C++ Programming: From Problem Analysis to Program Design, Fourth Edition

Chapter 8: User-Defined Simple Data Types, Namespaces, and the `string` Type

Objectives

In this chapter, you will:

- Learn how to create and manipulate your own simple data type—called the enumeration type
- Become familiar with the `typedef` statement
- Learn about the `namespace` mechanism
- Explore the `string` data type, and learn how to use the various `string` functions to manipulate strings

Enumeration Type

- Data type: a set of values together with a set of operations on those values
- To define a new simple data type, called enumeration type, we need three things:
  - A name for the data type
  - A set of values for the data type
  - A set of operations on the values

Enumeration Type (continued)

- A new simple data type can be defined by specifying its name and the values, but not the operations
  - The values must be identifiers
- Syntax:

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

  - `value1`, `value2`, … are identifiers called enumerators
  - `value1 < value2 < value3 <...`

Enumeration Type (continued)

- Enumeration type is an ordered set of values
- If a value has been used in one enumeration type it can’t be used by another in same block
- The same rules apply to enumeration types declared outside of any blocks

Enumeration Type (continued)

EXAMPLE 8-1

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 8-2

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`.

EXAMPLE 8-3

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 8-4

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,...;
```

- For example, given the following definition:

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

we can declare the following variables:

```
sports popularSport, mySport;
```

Assignment

- The statement:

```
popularSport = FOOTBALL;
```

stores `FOOTBALL` into `popularSport`

- The statement:

```
mySport = popularSport;
```

copies the value of the `popularSport` into `mySport`

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
```

- Solution: use a static cast:

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

Relational Operators

- An enumeration type is an ordered set of values:

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

- Enumeration type is an integer data type and can be used in loops:

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

Input /Output of Enumeration Types

- I/O are defined only for built-in data types
  - Enumeration type cannot be input/output (directly)

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

```
switch (registered)
{
case ALGEBRA:
    cout << "Algebra";
    break;
case ANALYSIS:
    cout << "Analysis";
    break;
case BASIC:
    cout << "Basic";
    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

- You can declare variables of an enumeration type when you define an enumeration type:

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

Anonymous Data Types

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

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

- 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;
languages = foreignLanguages; //illegal
```

`typedef` Statement

- You can create synonyms or aliases to a data type using the `typedef` statement
- Syntax:

```
typedef existingTypeName newTypeName;
```

- `typedef` does not create any new data types
  - Creates an alias to an existing data type

Namespaces

- ANSI/ISO standard C++ was officially approved in July 1998
- Most of the recent compilers are also 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 (continued)

- Global identifiers in a header file used in a program become global in the program
  - Syntax error occurs if an identifier in a program has the same name as a global identifier in the header file
- 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 _

Namespaces (continued)

- ANSI/ISO Standard C++ attempts to solve this problem with the namespace mechanism
- Syntax:

```
namespace namespace_name
{
    members
}
```

where a member is usually a variable declaration, a named constant, a function, or another namespace

Namespaces (continued)

EXAMPLE 8-8

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 (continued)

- The scope of a `namespace` member is local to the namespace
- Ways a `namespace` member can be accessed outside the `namespace`:

```
namespace_name::identifier
```

```
using namespace namespace_name;
```

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

Accessing a `namespace` Member

- Examples:

```
globalType::RATE
globalType::printResult();
```

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

`string` Type

- To use the data type `string`, the program must include the header file `string`
- The statement:

```
string name = "William Jacob";
```

  declares name to be a string variable and also initializes name to `"William Jacob"`
  - The first character, `'W'`, is in position 0
  - The second character, `'i'`, is in position 1
  - `name` is capable of storing any size string

`string` Type (continued)

- Binary operator `+` and the array subscript operator `[]`, have been defined for the data type `string`
  - `+` performs the string concatenation operation
- Example:

```
str1 = "Sunny";
str2 =  str1 + " Day";
```

stores "`Sunny Day`" into `str2`

Additional `string` Operations

- `length`
- `size`
- `find`
- `substr`
- `swap`

`length` Function

- Returns the number of characters currently in the string
- Syntax:

```
strVar.length()
```

  where `strVar` is variable of the type `string`
- `length` returns an unsigned integer
- The value returned can be stored in an integer variable

```
string firstName;
string name;
string str;

firstName = "Elizabeth";
name = firstName + " Jones";
str = "It is sunny.";
```

Statement	Effect
`cout << firstName.length() << endl;`	Outputs 9
`cout << name.length() << endl;`	Outputs 15
`cout << str.length() << endl;`	Outputs 12

```
string::size_type len;
```

Statement	Effect
`len = firstName.length();`	The value of `len` is 9
`len = name.length();`	The value of `len` is 15
`len = str.length();`	The value of `len` is 12

`size` Function

- `size` is the same as the function `length`
  - Both functions return the same value
- Syntax:

```
strVar.size()
```

  where `strVar` is variable of the type string
- As in the case of the function `length`, the function `size` has no arguments

`find` Function

- Searches a string for the first occurrence of a particular substring
- Returns an unsigned integer value of type `string::size_type`
  - Or `string::npos` if unsuccessful
- Syntax:

```
strVar.find(strExp)
```

```
strVar.find(strExp, pos)
```

  - `strExp` can be a string or a character

`find` Function (continued)

```
string sentence;
string str;
string::size_type position;

sentence = "Outside it is cloudy and warm.";
str = "cloudy";
```

Statement	Effect
`cout << sentence.find("is") << endl;`	Outputs 11
`cout << sentence.find("and") << endl;`	Outputs 21
`cout << sentence.find('s') << endl;`	Outputs 3
`cout << sentence.find('o') << endl;`	Outputs 16
`cout << sentence.find(str) << endl;`	Outputs 14
`cout << sentence.find("the") << endl;`	Outputs the value of `string::npos`
`cout << sentence.find('i', 6) << endl;`	Outputs 8
`position = sentence.find("warm");`	Assigns 25 to `position`

`substr` Function

- Returns a particular substring of a string
- Syntax:

```
strVar.substr(expr1, expr2)
```

`expr1` and `expr2` are expressions evaluating to unsigned integers

  - `expr1` specifies a position within the string (starting position of the substring)
  - `expr2` specifies the length of the substring to be returned

`substr` Function (continued)

```
string sentence;
string str;

sentence = "It is cloudy and warm.";
```

Statement	Effect
`cout << sentence.substr(0, 5) << endl;`	Outputs: `It is`
`cout << sentence.substr(6, 6) << endl;`	Outputs: `cloudy`
`cout << sentence.substr(6, 16) << endl;`	Outputs: `cloudy and warm.`
`cout << sentence.substr(17, 10) << endl;`	Outputs: `warm.`
`cout << sentence.substr(3, 6) << endl;`	Outputs: `is clo`
`str = sentence.substr(0, 8);`	`str = "It is cl"`
`str = sentence.substr(2, 10);`	`str = " is cloudy"`

`swap` Function

- Interchanges contents of two string variables
- Syntax:

```
strVar1.swap(strVar2);
```

  where `strVar1` and `strVar2` are string variables

- Suppose you have the following statements:

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

- After `str1.swap(str2);` executes, the value of `str1` is `"Cold"` and the value of `str2` is `"War"`

Programming Example: Pig Latin Strings

- Program prompts user to input a string
  - Then outputs the string in the pig Latin form
- The rules for converting a string into pig Latin form are as follows:
  - If the string begins with a vowel, add the string `"-way"` at the end of the string
    - Example: the pig Latin form of `"eye"` is `"eye-way"`

Programming Example: Pig Latin Strings (continued)

- Rules (continued):
  - If the string does not begin with a vowel, first add `"-"` at the end of the string
    - Then move the first character of the string to the end of the string until the first character of the string becomes a vowel
    - Next, add the string `"ay"` at the end
    - Example: pig Latin form of `"There"` is `"ere-Thay"`

Programming Example: Pig Latin Strings (continued)

- Rules (continued):
  - Strings such as "by" contain no vowels
    - The letter 'y' can be considered a vowel
    - For this program the vowels are a, e, i, o, u, y, A, E, I, O, U, and Y
  - Strings such as "1234" contain no vowels
    - The pig Latin form of a string that has no vowels in it is the string followed by the string "-way"
    - Example: pig Latin form of "1234" is "1234-way"

Programming Example: Problem Analysis

- If `str` denotes a string:
  - Check the first character, `str[0]`, of `str`
  - If it is a vowel, add `"-way"` at the end of `str`
  - If it is not a vowel:
    - First add `"-"` at the end of the string
    - Remove the first character of `str` from `str` and put it at end of `str`
    - Now the second character of `str` becomes the first character of `str`

Programming Example: Problem Analysis (continued)

- If `str` denotes a string (continued):
  - This process is repeated until either
    - The first character of `str` is a vowel
    - All characters of `str` are processed, in which case `str` does not contain any vowels

Programming Example: Algorithm Design

- The program contains the following functions:
  - `isVowel` determines if a character is a vowel
  - `rotate` moves first character of `str` to the end of `str`
  - `pigLatinString` finds pig Latin form of `str`
- Steps in the algorithm:
  - Get `str`
  - Use `pigLatinString` to find the pig Latin form of `str`
  - Output the pig Latin form of `str`

Programming Example: Function `isVowel`

```
bool isVowel(char ch)
{
    switch (ch)
    {
    case 'A':
    case 'E':
    case 'I':
    case 'O':
    case 'U':
    case 'Y':
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':
    case 'y':
        return true;
    default:
        return false;
    }
}
```

Programming Example: Function `rotate`

- Takes a string as a parameter
- Removes the first character of the string
  - Places it at end of the string by extracting the substring starting at position 1 until the end of the string, then adding the first character of the string

```
string rotate(string pStr)
{
    string::size_type len = pStr.length();

    string rStr;

    rStr = pStr.substr(1, len - 1) + pStr[0];

    return rStr;
}
```

Programming Example: Function `pigLatinString`

- If `pStr[0]` is a vowel, add `"-way"` at end
- If `pStr[0]` is not a vowel:
  - Move first character of `pStr` to the end of `pStr`
  - The second character of `pStr` becomes the first character of `pStr`
    - Now `pStr` may or may not contain a vowel
  - Use a `bool` variable, `foundVowel`, which is set to `true` if `pStr` contains a vowel and `false` otherwise
  - Initialize `foundVowel` to `false`

Programming Example: Function `pigLatinString` (continued)

  - If `pStr[0]` is not a vowel, move `str[0]` to the end of `pStr` by calling the function `rotate`
  - Repeat third step until either the first character of `pStr` becomes a vowel or all characters of `pStr` have been checked
- Convert `pStr` into the pig Latin form
- Return `pStr`

Programming Example: Main Algorithm

- Get the string
- Call `pigLatinString` to find the pig Latin form of the string
- Output the pig Latin form of the string

Summary

- Enumeration type: set of ordered values
  - Created with 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

Summary (continued)

- 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

Summary (continued)

- Keyword `namespace` must appear in the `using` statement
- A string is a sequence of zero or more characters
- Strings in C++ are enclosed in `""`
- In C++, `[]` is the array subscript operator
- The function `length` returns the number of characters currently in the string

Summary (continued)

- The function `size` returns the number of characters currently in the string
- The function `find` searches a string to locate the first occurrence of a particular substring
- The function `substr` returns a particular substring of a string
- The function `swap` is used to swap the contents of two string variables