나열형(enum)

- 나열형(enum)은 몇가지 한정된 값을 가지는 타입
- 가질 수 있는 모든 값들을 나열함으로써 정의

```
enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
```

- 나열형 타입 이름은 int, float 등과 동일한 방식으로 변수를 정의할 때 타입 이름으로서 사용

```
Grade curGrade = FRESH ;
```

나열형의 형변환

- **나열형 타입의 리터럴은 내부적으로 int 타입으로 처리**
- 첫번째 주어진 값을 0으로 해서 그 다음 값은 1로 처리
- **나열형 ➔ int 타입으로 묵시적으로 형변환**

```
int nextGrade = curGrade + 1 ;    // OK
Grade curGrade = 1 ;              // Error
```

- **int 값 ➔ 나열형 타입으로 명시적으로 형변환**

```
curGrade = Grade(nextGrade) ;
      또는
curGrade = static_cast<Grade>(nextGrade) ;
```

```
# include <iostream>
using namespace std ;

enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
int main() {
  // 현재 학년을 int 값으로 입력 받는다.
  int intGrade ;
  cin >> intGrade ;
  // int 값을 Grade 타입으로 변환한다.
  Grade curGrade = Grade(intGrade) ;
  if ( curGrade < FRESH || curGrade > SENIOR ) {
    cout << "Grade should be between 1 and 4₩n" ;
    return 0 ;
  }
  // 다음 학년을 구한다.
  if ( curGrade != SENIOR ) {
    int nextGrade = curGrade + 1 ;
    // Grade ==> int로의 묵시적 형 변환
    curGrade = Grade(nextGrade) ;
    // curGrade = static_cast<Grade>(nextGrade) 과 동일
  }
  cout << curGrade << endl ; // int로 묵시적 형변환 됨
}
```

Good Design: 나열형의 활용

- 가독성을 위해서

int 타입 사용시	void sort(int data[], int size, int kind) ;
enum 사용시	enum SortKind {ASCEND, DESCEND} ; void sort(int data[], int size, SortKind kind) ;

- 결함 방지를 위해서

int 타입 사용시	int intArray[] = {20, 10, 30, 25} ; sort(intArray, 4, 3) ;

enum 사용시	int intArray[] = {20, 10, 30, 25} ;
	sort(intArray, 4, 0) ; // 컴파일 오류
	sort(intArray, 4, ASCEND) ; // OK

Scoped enum (Since C++11)

- C++11 introduces so-called scoped enums, which are declared with the keywords enum class
  - Because enumerators are scoped, you can use the same enumerator name in multiple scoped enumerations.

```
enum class Color { RED, GREEN, BLUE };
enum class TrafficSign { RED, YELLOW, GREEN };

TrafficSign sign = TrafficSign::RED;
if ( sign == TrafficSign::YELLOW )
  ...
```

  - The type after the colon must be an integral type. If you omit the colon and type, the compiler implicitly uses int

```
enum class flags : unsigned {
  boolalpha, showbase, showpoint, showpos, skipws
};
```

Scoped enum (Since C++11)

- The enumerators of a scoped enum will **not automatically convert to int**

```
enum TrafficSign { RED, YELLOW, GREEN };          // unscoped enum
int main() {
    TrafficSign initial = RED;
    int value = initial;          // OK
}
```

```
enum class TrafficSign { RED, YELLOW, GREEN };  // scoped enum
int main() {
    TrafficSign initial = TrafficSign::RED;
    int value = initial;          // ERROR: cannot covert 'TrafficSign' to 'int'
    int value = static_cast<int>(initial); // OK
}
```

Scoped enum (Since C++11)

- Define operations on enum class

```
enum class TrafficSign { RED, YELLOW, GREEN };

void operator++(TrafficSign& sign) {
    const int nextValue = (sign == TrafficSign::GREEN) ?
        static_cast<int>(TrafficSign::RED) : static_cast<int>(sign)+1;
    sign = static_cast<TrafficSign>(nextValue);
}
int main() {
    TrafficSign initial = TrafficSign::RED;
    ++initial;
}
```

```
enum class Day { mon, tue, wed, thu, fri, sat, sun };

Day& operator++(Day& d) {
    return d = (d == Day::sun) ? Day::mon : static_cast<Day>(static_cast<int>(d)+1);
}
Day today = Day::sat;
Day tomorrow = ++today;
```