Function Templates

1. Introduction

A function template is a generic function that can work with any data type. Instead of hard coding or fixing the actual types for the parameters, return value, or local variables, you use a type parameter to specify a generic data type.

2. Default Arguments for Functions *(6.12)*

Default arguments are passed to parameters automatically if no argument is provided in the function call.

In the following example the default argument for length is 20, and the default argument for width is 10. These default values are used if the actual arguments are omitted in the function call. These default values are **not** used if values are provided in the function call.

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

void showArea(float length = 20, float width = 10) {
   float area = length * width;
   cout << "The area is " << area << endl;
}

int main() {
   showArea();     // using default values for length and width
   showArea(7);    // passing in 7 for length and using default value for width
   showArea(5, 3); // passing in values for both length and width
   return 0;
}
```

Sample output.

```
The area is 200
The area is 70
The area is 15
```

3. Overloading functions *(6.14)*

Sometimes, you will create two or more functions with the same name and performs the same operation, but use a different set of parameters or parameters of different data types. This is referred to as *overloading the function*.

For example, if we have a function called *square* for calculating the square of a number, we might overload the square function, one for calculating the square of an integer and one for calculating the square of a float as shown next.

```
int square(int number) {
   return number * number;
```

```
}

float square(float number) {
   return number * number;
}
```

This is needed in order for the square function to work for both an int and a float. In situations like this, it is more convenient to write a function template than an overloaded function.

4. Function template *(16.2)*

Instead of the two overloaded functions above, we can use the function template for the square function as shown next.

```
template <class T>
T square(T number) {
   return number * number;
}
```

The keyword **template** is used to denote that this is a function template. The generic data type is passed in inside the angle brackets. Recalled that previously the keyword **class** is used to create a user-defined data type; here it is used to say that a data type is passed in through the type variable T. You can pass in as many data types as needed by separating them with commas inside the angle brackets. After this the function definition is written as usual, except that the type parameters are substituted for the actual data type names.

When the function is called, if an int is passed in then T will be of type int, and all occurrences of T would be replaced by the compiler with int. Similarly, if a float is passed in then T will be of type float.

5. Complete file listing

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


////////////////////////////////////////////////////////////
// Function template
template <class T>
T square(T number) {
   return number * number;
}

int main() {
   int i = 4;
   float f = 3.5;
   cout << "Square of an int: " << square(i) << endl;
   cout << "Square of a float: " << square(f) << endl;
}
```

6. *Exercises* (Problems with an asterisk are more difficult)

1. Create a function template called *swap* that swaps two variables of any type that are passed in through the parameter list. Remember that swapping two variables simply means exchanging the values of the two variables. In order for the swapped result to be reflected outside you need to pass the two parameters in by reference, i.e., need to use the &.

2. Write a test program for the swap function you just created.

3. *Does your swap function also work for swapping two Temperature variables? If not, then do whatever is necessary in order to make it work.

4. *Does your swap function also work for swapping two Height variables? If not, then do whatever is necessary in order to make it work.

