Chapter 15

User Defined Templates

Templates

Templates are a feature of the C++ programming language that allows functions and classes to operate with *generic types*.

- Templates are basically a way for us to provide a blueprint for a function or class that the compiler will use to generate code for us!

- Template code is generated at *compile time.*

There are 2 different types of templates:

1. Function Templates

2. Class Templates

Function Templates

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

Useful when we have multiple functions that are nearly identical, differing only in their data types.

```
template <class identifier>
function_declaration;

template <typename identifier>
function_declaration;
```

example: functionTemplates.cpp

Function Templates

The **type parameter** is a generic type that is used throughout a function definition to represent the data type of determined at compile time.

The compiler automatically generates a unique function definition for each type that is called throughout the program. The programmer never sees these functions.

```
findMax(100, 200);
findMax('a', 'z');
findMax(100.0, 200.0);
```

Template Specialization

Often times templated functions need to differ depending on the type. We can combine templated functions with **fully specialized** functions that will be used for a specific type.

```
template<typename T>
T add(T val1, T val2);

template<>
char add(char val1, char val2);
```

Partial Specialization is used to represent a specific templated function pertaining to a set of generic data types

```
template<typename T>
T add(T val1, T val2);

template<typename T>
T* add(T* val1, T* val2);
```

example: templateSpecialization.cpp

Templates with multiple parameters

We can also have a function template that has multiple parameters

Construct 15.1.1: Function template with multiple parameters.

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

Feedback?

example: multipleParameters.cpp

Using Classes with Templated Functions

When using templated functions, you need to ensure that if you are using an object that the object supports all operations that may be performed on it during the templated function call.

```
Dog dog1 = Dog(10);
Dog dog2 = Dog(20);

Dog maxDog = findMax(dog1, dog2);
maxDog.print();
```

example: objects.cpp

Class templates

When multiple classes may be identical, differing only in data types, we can use **class templates.**

```
template<typename T>
class ClassDef {
  T data;
  public:
    void print();
};

template<typename T>
void ClassDef<T>::print() {
  ...
}
```

example: objects.cpp

Class templates

- Class templates must be defined in the same file, as the compiler will have to have all code available to it at compile time when it goes to generate the code for a particular type!

- It is best practice to create create templates in a *header file (i.e. .h file).*

- member functions of a class defined inline do not need to be marked with the *template* keyword

- member functions defined outside of the class definition DO.

```
template<typename T>
T Class<T>::getVal()
```

For more on templates…

Visit: http://www.cplusplus.com/doc/oldtutorial/templates/