COMPREHENSIVE EDITION

PROGRAMMING AND PROBLEM SOLVING WITH C++

SIXTH EDITION

Nell Dale and Chip Weems

Chapter 16

Templates and Exceptions

Background image © Toncsi/ShutterStock, Inc.

www.jblearning.com

Chapter 16 Topics

- C++ Function Templates
- Instantiating a Function Templates
- C++ Class Templates
- Instantiating Class Templates
- Function Definitions for Members of a Template Class

Generic Algorithms

- Generic algorithms are algorithms in which the actions or steps are defined, but the data types of the items being manipulated are not

Example of a Generic Algorithm

```
void PrintInt(int n)
{
    cout << "***Debug" << endl;
    cout << "Value is " << n << endl;
}
void PrintChar(char ch)
{
    cout << "***Debug" << endl;
    cout << "Value is " << ch << endl;
}
void PrintFloat(float x)
{
    cout << "***Debug" << endl;
    cout << "Value is " << x << endl;

}
void PrintDouble(double d)
{
    cout << "***Debug" << endl;
    cout << "Value is " << d << endl;


}
```

To output the traced values,we insert:

```
sum = alpha + beta + gamma;
PrintInt(sum);

PrintChar(initial);

PrintFloat(angle);
```

Function Overloading

- **Function overloading** is the use of the same name for different functions, distinguished by their parameter lists

  - Eliminates need to come up with many different names for identical tasks

  - Reduces the chance of unexpected results caused by using the wrong function name

Example of Function Overloading

```
void Print(int n)
{
    cout << "***Debug" << endl;
    cout << "Value is " << n << endl;
}
void Print(char ch)
{
    cout << "***Debug" << endl;
    cout << "Value is " << ch << endl;
}
void Print(float x)
{

}
```

To output the traced values, we insert:

```
Print(someInt);
Print(someChar);
Print(someFloat);
```

Function Template

- A C++ language construct that allows the compiler to generate multiple versions of a function by allowing parameterized data types

FunctionTemplate

Template < TemplateParamList >
FunctionDefinition

TemplateParamDeclaration

{ class
Identifier
typename

Example of a Function Template

```
template<class SomeType>

void Print(SomeType val)
{
    cout << "***Debug" << endl;
     cout << "Value is " << val <<
  endl;
}
```

Template parameter

Template argument

To output the traced values, we insert:

```
Print<int>(sum);
Print<char>(initial);
Print<float>(angle);
```

Instantiating a Function Template

- When the compiler instantiates a template, it substitutes the template argument for the template parameter throughout the function template

TemplateFunction Call

Function < TemplateArgList > (FunctionArgList)

Generic Functions, Function Overloading, Template Functions

Generic Function
Different Function Definitions
Different Function Names

Function Overloading
Different Function Definitions
Same Function Name

Template Functions
One Function Definition (a function template)
Compiler Generates Individual Functions

What is a Generic Data Type?

- It is a type for which the operations are defined but the data types of the items being manipulated are not

What is a Class Template?

- **It is a C++ language construct that allows the compiler to generate multiple versions of a class by allowing parameterized data types**

Example of a Class Template

```
template<class ItemType>
class GList
{
public:
    bool IsEmpty() const;
    bool IsFull() const;
    int  Length() const;
    void Insert(/* in */ ItemType item);
    void Delete(/* in */ ItemType item);
    bool IsPresent(/* in */ ItemType item) const;
    void SelSort();
    void Reset() const;
    ItemType GetNextItem();
    GList();                        // Constructor
```

Template parameter

Example of a Class Template, cont. . .

```
private:
    int      length;
    ItemType data[MAX_LENGTH];
};
```

Instantiating a Class Template

To create lists of different data types

template argument

```
// Client code

GList<int> list1;
GList<float> list2;
GList<string> list3;

list1.Insert(356);
list2.Insert(84.375);
list3.Insert("Muffler bolt");
```

Compiler generates 3 distinct class types

```
GList_int list1;
GList_float list2;
GList_string list3;
```

Instantiating a Class Template

- Class template arguments *must* be explicit
- The compiler generates distinct class types called template classes or generated classes
- When instantiating a template, a compiler substitutes the template argument for the template parameter throughout the class template

Substitution Example

```
class GList_int
{
public:
                                    int
void Insert(/* in */ ItemType item);
                                         int
    void Delete(/* in */ ItemType item);

    bool IsPresent(/* in */ ItemType item) const;
                                   int
private:
    int      length;
    ItemType data[MAX_LENGTH];
                  int
};
```

Writing Function Templates

```
template<class ItemType>
void GList<ItemType>::Insert(/* in */ ItemType item)
{
    data[length] = item;
    length++;
}
```

Writing Function Templates

```
void GList<float>::Insert(/* in */ float item)
{
    data[length] = item;
    length++;
}
```

Organization of Program Code

- A compiler must know the argument to the template in order to generate a function template, and this argument is located in the client code
- Solutions
  - Have specification file include implementation file
  - Combine specification file and implementation file into one file

Warning!

Are you using an IDE (integrated development environment) where the editor, compiler, and linker are bundled into one application?

Remember The compiler must know the template argument

How you organize the code in a project may differ depending on the IDE you are using