Lecture Notes

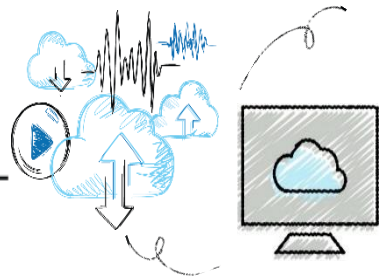
Chapter 7

User-Defined Simple Data Types, Namespaces, and the `string` Type

ECE 111: Introduction to C and C++ Programming

Instructor: Dr. Shayan (Sean) Taheri

Gannon University (GU)



Personal Information

- Name: Shayan (Sean) Taheri.
- Date of Birth: July/28/1991.
- Current Position: Assistant Professor at Gannon University
- Previous Position: Postdoctoral Fellow at University of Florida.
- Ph.D. Degree: Electrical Engineering from the University of Central Florida.
- M.S. Degree: Computer Engineering from the Utah State University.
- University Profile: https://www.gannon.edu/FacultyProfiles.aspx?profile=taheri001

Objectives (1 of 2)

- In this chapter, you will:
  - Learn how to create and manipulate your own simple data type—called the enumeration type
  - Explore how the assignment statement, and arithmetic and relational operators work with **`enum`** types
  - Learn how to use **`for`** loops with **`enum`** types
  - Learn how to input data into an **`enum`** type
  - Learn how to output data stored in an **`enum`** type

Objectives (2 of 2)

- Explore how to write functions to process **`enum`** types
- Learn how to declare variables when defining the enumeration type
- Become familiar with anonymous types
- Become familiar with the **`typedef`** statement
- Learn about the **`namespace`** mechanism
- Explore the **`string`** data type, and learn how to use **`string`** functions to manipulate strings

CENGAGE Learning

Enumeration Type (1 of 5)

- A data type is a set of values with a set of operations on them
- Enumeration type is a simple data type created by the programmer
- To define an enumeration type, you need:
  - A name for the data type
  - A set of values for the data type
  - A set of operations on the values

CENGAGE Learning

Enumeration Type (2 of 5)

- You can specify the name and the values, but not the operations
- The syntax for enumeration type is:

```
enum typeName {value1, value2, ...};
```

  - **value1**, **value2**, … are identifiers called enumerators
  - List specifies the ordering:

```
value1 < value2 < value3 <...
```

Enumeration Type (3 of 5)

- The enumeration type is an ordered set of values
  - Default value assigned to enumerators starts at 0
- A value used in one enumeration type cannot be used by another in the same block
- Same rules apply to enumeration types declared outside of any blocks

Enumeration Type (4 of 5)

EXAMPLE 7-1

The statement:

```
enum colors {BROWN, BLUE, RED, GREEN, YELLOW};
```

defines a new data type called `colors`, and the values belonging to this data type are `BROWN`, `BLUE`, `RED`, `GREEN`, and `YELLOW`.

EXAMPLE 7-2

The statement:

```
enum standing {FRESHMAN, SOPHOMORE, JUNIOR, SENIOR};
```

defines `standing` to be an enumeration type. The values belonging to `standing` are `FRESHMAN`, `SOPHOMORE`, `JUNIOR`, and `SENIOR`.

Enumeration Type (5 of 5)

EXAMPLE 7-3

Consider the following statements:

```
enum grades {'A', 'B', 'C', 'D', 'F'}; //illegal enumeration type
enum places {1ST, 2ND, 3RD, 4TH};  //illegal enumeration type
```

These are illegal enumeration types because none of the values is an identifier. The following, however, are legal enumeration types:

```
enum grades {A, B, C, D, F};
enum places {FIRST, SECOND, THIRD, FOURTH};
```

EXAMPLE 7-4

Consider the following statements:

```
enum mathStudent {JOHN, BILL, CINDY, LISA, RON};
enum compStudent {SUSAN, CATHY, JOHN, WILLIAM}; //illegal
```

Suppose that these statements are in the same program in the same block. The second enumeration type, `compStudent`, is not allowed because the value `JOHN` was used in the previous enumeration type `mathStudent`.

Declaring Variables

- Syntax

```
dataType identifier, identifier,...;
```

- Example

```
enum sports {BASKETBALL, FOOTBALL, HOCKEY, BASEBALL, SOCCER,
             VOLLEYBALL};
```

  - Can declare variables such as:

```
sports popularSport, mySport;
```

CENGAGE Learning

Assignment

- Values can be stored in enumeration data types:

```
popularSport = FOOTBALL;
```

  - Stores **`FOOTBALL`** into **`popularSport`**

Operations on Enumeration Types

- No arithmetic operations are allowed on enumeration types

```
mySport = popularSport + 2; //illegal
popularSport = FOOTBALL + SOCCER; //illegal
popularSport = popularSport * 2; //illegal
```

- ++ and -- are illegal, too

```
popularSport++; //illegal
popularSport--; //illegal
```

- The solution is applying the cast operator

```
popularSport = FOOTBALL;
popularSport = static_cast<sports>(popularSport + 1);
```

CENGAGE Learning

Relational Operators

- An enumeration type is an ordered set of values:

```
FOOTBALL <= SOCCER is true
HOCKEY > BASKETBALL is true
BASEBALL < FOOTBALL is false
```

- An enumeration type is an integral data type and can be used in loops:

```
for (mySport = BASKETBALL; mySport <= SOCCER; mySport =
                                    static_cast<sports>(mySport + 1))
```

.
.
.

This `for` loop has five iterations

CENGAGE Learning

Input /Output of Enumeration Types

- An enumeration type cannot be input/output (directly)
  - Can input and output indirectly – refer to code segments below:

```
enum courses {ALGEBRA, BASIC, PYTHON, CPP, PHILOSOPHY,
              ANALYSIS, CHEMISTRY, HISTORY};
courses registered;

switch (ch1)
{
case 'a':
    if (ch2 == 'l')
        registered = ALGEBRA;
    else
        registered = ANALYSIS;
    break;
```

Functions and Enumeration Types

- Enumeration types can be passed as parameters to functions either by value or by reference
- A function can return a value of the enumeration type

Declaring Variables When Defining the Enumeration Type

- Can declare variables of an enumeration type when you define an enumeration type:

```
enum grades {A, B, C, D, F} courseGrade;
```

Anonymous Data Types (1 of 2)

- Anonymous type values are directly specified in the declaration, with no type name
- Example:

```
enum {BASKETBALL, FOOTBALL, BASEBALL, HOCKEY} mySport;
```

Anonymous Data Types (2 of 2)

- Drawbacks:
  - Cannot pass/return an anonymous type to/from a function
  - Values used in one type can be used in another, but are treated differently:

```
enum {ENGLISH, FRENCH, SPANISH, GERMAN, RUSSIAN} languages;
enum {ENGLISH, FRENCH, SPANISH, GERMAN, RUSSIAN} foreignLanguages;
```

  - This statement is illegal:

```
languages = foreignLanguages; //Illegal
```

- Best practices: to avoid confusion, define an enumeration type first, then declare variables

`typedef` Statement (1 of 2)

- The **`typedef`** statement is used to create synonyms or aliases to a data type
- The syntax of the **`typedef`** statement is:

```
typedef existingTypeName newTypeName;
```

- **`typedef`** does not create any new data types
  - Only creates an alias to an existing data type

CENGAGE Learning

`typedef` Statement (2 of 2)

- The **typedef** statement is used to create synonyms or aliases to a data type
- The syntax of the **`typedef`** statement is:

```
typedef existingTypeName newTypeName;
```

- **`typedef`** does not create any new data types
  - Only creates an alias to an existing data type

Namespaces (1 of 6)

- ANSI/ISO standard C++ was officially approved in July 1998
- Most recent compilers are compatible with ANSI/ISO standard C++
- For the most part, standard C++ and ANSI/ISO standard C++ are the same
  - However, ANSI/ISO Standard C++ has some features not available in Standard C++

Namespaces (2 of 6)

- Global identifiers in a header file used in a program become global in the program
  - A syntax error occurs if a program's identifier has the same name as a global identifier in the header file
- The same problem can occur with third-party libraries
  - Common solution: third-party vendors begin their global identifiers with _ (underscore)
  - Do not begin identifiers in your program with _

CENGAGE Learning

Namespaces (3 of 6)

- ANSI/ISO Standard C++ attempts to solve this problem with the namespace mechanism
- The general syntax of the statement **namespace** is:

```
namespace namespace_name
{
    members
}
```

where **members** consist of variable declarations, named constants, functions, or another **namespace**

Namespaces (4 of 6)

EXAMPLE 7-8

The statement:

```
namespace globalType
{
    const int N = 10;
    const double RATE = 7.50;
    int count = 0;
    void printResult();
}
```

defines `globalType` to be a `namespace` with four members: named constants `N` and `RATE`, the variable `count`, and the function `printResult`.

Namespaces (5 of 6)

- A **namespace** member has scope local to the namespace
- A **namespace** member can be accessed outside the **namespace**
  - The general syntax for accessing a **namespace** member is:

```
namespace_name::identifier
```

  - ANSI/ISO Standard C++ provides the use of the statement **using**

```
using namespace namespace_name;
```

```
using namespace_name::identifier;
```

CENGAGE Learning

Namespaces (6 of 6)

- Examples with namespaces

```
globalType::RATE
using namespace globalType;
using globalType::RATE;
```

- After the **using** statement, it is not necessary to put the **namespace_name::** before the **namespace** member
  - Unless a **namespace** member and a global identifier or a block identifier have the same name

`string` Type

- To use data type **`string`**, a program must include the header file **`string`**
- A string is a sequence of zero or more characters
  - The first character is in position **0**
  - The second character is in position **1**, etc.
- Binary operator **+** performs the string concatenation operation
- Array subscript operator **`[]`** allows access to an individual character in a string

Additional `string` Operations

- The data type **`string`** has a data type, **`string::size_type`**, and a named constant, **`string::npos`**, defined as follows:

`string::size_type`	An unsigned integer (data) type
`string::npos`	The maximum value of the (data) type **`string::size_type`**, a number such as **`4294967295`** on many machines

Example 7-18: `swap` Function

EXAMPLE 7-18

The **`swap`** function is used to swap—that is, interchange—the contents of two string variables.

Suppose you have the following statements:

```
string str1 = "Warm";
string str2 = "Cold";
```

After the following statement executes, the value of **`str1`** is **`"Cold"`** and the value of **`str2`** is **`"Warm"`**.

```
str1.swap(str2);
```

Quick Review (1 of 3)

- Enumeration type: set of ordered values
  - Reserved word `enum` creates an enumeration type
- No arithmetic operations are allowed on the enumeration type
- Relational operators can be used with `enum` values
- Enumeration type values cannot be input or output directly
- Enumeration types can be passed as parameters to functions by value or by reference

Quick Review (2 of 3)

- Anonymous type: a variable's values are specified without any type name
- Reserved word **`typedef`** creates synonyms or aliases to previously defined data types
- The **`namespace`** mechanism is a feature of ANSI/ISO Standard C++
- A **`namespace`** member is usually a named constant, variable, function, or another namespace
- Scope of a namespace member is local to namespace

Quick Review (3 of 3)

- **`using`** statement simplifies access to **`namespace`** members
- A **`string`** is a sequence of zero or more characters
- Strings in C++ are enclosed in **`""`**
- The first character of a string is in position **0**
- In C++, **`[]`** is the array subscript operator

Reading Assignment – Very Important for "GU – ECE 111"

- Malik, D.S., 2014. **C++ programming: Program design including data structures**. Cengage Learning.
  - "**Chapter 7**: **User-Defined Simple Data Types, Namespaces, and the string Type**".

CENGAGE Learning