Enumeration

Lecture 6

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
- Sy

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

  - `value1`, `value2`, … are identifiers called enumerators

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};
```

```
sports popularSport, mySport;
```

we can declare the following variables:

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
```

- ```
  popularSport++; //illegal
  popularSport--; //illegal
  ```
  ll too:

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

- Solution: use a static cast:
```

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
  - 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)

Namespaces (continued)

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

```
namespace namespace_name
{
    members
}
```

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

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 [illegible] `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