Chapter 7: User-Defined Simple Data Types, Namespaces, and the `string` Type

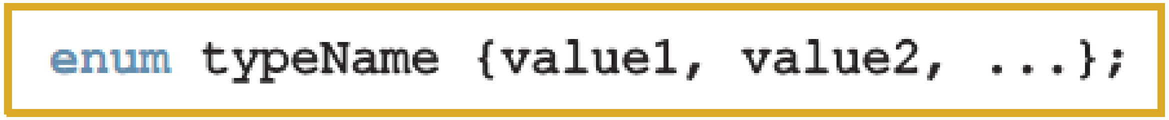
C++ PROGRAMMING:
Program Design Including Data Structures

Enumeration Type

- Data type: a set of values with a set of operations on them
- Enumeration type: 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

Enumeration Type (cont'd.)

- You can specify the name and the values, but not the operations
- Syntax:



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

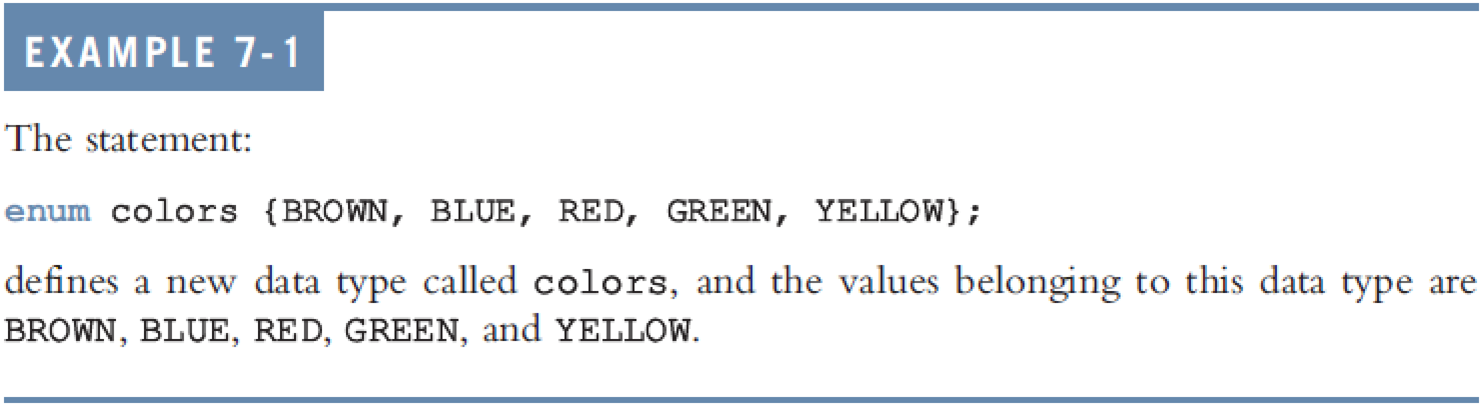
  - `value1`, `value2`, … are identifiers called enumerators
  - List specifies the ordering:

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

Enumeration Type (cont'd.)

- 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 same block
- Same rules apply to enumeration types declared outside of any blocks

Enumeration Type (cont'd.)

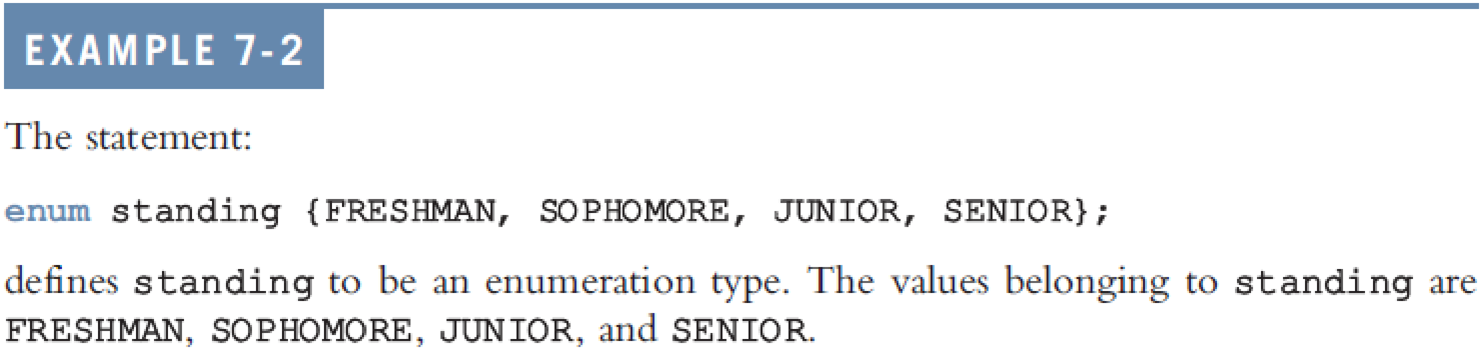


EXAMPLE 7-1

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`.



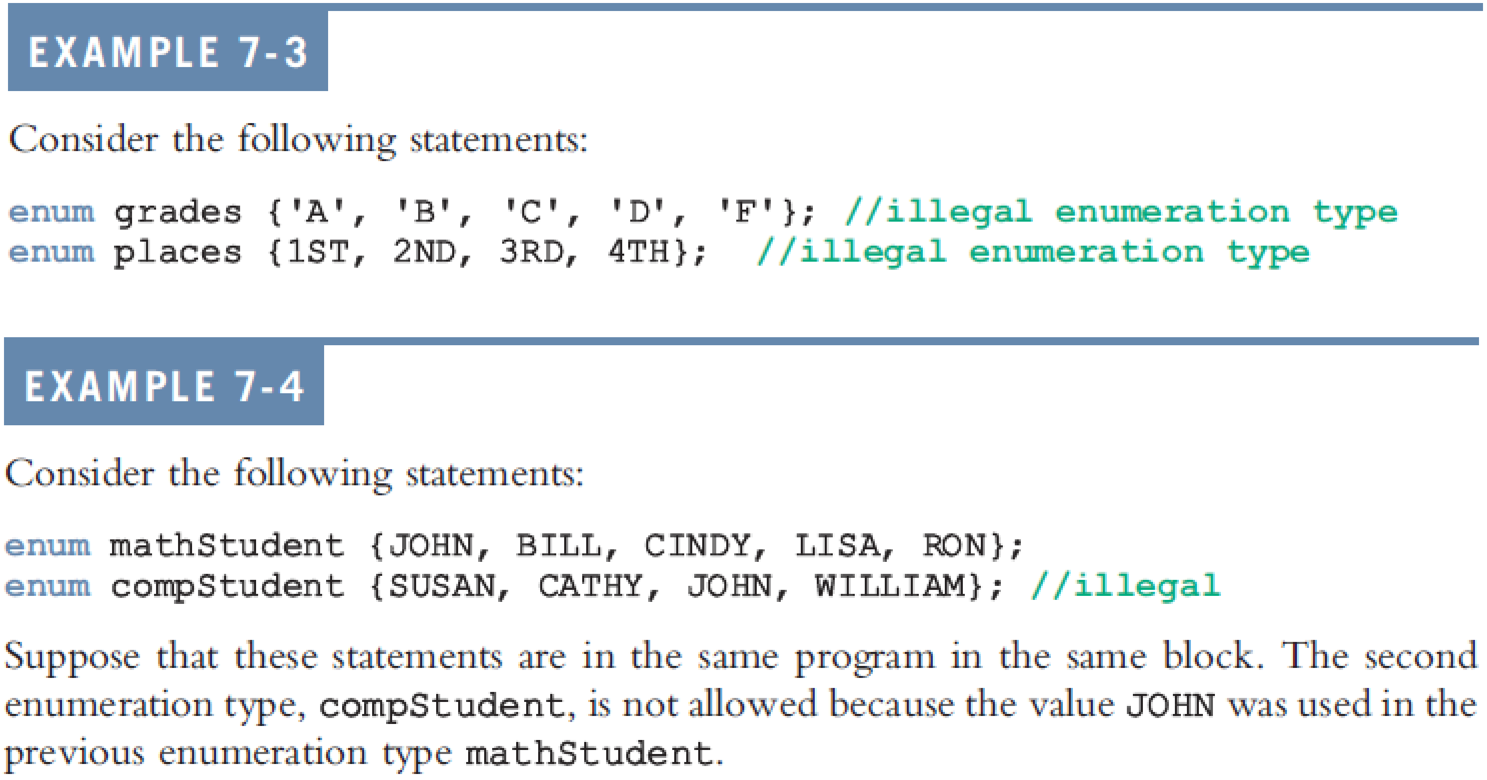
EXAMPLE 7-2

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 (cont'd.)



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
```

EXAMPLE 7-4

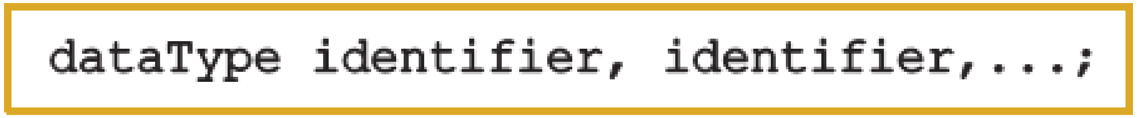
Consider the following statements:

```
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`.

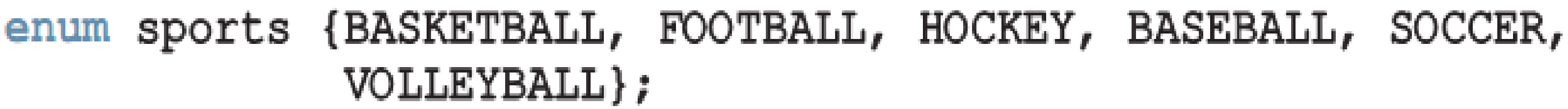
Declaring Variables

- Syntax:



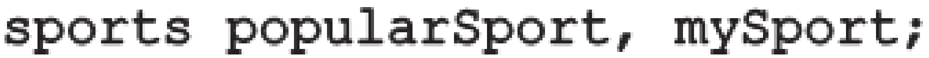
```
dataType identifier, identifier,...;
```

- Example:



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

  – Can declare variables such as:



```
sports popularSport, mySport;
```

Assignment

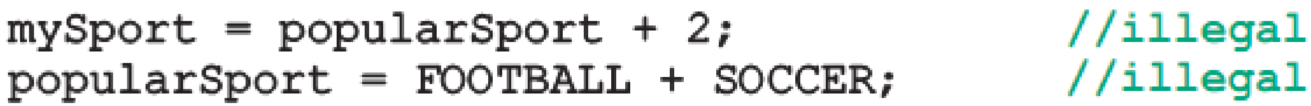
- Values can be stored in enumeration data types:

```
popularSport = FOOTBALL;
```

  - Stores `FOOTBALL` into `popularSport`

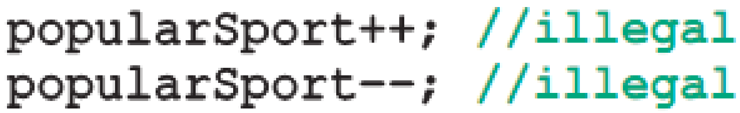
Operations on Enumeration Types

- No arithmetic operations are allowed on enumeration types :



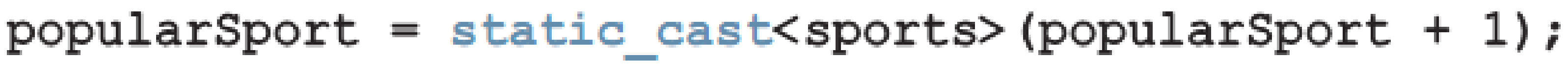
```
mySport = popularSport + 2;              //illegal
popularSport = FOOTBALL + SOCCER;        //illegal
```

- ++ and -- are illegal, too:



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

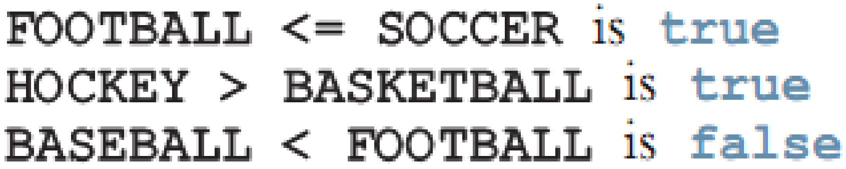
- Solution: use a static cast



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

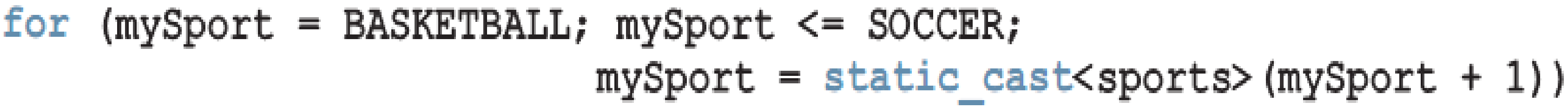
Relational Operators

- An enumeration type is an ordered set of values:



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

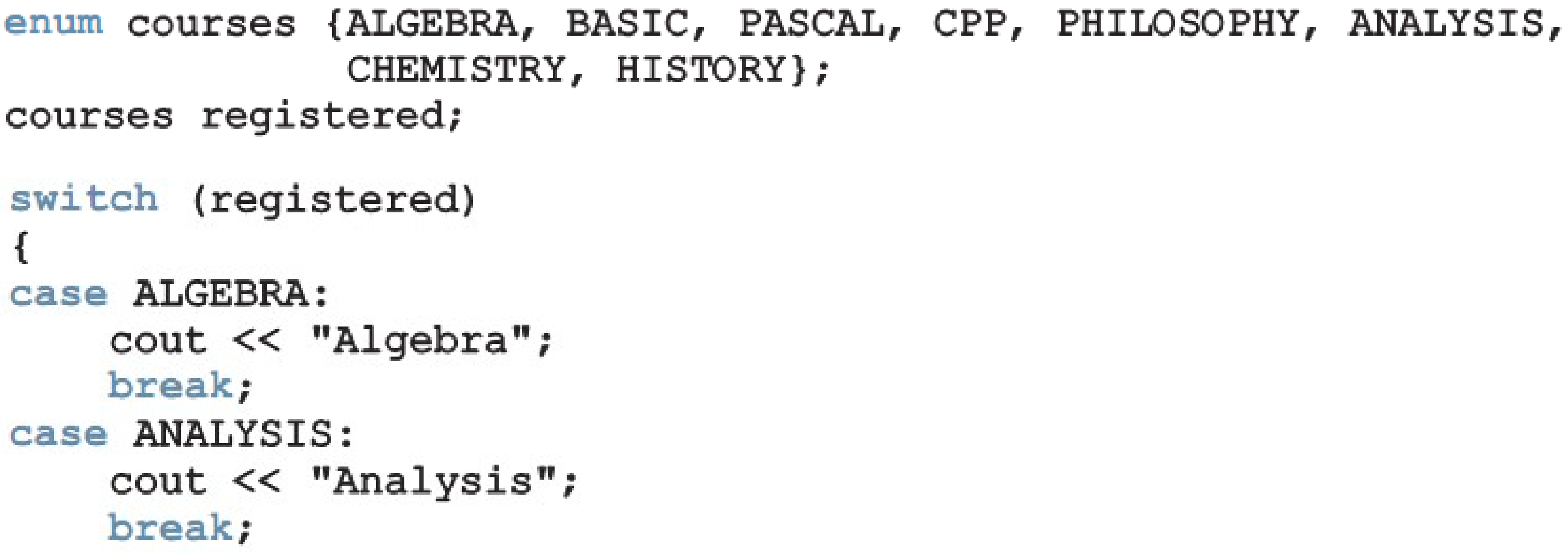
- An enumeration type is an integral data type and can be used in loops:



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

Input /Output of Enumeration Types

- An enumeration type cannot be input/output (directly)
  - Can input and output indirectly



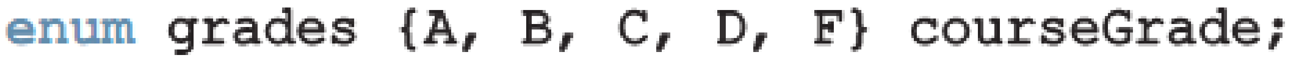
```
enum courses {ALGEBRA, BASIC, PASCAL, CPP, PHILOSOPHY, ANALYSIS,
              CHEMISTRY, HISTORY};
courses registered;

switch (registered)
{
case ALGEBRA:
    cout << "Algebra";
    break;
case ANALYSIS:
    cout << "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

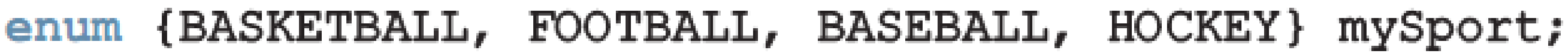
- Can declare variables of an enumeration type when you define an enumeration type:



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

Anonymous Data Types

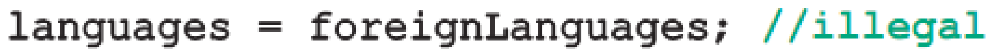
- Anonymous type: values are directly specified in the declaration, with no type name
- Example:



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

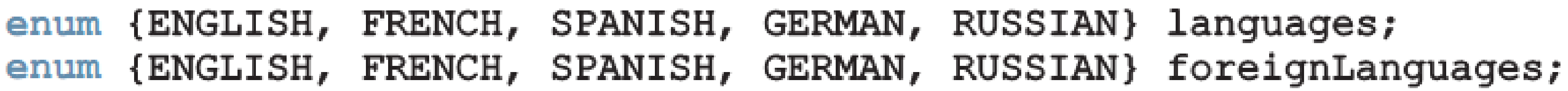
Anonymous Data Types (cont'd.)

- 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:



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

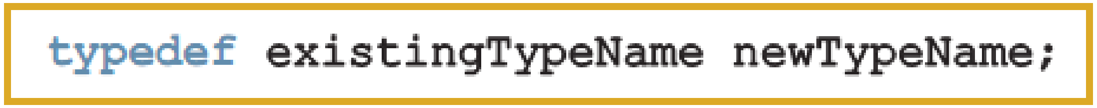
- Best practices: to avoid confusion, define an enumeration type first, then declare variables



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

typedef Statement

- typedef statement: used to create synonyms or aliases to a data type
- Syntax:



```
typedef existingTypeName newTypeName;
```

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

Namespaces

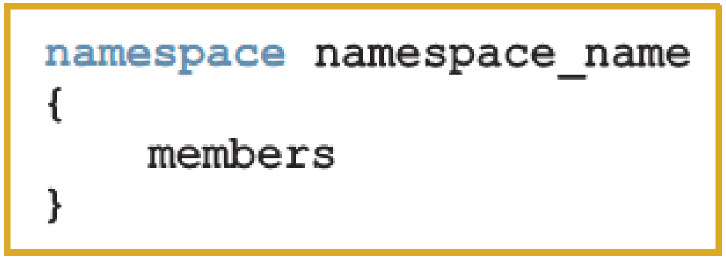
- ANSI/ISO standard C++ was officially approved in July 1998
- Most recent compilers are compatible with ANSI/ISO standard C++
- For the most part, standard C++ and ANSI/ISO standard C++ are the same
  - However, ANSI/ISO Standard C++ has some features not available in Standard C++

Namespaces (cont'd.)

- Global identifiers in a header file used in a program become global in the program
  - Syntax error occurs if a program's identifier has 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 (cont'd.)

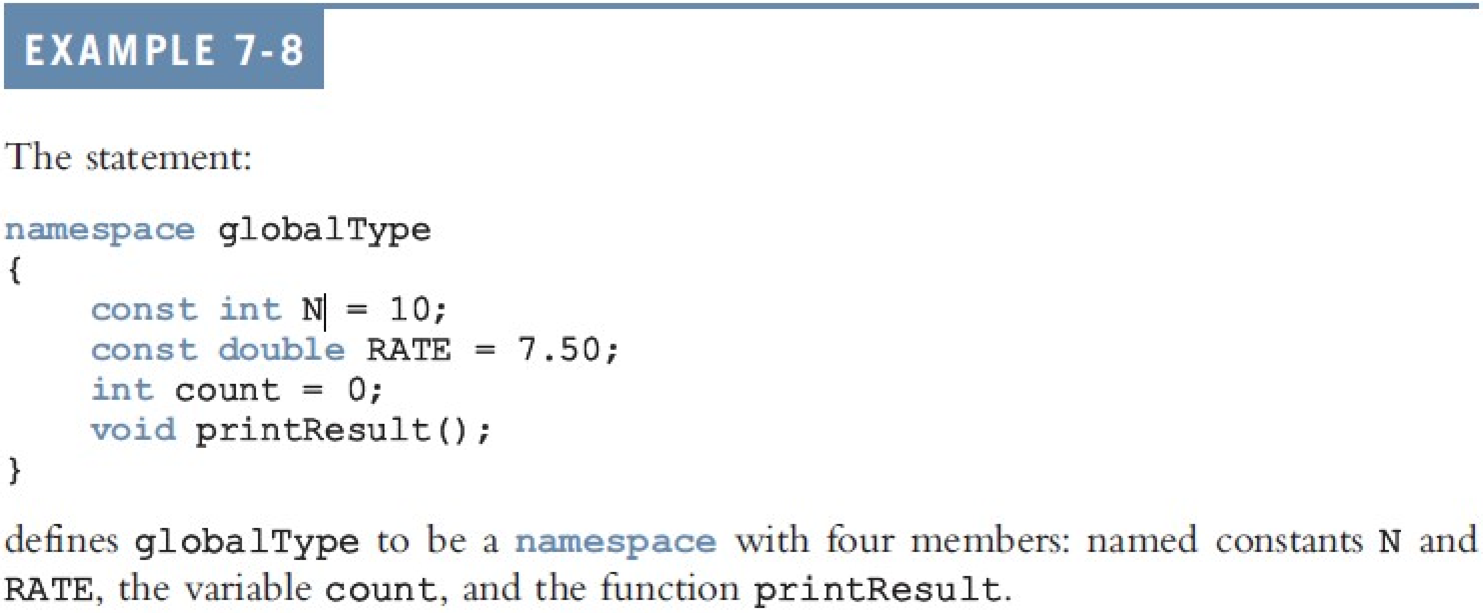
- ANSI/ISO Standard C++ attempts to solve this problem with the namespace mechanism
- Syntax:



```
namespace namespace_name
{
     members
}
```

  - Where `members` consist of variable declarations, named constants, functions, or another namespace

Namespaces (cont'd.)



EXAMPLE 7-8

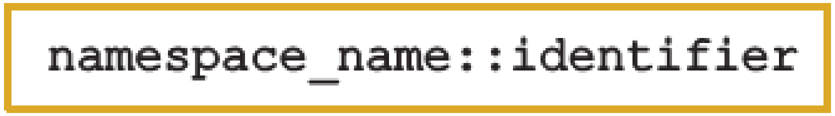
The statement:

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

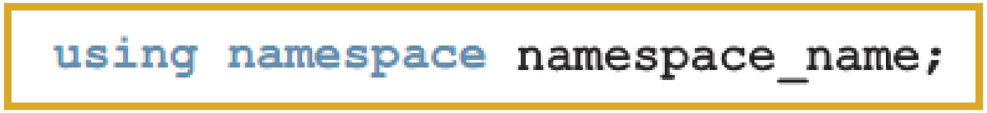
defines `globalType` to be a `namespace` with four members: named constants `N` and `RATE`, the variable `count`, and the function `printResult`.

Namespaces (cont'd.)

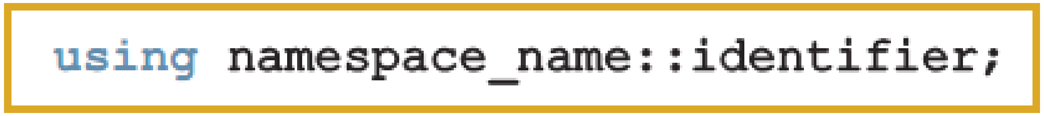
- A `namespace` member has scope local to the namespace
- A `namespace` member can be accessed outside the `namespace`:



```
namespace_name::identifier
```



```
using namespace namespace_name;
```



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

Namespaces (cont'd.)

- Examples:

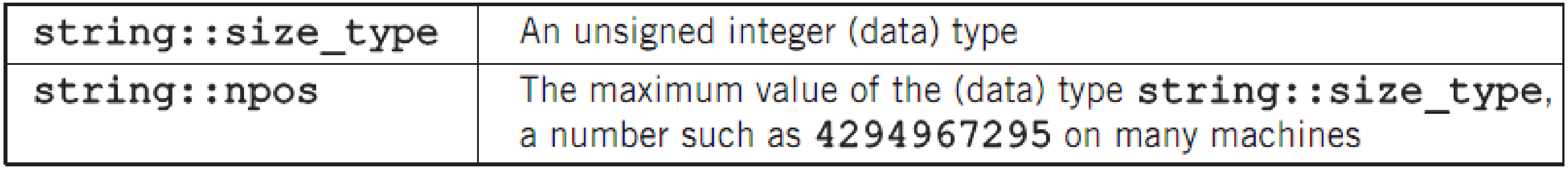
```
globalType::RATE
using namespace globalType::printResult();
using globalType::RATE;
```

- 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

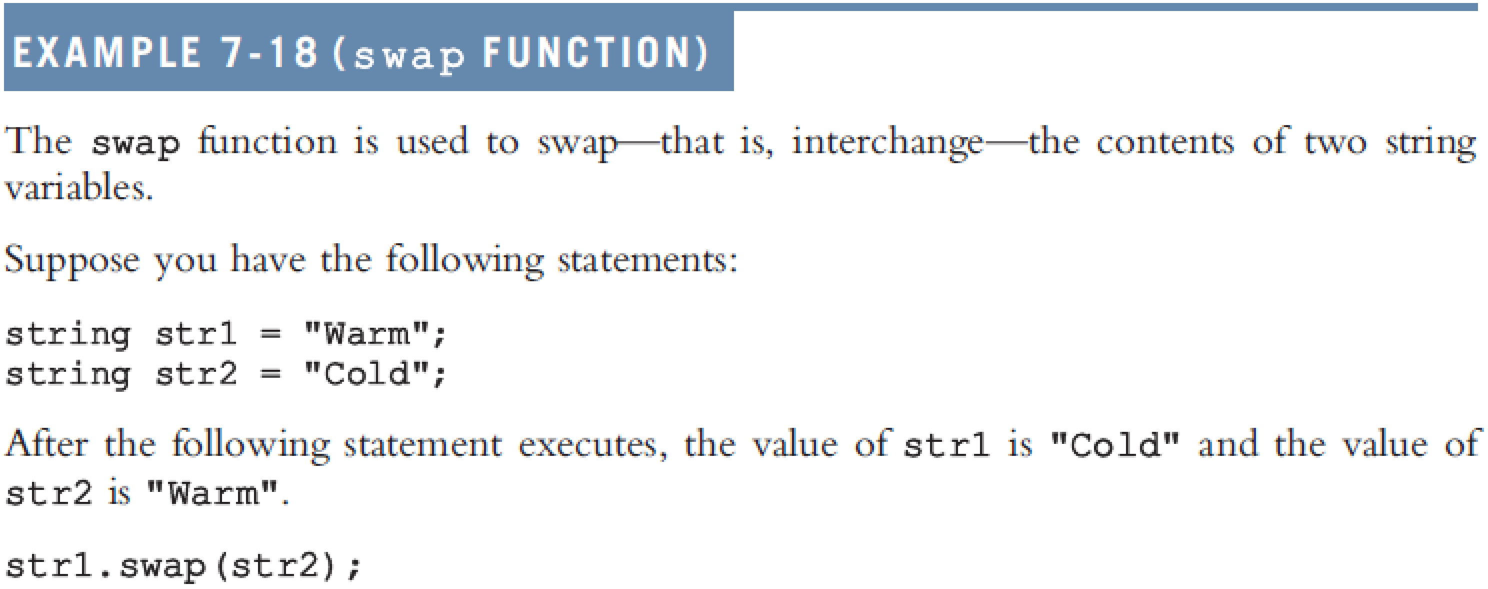
- To use data type `string`, a program must include the header file `string`
- A string is a sequence of 0 or more characters
  - The first character is in position 0
  - The second character is in position 1, etc.
- Binary operator `+` performs the string concatenation operation
- Array subscript operator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

Additional `string` Operations



`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 (swap FUNCTION)

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);
```

Summary

- 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

Summary (cont'd.)

- 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

Summary (cont'd.)

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