types or even another template parameter.

The compiler automatically generates a unique function definition for each type appearing in function calls to the function template. Thus, the above example's calls would create three TripleMin() function definitions using int, char, and string as in this section's introductory example. The programmer never sees those function definitions.

PARTICIPATION ACTIVITY 18.1.1: Function templates.

1) Fill in the blank.

```
template<typename MyType>
_____ MyAvg3 (MyType i, MyType
j, MyType k) {
   return (i + j + k)/3;
}
```

- ○ TheType
- ○ int
- ○ MyType

2) Fill in the blank.

```
template<typename _________>
T TripleMin(T item1, T item2, T
item3) { ... }
```

- ○ int
- ○ TheType
- ○ T
- ○ Not possible; T is not a valid type.

3) For the earlier TripleMin function template, what happens if a call is TripleMin(i, j, k) but those arguments are of type long long?

- ○ The compiler generates an error message because only int and char are supported.
- ○ During runtime, the long long values are forced to be int values.
- ○ The compiler creates function with long long types and calls that function.

4) For the earlier TripleMin function template, what happens if a call is TripleMin(i, j, k) but those arguments are of type string?

- ○ string is just another type, so the function will compare strings.
- ○ The compiler generates an error, because only numerical types can be passed.

5) For the earlier TripleMin function template, what happens if a call is TripleMin(i, j, z), where i and j are ints, but z is a string?

- ○ The function will compare the ints and the string.
- ○ The compiler will generate an error, because TheType must be the same for all three arguments.

Programmers optionally may explicitly specify the type as a special argument, as in `TripleMin<int>(num1, num2, num3);`.

A function template may have multiple parameters:

Construct 18.1.1: Function template with multiple parameters.

```
template<typename T1, typename T2>
ReturnType FunctionName(Parameters)
{
   ...
}
```

Earlier versions of C++ used the word "class" rather than "typename". Though misleading (the type need not be a class, but can be int or double, for example), the word class is still allowed for backwards compatibility, and much existing C++ code uses that word.

zyDE 18.1.1: Function templates with multiple parameters.

This program currently fails to compile. Modify TripleMin() so that item1 can be of a different type than item2 and item3.

Load default template...

Run

```

#include <iostream>
using namespace std;

template<typename TheTyp
TheType TripleMin(TheTyp
   TheType minVal = iten

   if (item2 < minVal) {
      minVal = item2;
   }
   if (item3 < minVal) {
      minVal = item3;
   }

```

Exploring further:

- Templates from cplusplus.com
- Function templates from msdn.microsoft.com

18.2 Class templates

Multiple classes may be nearly identical, differing only in their data types. The following shows a class managing three int numbers, and a nearly identical class managing three short numbers.

Figure 18.2.1: Classes may be nearly identical, differing only in data type.

```
class TripleInt {
public:
   TripleInt(int val1 = 0, int val2 = 0, int val3 = 0);
   void PrintAll() const; // Print all data member values
   int MinItem() const;   // Return min data member value
private:
   int item1;             // Data value 1
   int item2;             // Data value 2
   int item3;             // Data value 3
};

TripleInt::TripleInt(int i1, int i2, int i3) {
   item1 = i1;
   item2 = i2;
   item3 = i3;
}

// Print all data member values
void TripleInt::PrintAll() const {
   cout << "(" << item1 << "," << item2
        << "," << item3 << ")" << endl;
}

// Return min data member value
int TripleInt::MinItem() const {
   int minVal;

   minVal = item1; // Holds min item value, init to first
item

   if (item2 < minVal) {
      minVal = item2;
   }
   if (item3 < minVal) {
      minVal = item3;
   }

   return minVal;
}
```

```
class TripleShort {
public:
   TripleShort(short val1 = 0, short val2 = 0, short val3 =
0);
   void PrintAll() const; // Print all data member values
   short MinItem() const; // Return min data member value
private:
   short item1;           // Data value 1
   short item2;           // Data value 2
   short item3;           // Data value 3
};

TripleShort::TripleShort(short i1, short i2, short i3) {
   item1 = i1;
   item2 = i2;
   item3 = i3;
}

// Print all data member values
void TripleShort::PrintAll() const {
   cout << "(" << item1 << "," << item2
        << "," << item3 << ")" << endl;
}

// Return min data member value
short TripleShort::MinItem() const {
   short minVal;

   minVal = item1; // Holds min item value, init to first
item

   if (item2 < minVal) {
      minVal = item2;
   }
   if (item3 < minVal) {
      minVal = item3;
   }

   return minVal;
}
```

Writing and maintaining redundant classes that only differ by data type can be time-consuming and error-prone. The language supports a better approach.

A ***class template*** is a class definition having a special type parameter that may be used in place of types in the class. A variable declared of that class type must indicate a specific type.

Figure 18.2.2: A class template enables one class to handle various data types.

```
#include <iostream>
using namespace std;

template<typename TheType>
class TripleItem {
public:
   TripleItem(TheType val1 = 0, TheType val2 = 0, TheType val3 = 0);
   void PrintAll() const;   // Print all data member values
   TheType MinItem() const; // Return min data member value
private:
   TheType item1;           // Data value 1
   TheType item2;           // Data value 2
   TheType item3;           // Data value 3
};

template<typename TheType>
TripleItem<TheType>::TripleItem(TheType i1, TheType i2, TheType i3) {
   item1 = i1;
   item2 = i2;
   item3 = i3;
}

// Print all data member values
template<typename TheType>
void TripleItem<TheType>::PrintAll() const {
   cout << "(" << item1 << "," << item2
        << "," << item3 << ")" << endl;
}

// Return min data member value
template<typename TheType>
TheType TripleItem<TheType>::MinItem() const {
   TheType minVal = item1; // Holds value of min item, init to first item

   if (item2 < minVal) {
      minVal = item2;
   }
   if (item3 < minVal) {
      minVal = item3;
   }

   return minVal;
}

int main() {
   TripleItem<int> triInts(9999, 5555, 6666); // TripleItem class with ints
   TripleItem<short> triShorts(99, 55, 66);   // TripleItem class with
shorts

   // Try functions from TripleItem
   triInts.PrintAll();
   cout << "Min: " << triInts.MinItem() << endl << endl;

   triShorts.PrintAll();
   cout << "Min: " << triShorts.MinItem() << endl << endl;

   return 0;
}
```

```
(9999,5555,6666)
Min: 5555

(99,55,66)
Min: 55
```

The class declaration is preceded by `template<typename TheType>`, where TheType can be any identifier. That type is known as a ***template parameter*** and can be used throughout the class, such as for parameter types, function return types, or local variable types. The identifier is known as a template parameter, and may be various items such as an int, double, char, or string, or a pointer or reference, or even another template parameter.

Any of the class's functions defined outside the class declaration must also be preceded by the template declaration, and have the type in angle brackets appended to its name as in `void TripleItem<TheType>::Print()`. An object of this class can be declared by appending after the class name a specific type in angle brackets, such as `TripleItem<short> triShorts(99,55,66);`.

PARTICIPATION ACTIVITY 18.2.1: Class templates.

1) A class has been defined using the type GenType throughout, where GenType is intended to be chosen by the programmer when declaring a variable of this class. The code that should immediately precede the class definition is `template<GenType>`

- True
- False

2) A key advantage of a class template is relieving the programmer from having to write redundant code that differs only by type.

- True
- False

3) For a class template defined as `template<typename T> class Vehicle { … }` an appropriate variable declaration of that class would be `Vehicle<T> v1;`

- ○ True
- ○ False

A template may have multiple parameters, separated by commas:

Construct 18.2.1: Class template with multiple parameters.

```
template<typename T1, typename
T2>
class ClassName {
   ...
};
```

Earlier versions of C++ used the word "class" rather than "typename". Though misleading (the type need not be a class, but can be int or double, for example), the word class is still allowed for backwards compatibility, and much existing C++ code uses that word.

Note that C++'s vector class is a class template, which is why a variable declared as a vector indicates the type in angle brackets, as in `vector<int> nums(100);`.

zyDE 18.2.1: Class templates.

Modify the TimeHrMn class to utilize a class template. Note that the main() function passes int and double as parameters for the SetTime() member function.

Load default template... Run

```
#include <iostream>
using namespace std;

class TimeHrMn {
public:
   void SetTime(int user
   void PrintTime() cons
private:
   int hrsVal;
   int minsVal;
};

void TimeHrMn::SetTime(i
   minsVal = userMin;
```

Exploring further:

- Templates from cplusplus.com

18.3 C++ example: Map values using a function template

zyDE 18.3.1: Map a value using a function template.

The program below uses a function template to map numeric, string, or character values to a shorter list of values. The program demonstrates a mapping for integers using a table of:

```
100
200
300
400
500
600
```

The program gets an integer value from a user and returns the first value in the table that is greater than or equal to the user value, or the user value itself if that value is greater than the largest value in the table. Ex:

```
165 returns 200
444 returns 500
888 returns 888
```

1. Run the program and notice the input value 137 is mapped to 200. Try changing the input value and running again.
2. Modify the program to call the GetMapping function for a double and a string, similar to the integer.
3. Run the program again and enter an integer, a double, and a string

Load default template...

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

template<typename MapType>
MapType GetMapping(MapType mapMe, vector<MapType> map
   MapType result = mapMe;
   unsigned int i;
   bool keepLooking;

   keepLooking = true;

   cout << endl;
   cout << "Mapping range: ";
```

```
137
```

Run

zyDE 18.3.2: Map a value using a function template (solution).

A solution to the above problem follows.

Load default template...

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

template<typename MapType>
MapType GetMapping(MapType mapMe, vector<MapType> map
   MapType result = mapMe;
   unsigned int i;
   bool keepLooking;

   keepLooking = true;

   cout << endl;
   cout << "Mapping range: ";
```

```
137
4.44444
Hi
```

Run

18.4 LAB: What order? (function templates)

Define a generic function called CheckOrder() that checks if four items are in ascending, neither, or descending order. The function should return -1 if the items are in ascending order, 0 if the items are unordered, and 1 if the items are in descending order.

The program reads four items from input and outputs if the items are ordered. The items can be different types, including integers, strings, characters, or doubles.

Ex. If the input is:

```
bat hat mat sat
63.2 96.5 100.1 123.5
```

the output is:

```
Order: -1
Order: -1
```

LAB ACTIVITY 18.4.1: LAB: What order? (function templates) 0 / 10

main.cpp

Load default template...

```
#include <string>
#include <iostream>

using namespace std;

// TODO: Define a generic method called CheckOrder() that
//       takes in four variables of generic type as arguments.
//       The return type of the method is integer

   // Check the order of the input: return -1 for ascending,
   // 0 for neither, 1 for descending


int main() {
   // Read in four strings
```

Develop mode Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → main.cpp (Your program) → Output

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

18.5 LAB: Zip code and population (class templates)

Define a class `StatePair` with two template types (`T1` and `T2`), constructors, mutators, accessors, and a PrintInfo() method. Three vectors have been pre-filled with StatePair data in main():

- `vector<StatePair <int, string>> zipCodeState`: ZIP code - state abbreviation pairs
- `vector<StatePair<string, string>> abbrevState`: state abbreviation - state name pairs
- `vector<StatePair<string, int>> statePopulation`: state name - population pairs

Complete main() to use an input ZIP code to retrieve the correct state abbreviation from the vector zipCodeState. Then use the state abbreviation to retrieve the state name from the vector abbrevState. Lastly, use the state name to retrieve the correct state name/population pair from the vector statePopulation and output the pair.

Ex: If the input is:

```
21044
```

the output is:

```
Maryland: 6079602
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 18.5.1: LAB: Zip code and population (class templates) 0 / 10

Current file: main.cpp Load default template...

```
#include<iostream>
#include <fstream>
#include <vector>
#include <string>
#include "StatePair.h"
using namespace std;

int main() {
   ifstream inFS; // File input stream
   ifstream inFS; // File input stream
   int zip;
   int population;
   string abbrev;
   string state;
   unsigned int i;

```

Develop mode Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **main.cpp** (Your program) → Output

Program output displayed here

Coding trail of your work [What is this?](#)

```
History of your effort will appear here once you begin working
on this zyLab.
```

18.6 LAB: Pairs (class templates)

Complete `template<typename TheType> class Pair` by defining the following methods:

1. void Input()
   - Read two values from input and initialize the data members with the values in the order in which they appear
2. void Output()
   - Output the Pair in the format "[firstVal, secondVal]"
3. char CompareWith(Pair* otherPair)
   - Return the character '<', '=', or '>' according to whether the Pair is less than, equal to, or greater than otherPair

- Precedence of comparisons is firstVal then secondVal

4. char ShowComparison(Pair* otherPair)
    - Compare with otherPair by calling CompareWith()
    - Output the two Pairs separated by the character returned by CompareWith(). Hint: Output each Pair using Output()

Note: For each type main() calls Input() twice to create two Pairs of that type.

Ex: If the input for two Integer Pairs is:

```
4 6 3 5
```

the first Pair is [4, 6], and the second Pair is [3, 5].

Ex: If the input of the program is:

```
4 6 3 5
4.3 2.1 4.3 2.1
one two three four
```

the output is:

```
[4, 6] > [3, 5]
[4.3, 2.1] = [4.3, 2.1]
[one, two] < [three, four]
```

539740.3879454.qx3zqy7

LAB ACTIVITY 18.6.1: LAB: Pairs (class templates) 0 / 10

main.cpp

```
Loading latest submission...
```

Develop mode Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **main.cpp** (Your program) → Output

Program output displayed here

Coding trail of your work What is this?

Retrieving signature

18.7 LAB: Min, max, median (function templates)

Given the definitions of main(), Read() and Write(), complete main.cpp by implementing the following member functions:

`template<typename TheType> vector<TheType> GetStatistics(vector<TheType>& list)`

- Take a vector template as a parameter
- Store the minimum, median, and maximum values of the vector parameter into a new vector
- Return the new vector

`template<typename TheType> void Run(vector<TheType>& list)`

- Take a vector template as a parameter
- Call Read() with the vector parameter as an argument. Read() stores 5 input values into the vector parameter
- Sort the vector parameter
- Call Write() to output the sorted vector
- Output a new line

- Call GetStatistics() with the sorted vector as an argument. GetStatistics() returns a vector containing the minimum, median, and the maximum values of the sorted vector
- Call Write() to output the vector returned by GetStatistics()
- Output a new line

Note: vector.size() returns a long unsigned int, not an int.

Hint: Use the built-in sort function to sort a vector.

Ex: If the input is:

```
1 3 5 9 7
2.2 3.3 1.1 4.4 5.5
one two three four five
```

the output is:

```
1 3 5 7 9
1 5 9

1.1 2.2 3.3 4.4 5.5
1.1 3.3 5.5

five four one three two
five one two
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 18.7.1: LAB: Min, max, median (function templates) 0 / 10

main.cpp

Load default template...

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm> // to use sort()
using namespace std;

const int NUM_VALUES = 5;

// Input NUM_VALUES of TheType into the vector parameter
template<typename TheType> void Read(vector<TheType>& list) {
   for (int j = 0; j < NUM_VALUES; ++j) {
      cin >> list.at(j);
   }
}

```

Develop mode Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **main.cpp** (Your program) → Output

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

18.8 LAB: Ordered lists

An OrderedList is a vector that keeps elements in sorted order.

Complete `template <typename TheType> class OrderedList` by defining the following functions:

1. `int Size()`
   - Return the size of the list
2. `TheType At(int index)`
   - Return the element of the list at parameter index.
3. `int Find(TheType value)`
   - Return the index of the first element in the list equal to parameter value.
   - Return -1 if parameter value is not found in the list.
4. `bool Remove(TheType value)`
   - Search the list for parameter value. Hint: Use Find().
   - If parameter value is found in the list, remove the element found by moving the subsequent elements towards the beginning of the list. Decrement list size and return true.
   - Return false if parameter value is not found in the list.

Hint: Use any vector functions to simplify the implementations.

The template code provides the implementations of the following functions:

1. `void Insert(TheType value)`
   - Search the list for an element that is greater than parameter value.
   - If an element is found, increment list size and move the element and all subsequent elements towards the end of the list to make room for parameter value. Copy parameter value at the location that was occupied by the first element greater than parameter value.
   - If no such element is found, increment list size and add parameter value at the end of the list.
2. `void Print()`
   - Output the list, separated by a space character.

A main program is provided as a sample test in the develop mode. Unit tests will be used during a submission.

Ex: if the given main() is executed, the output of the program is

```
Size is correct
List is in correct order: 3 7 11
Index of 11 is correct -- 2
7 was removed correctly
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 18.8.1: LAB: Ordered lists | 0 / 10

main.cpp Load default template...

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

/* ----------- Template class OrderedList declaration ----------- */
template<typename TheType> class OrderedList {
   public:
      int Size();                                   // Number of elements in th
      TheType At(int index);                            // Return the element a

      int Find(TheType value);                      // Return index of first oc
                                                    // of value or -1 if not fo
      void Insert(TheType value);                   // Insert value at its sort
      bool Remove(TheType value);                   // Find the first occurrenc
```

Develop mode Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → **main.cpp** (Your program) → Output

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```