ANSI C++ Library and Namespaces

Chapter 11

Objectives

- **Outline the functionality of the ANSI C++ library.**
- **Explain the header files.**
- **Explain the use of namespaces in ANSI C++.**
- **Describe the string class in ANSI C++**
- **Describe the use of templates in ANSI C++.**
- **Implement programs using features of the ANSI C++ library.**
- **Gain experience through code walk-throughs and lab exercises.**
  - The example programs are in the chapter directory.
  - Labs located in Labs/Lab11

ANSI C++ Library

- **ANSI C++ comes with a greatly enhanced library with new header files and organization.**
  - Header files do not take a .h extension.
  - Symbols in the library are in the namespace “std”.
  - Templates are used extensively.

Features in the Library

- **There are a number of features in the library**
  - The standard C library.
  - Iostreams has templates, throws exceptions, and supports strings.
  - There is a standard string class.
  - The “Standard Template Library” has been incorporated, including many container classes and algorithms.
  - There is support for numeric processing, including support for complex numbers, higher precision, and compiler optimizations.
  - Diagnostic support includes a number of exception classes.

Global Namespace

- **C++ provides a single global namespace in which all names declared in global scope are entered.**
  - Single namespace is difficult for library providers and users.
  - Global names in a library may collide with the global names in a user application or another library (e.g. there may be two Vector classes).

Namespaces

- **ANSI C++ provides a “namespace” mechanism to avoid such conflicts.**
- **The ANSI C++ standard library is in the namespace “std”.**
  - Avoid using default access to symbols in a namespace when designing libraries.
- **Define your own namespace using the namespace keyword.**

```
/*
For default access to the symbols in a
namespace employ:
 a “using namespace std” statement
 for a single type “using std::typename”
*/

#include <iostream>

using std::cout, std::string;

namespace ns1{string message(“Hello\n"); }
namespace ns2 { string message("Goodbye\n"); }

int main(){
    using namespace ns1;
    cout << message << std::endl;

    using std::endl;
    cout << ns2::message << endl;
}
```

Anonymous Namespaces

- **Use an anonymous namespace to localize global symbols to a translation unit (.cpp file and all it's includes)**

```
namespace {
    int x = 1, y = 2;
}
```

- **If another symbol with the same name is defined elsewhere there will not be a violation of the One Definition Rule.**
- **All anonymous namespaces in the same file are treated as the same namespace and all anonymous namespaces in different files are distinct.**

Nested Namespaces

- **Namespaces may be nested.**
- **An ordinary nested namespace has unqualified access to its parent's members**
- **The parent members do not have unqualified access to the nested namespace (unless it is declared as inline).**
- **Examine the example nested app in folder Namespaces**

```
namespace DataServer
{
    void Foo();
    namespace Details
    {
      int CountImpl;
      void Ban() { return Foo(); }
    }
    int Bar(){...};
    int Baz(int i) { return Details::CountImpl;
}
```

Organize UDT Models

- **Namespaces are often used to organize object libraries.**
  - **The C++ Standard Library (std) and Boost Library (boost) are examples**
- **Namespace span .cpp and .h files for each UDT in the model.**
- **Examine the example ns-model app in folder Namespaces**

```
// in header file Shape.h:
namespace learncpp {
  class Shape {
  public:
       Shape(){}
       void Render();
       ~Shape(){}

  };

}
// in implementation file Shape.cpp:
#include "Shape.h"
void learncpp::Shape::Render(){
  std::cout << "Shape" << std::endl;
}
```

ANSI C++ string

- **std::string implements a first-class character string data type.**
- **Avoids many problems associated with simple character arrays ("C-style strings").**
- **You can define a string object very simply by including the <string> library, as shown in the TryString program.**

Templates

- **A t*emplate* can be used to generate a function or class based on type parameters.**
- **You instantiate a template by passing an actual type parameter in angle brackets.**
- **The Standard Library type *string* is actually a typedef for a template instantiation of the fundamental type *basic_string*:**

  typedef basic_string<char> string;

  More on templates in a later module…
- **The IntString sample program illustrates a basic_string of int.**
- **Much more on this topic in the next module.**

Summary

- **The ANSI C++ Standard Library has extensive functionality, including support for numeric processing and strings, and it incorporates the Standard Template Library.**
- **The Standard Template Library (STL) has extensive support for containers and algorithms.**
- **The new header files do not take a .h extension, and all symbols are in the namespace *std*.**
- **The *string* class in ANSI C++ is a template instantiation of *basic_string*, which can be used to implement sequences of other types besides *char*.**