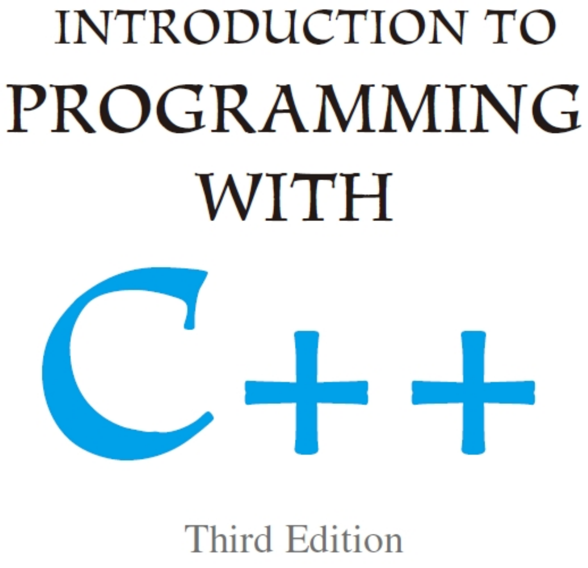
Introduction to Programming with C++

Chieh-Sen (Jason) Huang

Department of Applied Mathematics

National Sun Yat-sen University



INTRODUCTION TO
PROGRAMMING
WITH
C++

Third Edition

Contents are based on book by Y. Daniel Liang

Function Templates

- C++ provides functions and classes for developing reusable software.
- Templates provide the capability to parameterize types in functions and classes.
- With this capability, you can define one function or one class with a generic type that the compiler can substitute for a concrete type.

```
int maxValue(int value1, int value2)
{
 if (value1 > value2)
  return value1;
 else
  return value2;
 }
```

```
double maxValue(double value1, double value2)
{
 if (value1 > value2)
  return value1;
 else
  return value2;
 }
```

Function Templates

- It would save typing, save space, and make the program easy to maintain if you could simply define one function with a generic type as follows:

```
GenericType maxValue(GenericType value1, GenericType value2)
{
  if (value1 > value2)
   return value1;
  else
   return value2;
}
```

Function Templates

- The definition for the function template begins with the keyword **template** followed by a list of parameters.
- Each parameter must be preceded by the interchangeable keyword typename in the form **<typename typeParameter>**.

```
template<typename T>  \\ List12_1.cpp
T maxValue(T value1, T value2)
{
 if (value1 > value2)
  return value1;
 else
  return value2;
}
int main
{
    cout << "Maximum between 1 and 3 is "
         << maxValue(1, 3) << endl;
    cout << "Maximum between 1.5 and 0.3 is "
         << maxValue(1.5, 0.3) << endl;
-----------------------------------------------
Maximum between 1 and 3 is 3
Maximum between 1.5 and 0.3 is 1.5
```

Function Templates

- You can modify it using pass-by-reference.

```
template<typename T>  \\ List12_1.cpp
T maxValue(T& value1, T& value2)
{
 if (value1 > value2)
  return value1;
 else
  return value2;
}
int main
{
    cout << "Maximum between 1 and 3 is "
         << maxValue(1, 3) << endl;
    cout << "Maximum between 1.5 and 0.3 is "
         << maxValue(1.5, 0.3) << endl;
-----------------------------------------------
Maximum between 1 and 3 is 3
Maximum between 1.5 and 0.3 is 1.5
```

In-Class Exercise: Write template for sum function.
Homework 12.1: Write template for swap function.

Class Templates

- Note also that the class name before the scope resolution operator `::` is Array<T>, not Array.

```
template <typename T>
class Array{ \\ Lits12_2a.cpp
   private:
      T data[5];
   public:
      void setData(int num, T d);};
template <typename T>
void Array<T>::setData(int num, T d)
{
   if(num < 0 || num > 4 )
      cout << "oversize \n";
   else
      data[num] = d;
}
int main(){
   cout << "Create Array of type int\n";
   Array<int> i_array;
   i_array.setData(0, 80);}
```

In-Class Exercise: Use Class Template to rewrite Vector class so that it takes integers member variables x and y.

Class Templates with a default type parameter

- C++ allows you to assign a default type for a type parameter in a class template.
- For example, you may assign int as a default type in the generic Array class as follows: `template<typename T = int>`
- We use `Array<> i_array;` when declare the class.

```
template <typename T = int>
class Array{ \\ Lits12_2a.cpp
   private:
      T data[5];
   public:
      void setData(int num, T d);
};
template <typename T>
void Array<T>::setData(int num, T d)
{
   if(num < 0 || num > 4 )
      cout << "oversize \n";
   else
      data[num] = d;
}
int main()
{
   cout << "Create Array of type int\n";
   Array<> i_array;
   i_array.setData(0, 80);
}
```

Mixed nontype and type parameters

- You also can use nontype parameters along with type parameters in a template prefix.

```
template <typename T, int length>
class Array{ \\ Lits12_3a.cpp
 private:
   T data[length];
 public:
   void setData(int num, T d);
};
template <typename T, int length>
void Array<T, length>::setData(int num, T d)
{
   if(num < 0 || num > length )
      cout << "oversize \n";
   else
      data[num] = d;
}
int main()
{
   cout << "Create Array of type int\n";
   Array<int, 5> i_array;
   i_array.setData(0, 80);
}
```

Templates

- Find the biggest in the “Array”.

```
template <typename T, int length>
T Array<T, length>::getMax()
{
T max = data[0];
for(int i = 1; i < length; i++)
  max = (data[i] > max) ? data[i] : max;
    return max;
}
```

In-Class Exercise: Find the average of the “Array”.
Homework 12.2: Write the bubble sort function to use a generic type for array elements.