객체지향 프로그래밍 노트
스트링 리터럴(String Literal)

강대기*

2007년 9월 30일

제 1 절 문제 정의

다음의 프로그램을 보도록 하자. "Yello!"를 출력하고 싶은 경우이다.

```
#include <iostream>
using namespace std;
int main()
{
  char * x ="Hello!";
  x[0]='Y'; // "Yello!"
  cout << x << endl;
}
```

이 프로그램을 대부분의 최신 Visual Studio나 gnu g++에서 컴파일해보면 아무런 에러나 경고도 발생시키지 않을 것이다. 그러나, 만들어진 실행 파일을 실행하려는 순간, 실행 시간 에러가 나면서 작동이 되지 않을 것이다.

참고로 필자는 대부분의 컴파일러에서 그러한 것으로 알고 있지만, 아무 문제 없이 수행되는 경우도 있을 것이다. 제대로 수행된다면, 그냥 이 문서를 읽기만 하면 된다. 그러나 적어도 필자가 윈도우즈 상에서 Visual Studio 2005를 가지고 테스트해보고, 리눅스에서 gnu g++ 3.4.6으로 테스트해 본 결과, 두 경우 모두 실행 시간에 에러를 발생시켰다. 이 두 컴파일러들이 많이 쓰인다는 점을 감안할 때, 이 문서에서 논의하는 바는 분명히 알고 있어야 할 부분인 것이다.

제 2 절 스트링 리터럴 문제이란?

스트링 리터럴은 단순히 스트링 상수같은 것이다. 스트링 문자열 자체로 그 내용이 변하지 않는 것이다. 상수란 것이 정의상 항상 같은 수라는 점에서, 스트

링의 경우는 스트링 상수보다는 스트링 리터럴이라는 호칭이 더 적절하다고 하겠다.

예를 들어 “Hello”나 “Dongseo”, 그 외 어떤 것이든 “ 와 ” 안에 들어가는 문자들을 단순히 스트링 리터럴이라고 한다. 즉 스트링 리터럴은 우리가 이미 잘 알고 있는 것이라 하겠다.

참고로 C++에서는 L“Hello” 처럼 문자열 따옴표 앞에 붙는 스트링 리터럴도 있다. 이것은 와이드 스트링 리터럴(wide string literal)로 스트링 리터럴을 구성하는 문자들의 형(type)이 wchar_t이다. wchar_t를 교재에서는 확장 char 형이라고 하고 있다.

그런데 이 스트링 리터럴을 C나 C++ 등의 언어에서 다루는 데 있어 한 가지 잘 알려진 문제가 있다. 스트링 리터럴 문제(issue of string literal)란, C++나 C 언어에 대한 기술 문서나 명세에서 근본적으로 스트링 리터럴이 변경될 수 있는가(mutable) 그렇지 않은가(immutable)에 대해서 확실하게 정의를 내리지 않았다는 것이다.

그런데, 실제로 컴파일러나 만들어진 실행 파일에서 스트링 리터럴이 변경되지 않도록 했을 경우 상당히 성능이 좋아지거나, 프로그램을 개발할 때에서 개발이 편해지는 경우가 많다. 예를 들어 처음에 스트링 리터럴을 가지고 포인터 변수를 정의를 하고 초기화할 때에도 변경되지 않도록 하면, 더 간단하게 정의되고 함수의 인자로 사용될 때에도 혼돈이 없이 더 편하게 사용되어질 수 있다. 또한 그 외에도 멀티 쓰레드 응용 프로그램에서 스트링 리터럴이 바뀌지 않는다는 것이 보장되면 동기화를 할 필요가 없어지거나 변경되었으지도 모른다고 고민할 필요가 없을 것이다.

예를 들어 다음과 같은 경우를 보자.

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

void funcI(int i) { i++; }
void funcS(char *s) { s[0]='Y'; } // 여기서 에러가 안난다고 치자.

int main()
{
  int x = 0;
  funcI(x);
  cout << x << endl; // x == 0

  char *y ="Hello!";
  funcS(y); // 원래는 에러가 나겠지만, 안난다고 가정하였다.
  cout << y << endl; // y == "Yello!" 일 것이다.

  funcI(0);  // 0은 그대로 0일 뿐...
  funcS("Dongseo"); // 그런데 이 경우는?
}
```

즉, 마지막의 funcS(”Dongseo”);의 경우, 변수(variable)가 아니라 값(value)일 뿐인 “Dongseo”의 내부값이 바뀐 채로 기억 장소의 어딘가에 남게 된다. 물론

이러한 점에 대해 과거 C 프로그래머들처럼 포인터의 위험성으로 인지하고 있으면 되겠으나, 결국 이런 모순이 C 언어가 문법 이론적으로 그다지 엄밀하지 않게 됨을 알게 될 것이다. (C 언어를 잘 쓰는 사람들에겐 이것도 하나의 강점이긴 하다.)

이런 점들을 극복하고자 최신의 C++ 컴파일러에서는 포인터로 받은 스트링 리터럴을 변경하는 것을 결국 금지하고 있는 것이다. C++ 언어의 정의 상으로는 문제가 없으므로 컴파일은 되지만, 실제로 실행하는 과정에서 에러를 발생하게 되는 것이다.

그렇다면, 스트링을 쓰면서 변경하고 싶을 때는 어떻게 해야 할까? 그럴 경우에는, char 배열을 선언해서 스트링 리터럴을 초기값으로 받아서 복사하는 방법을 사용하면 문제가 없다. 이에 대한 코드는 다음과 같다.

```
#include <iostream>
using namespace std;
int main()
{
  char x[] ="Hello!";
  x[0]='Y'; // "Yello!"
  cout << x << endl;
}
```

제 3 절 스트링 리터럴 풀링 (String Literal Pooling)

더 재미있는 사실은 똑같은 스트링 리터럴인 경우, 같은 파일 내에서 여러 개가 나와도 다 같은 주소로 관리하고 있다는 것이다. 이는 이미 자바(Java)에서도 하고 있는 방식이다.

이 점은 다음을 실행해 보면 더 이해가 빠를 것이다.

```
#include <iostream>
using namespace std;
int main()
{
  char *x ="Hello!";
  char *y ="Hello!";
  cout << x << endl;
  cout << y << endl;
  cout << (int *)x << endl;
  cout << (int *)y << endl;
}
```

이 프로그램을 Visual Studio 2005에서 실행한 결과이다.

```
Hello!
Hello!
004177FC
```

```
004177FC
```

또한 다음은 gnu g++에서 실행한 결과이다.

```
Hello!
Hello!
0x8048a00
0x8048a00
```

서로 다른 x와 y임에도 같은 주소를 가리키고 있음을 알 수 있다.