객체지향 프로그래밍 노트
const 선언 읽는 법

강대기

2007년 10월 9일

C++에 새로 등장한 키워드 중 하나가 “const” 이다. 그런데 const의 사용은 혼동하기 쉬우므로 이에 대해 논해보고자 한다.

제 1 절 기본적인 내용

const는 상수를 선언하기 위한 방법 중 진보된 방법이다. define 매크로에 의해 방법에 비해, 특정 형을 선언할 수 있고, 스코프 즉 범위를 가지며, 복잡한 구조체나 배열 등에 대해서도 선언이 가능한 강점들을 가지고 있다.

쉬운 예로 정수를 상수로 선언하기 위해서는 다음 두 가지 방법 중 편한 것을 쓰면 된다.

```
const int m = 10;
int const l = 20;
```

위의 m과 l은 모두 정수 상수이다. 주의할 것은 상수는 처음에 값이 선언되고 변화될 수 없으므로, 처음에 선언하고 정의할 때 이미 초기화도 시켜야 한다는 것이다.

제 2 절 포인터와 const를 혼용한 선언

포인터와 상수가 같이 쓰이면 혼동되기 쉽다.

다음의 프로그램을 보도록 하자.

```
#include <iostream>
using namespace std;
int main()
{
  char aword[100] = "abcd";
  const char * a = aword;
  a++;
//  a[0]='1';  // 1. 배열의 내용을 바꿀 수 없다.
```

```
  char bword[100] = "pqrs";
  char const * p = bword;
  p++;
//  p[0]='1'; // 2. 배열의 내용을 바꿀 수 없다.

  char cword[100] = "pqrs";
  char * const x =cword;
//  x++;  // 3. 포인터 상수를 변경할 수 없다.
  x[0]='1';

  cout << a << endl;
  cout << p << endl;
  cout << x << endl;
}
```

이 프로그램을 실행하면 결과는 다음과 같다.

```
bcd
qrs
1qrs
```

위 프로그램에서 a 는 상수 문자열에 대한 포인터이다. 이를 읽는 방법은 우선 a에 대한 선언을 다시 보도록 하자.

```
const char * a = aword;
```

일단 a 에서 시작한다. *가 a 의 왼쪽 옆에 있으므로 a는 pointer이다. 그 다음 char 이 왼쪽 옆에 있으므로 a는 pointer to char 이다. 그 다음 const 가 왼쪽 옆에 있으므로, a는 pointer to char that is const 인 것이다. 이를 해석하면 a 는 const 인 char 의 포인터가 된다.

이제 p의 경우를 보자. p의 선언은 다음과 같다.

```
char const * p = bword;
```

간단히 말하면 p는 pointer to const char, 즉 a 처럼 const 인 char 의 포인터이다.

따라서, a와 p의 경우, char 들이 상수이므로 char 배열의 내용을 바꿀 수 없는 것이다.

이제 x 를 보자.

```
char * const x =cword;
```

x는 바로 왼쪽이 const 이므로, 일단 상수이다. 즉 영어로는 x is const 인 것이다. 그 다음 * 가 오므로, x is const pointer가 된다. 즉 x는 포인터 상수이다. 그 다음 char 가 오므로, 결론적으로 x is const pointer to char 이다. 즉 x 는 문자열에 대한 포인터 상수인 것이다.

따라서 x++과 같이 포인터 상수를 변경할 수는 없으나, 그것이 가리키는 배열의 내용을 바꿀 수는 있는 것이다.

이는 정수에 대한 포인터도 마찬가지이다. 다음을 보도록 하자.

```
// 아래에서 n 을 일부러 const int로 하였다.
// 이에 대한 논의는 다음 절에서 한다.
const int n = 10;
const int *p = &n;
int const *q = &n;
int m = 20;
int * const r = &m;
```

p 와 q는 모두 상수에 대한 포인터이다. 포인터로서 p 나 q 자체는 변할 수 있으나, p나 q가 가리키고 있는 정수값은 p나 q를 통해서는 변경할 수 없다.

r은 상수인 포인터, 즉 포인터 상수이다. r이 가리키는 것은 r을 통해 바꿀 수 있으나, r 자체는 다른 것으로 가리키게 바꿀 수 없다.

그렇다면 상수를 가리키는 포인터 상수를 정의할 수 있을까? 물론이다. 다음과 같이 한다.

```
const int n = 10;
const int * const p = &n;
```

이런 경우, p는 포인터 상수이므로 가리키는 것을 다른 것으로 바꿀 수 없고, 그 가리키는 것이 const int이므로 그 가리키는 내용도 변경할 수 없다.

이제 연습 문제 삼아 다음의 선언들을 읽어보겠다.

- char ** p1; ⟶ p1은 pointer to pointer to char ⟶ p1은 문자에 대한 포인터의 포인터
- const char **p2; ⟶ p2는 pointer to pointer to const char ⟶ p2는 문자 상수에 대한 포인터의 포인터
- char * const * p3; ⟶ p3는 pointer to const pointer to char ⟶ p3는 문자에 대한 포인터 상수에 대한 포인터
- const char * const * p4; ⟶ p4는 pointer to const pointer to const char ⟶ p4는 문자 상수에 대한 포인터 상수에 대한 포인터
- char ** const p5; ⟶ const pointer to pointer to char ⟶ p5는 문자를 가리키는 포인터에 대한 상수 포인터
- const char ** const p6; ⟶ const pointer to pointer to const char ⟶ p6
- char * const * const p7; ⟶ const pointer to const pointer to char ⟶ p7
- const char * const * const p8; ⟶ const pointer to const pointer to const char ⟶ p8은 상수 문자에 대한 상수 포인터에 대한 상수 포인터

제 3 절 포인터와 const

기본적으로 const 개체에 대한 포인터는 const 가 아닌 것을 가리키는 포인터로 초기화될 수 있다.

```
int n = 10;
const int * p = &n;
```

위에서 n은 여전히 변경가능하지만, n을 p를 통해 고칠 수는 없음은 물론이다.

또한 const 변수의 주소를 const 를 지시하는 포인터에는 대입할 수는 있으나, const 변수의 주소를 const가 아닌 포인터에 대입할 수 없다.

```
const float g_earth = 9.80;
const float * pe = &g_earth; // 가능하다.

const float g_moon = 1.63;
float * pm = &g_moon; // 불가능하다.
```

이에 관련된 논의는 교재 379 쪽에 나와있다.

즉, 기본적으로는 C++는 const 형에서 const가 아닌 형으로 형 변환을 할 수 없다. 그러나 C++ 컴파일러는, 다른 언어의 컴파일러들이 그러하듯, 프로그래머가 일부러까지 저지르는 바보 짓에 대해서는 막을 수 없다. 예를 들어 const에 대한 강제적인 형 강제(explicit type casting)를 허용할 수도 있다. 그러한 경우, 어떠한 결과를 낼지는 정의되어 있지 않으므로 피해야 한다.

```
const int x = 4; // x 는 const 로 변경될 수 없다.
const int* pX = &x; // pX 포인터 상수에 대해
                    // 상수 x 의 주소로 초기화할 수 있다.
cout << x << endl;  // "4" 가 출력된다.

int* pX2 = (int *)pX; // (const int*) 인 pX 를 일부러 (int*) 로
                      // 형 강제한다.
*pX2 = 3; // 형 강제된 pX2가 가리키는 주소에 3를 대입한다.
cout << x << endl; // 과연 결과는? 예측 불허...
```

참고로 필자가 Microsoft Visual Studio 2005로 수행해 본 결과, 두번째 경우에도 "4"가 출력된다.

함수나, 멤버 함수, 또는 기타 변수 선언에서 const를 사용할 수 있다면 사용하는 게 좋다. 예를 들어 함수에서 포인터 전달 인자가 가리키는 것이 변경되지 않아야 하는 거라면 그것을 const로 선언해야 한다. 그렇게 되면, 실수로 데이터를 변경하는 프로그래밍 에러를 막을 수 있다. 또한, const를 사용하는 함수는 const 또는 const가 아닌 전달 인자를 모두 처리할 수 있지만, 함수 원형에서 const가 생략되면 const가 아닌 데이터만 처리할 수 있다.

```
const int months[12] = {31,28,31,30,31,30,31,31,30,31,30,31};

int sum(int arr[], int n); // const int arr[]로 선언해야 한다.
...
int j = sum(months, 12); // 허용되지 않는다.
```