C++에 대한 다섯 가지 일반적인 미신들

Bjarne Stroustrup

Morgan Stanley, Columbia University, Texas A&M University

Korean Translation: Chan-Ho Chris Ohk (utilForever@gmail.com)

1. 서문

C++에 대한 5 가지 일반적인 미신들에 대한 실체를 살펴볼 것이다.

1. “C++을 이해하려면, C 를 먼저 배워야 한다.”
2. “C++은 객체 지향 언어다.”
3. “신뢰할 수 있는 소프트웨어를 위해, 가비지 컬렉션이 필요하다.”
4. "효율성을 위해, 저수준 코드를 작성해야 한다.”
5. “C++은 대규모의 복잡한 프로그램 만을 위한 언어다.”

만약 여러분 또는 동료들이 이러한 미신들을 믿고 있다면, 이 글이 여러분 또는 동료들에게 많은 도움이 될 것이다. 미신들 중 일부는 한 때 누군가에게, 또는 어떤 작업에서는 사실이었다. 그러나 오늘날의 C++은 최신 ISO C++ 2011 컴파일러 및 도구들을 널리 사용하고 있기에 단순히 미신에 불과하다.

필자는 이러한 미신들을 자주 들었기 때문에 “일반적”이라고 표현했다. 사람들은 이러한 미신들을 다양한 이유로 지지하지만, 당연한 것으로 받아들이는 경우도 있다. 심지어 이러한 미신들로 인해 C++을 사용할 프로그래밍 언어에서 제외시키기도 한다.

각 미신에 대한 실체를 파악하기 위해선 장문의 글이나 책을 집필해야 하지만, 여기서 필자의 목표는 간단히 문제를 제기하고 필자의 의견을 설명하는 것이다.

2. 미신 1: “C++을 이해하려면, C 를 먼저 배워야 한다.”

아니다. C++을 사용해 기초 프로그래밍을 배우는 것이 C 를 사용해 배우는 것보다 쉽다.

C 의 대부분은 C++의 부분 집합이지만, 처음 배우기 위한 가장 좋은 부분 집합은 아니다. 그 이유는 C 는 표기법 지원, 타입 안전성, 그리고 간단한 작업을 단순화하기 위해 C++에서 제공하는 편리한 표준 라이브러리 등이 부족하기 때문이다.

이메일 주소를 작성하는 간단한 함수를 생각해 보자.

```
string compose(const string& name, const string& domain)
{
        return name + '@' + domain;
}
```

이를 다음과 같이 사용할 수 있다.

```
string addr = compose("gre", "research.att.com");
```

물론, 실제 프로그램에서 모든 인수가 리터럴 문자열은 아닐 것이다.

C 버전은 문자 조작 및 메모리 관리를 직접 해야된다.

```
char* compose(const char* name, const char* domain)
{
        char* res = malloc(strlen(name) + strlen(domain) + 2);    // 문자열, '@', 그리고 0 을 위한 공간
        char* p = strcpy(res, name);
        p += strlen(name);
        *p = '@';
        strcpy(p + 1, domain);
        return res;
}
```

이를 다음과 같이 사용할 수 있다.

```
char* addr = compose("gre", "research.att.com");
// …
free(addr);       // 모든 작업이 끝나면 메모리를 해제한다.
```

어떤 버전으로 가르칠 것인가? 어떤 버전이 사용하기 쉬운가? 정말 C 버전을 사용하는 것이 올바를까? 확실한가? 왜 그런가?

마지막으로, 어떤 버전의 compose() 함수가 더 효율적이라고 생각하는가? 물론 C++ 버전이다. 왜냐하면 인수로 들어오는 문자열의 길이를 셀 필요도 없으며 인수로 들어오는 짧은 문자열을 위한 여유 공간(동적 메모리)을 사용하지 않아도 되기 때문이다.

2.1. C++ 배우기

이런 예가 하나만 있는 것은 아니다. 필자는 전형적인 예제라고 생각한다. 그렇다면 많은 선생님들이 왜 "C 를 먼저 배워야 한다는" 방식을 주장하는가?

- 오랫동안 했던 것이기 때문에
- 교육 과정이 요구하는 것이기 때문에
- 선생님들이 젊었을 때 배웠던 방법이기 때문에
- C 는 C++보다 작아서 사용하기에 간단하다고 가정하기 때문에
- 어쨌든 학생들은 조만간 C (또는 C++의 부분 집합인 C)를 배워야 하기 때문에

그러나, C 는 C++의 부분 집합으로서 먼저 배우기에 가장 쉽거나, 또는 가장 유용한 언어가 아니다. 게다가, C++을 어느 정도 알고 나면, 부분 집합인 C 를 쉽게 배울 수 있다. C++을 배우기 전에 C 를 배우게 되면 C++에서 쉽게 피할 수 있는 오류를 겪게 되고, 개선하기 위한 기술들을 배워야 한다.

C++를 가르치기 위한 현대적인 접근법을 알고 싶다면, C++로 배우는 프로그래밍의 원리와 실제 (Programming: Principles and Practice Using C++) [13]을 참고하라. 물론 이 책의 끝부분에 C 를 사용하는 방법을 알려준다. 이 책은 여러 대학에서 프로그래밍을 처음 시작하는 수만명의 학생들이 합리적으로, 성공적으로 배우는데 사용되어왔다. 이 책의 2 판은 쉽게 배울 수 있도록 C++11 및 C++14 표준을 사용한다.

C++은 C++11 [11-12]을 통해 초보자에게 더욱 가까워지고 있다. 예를 들어, 표준 라이브러리에 있는 벡터(Vector)는 다음과 같이 연속된 요소들로 초기화된다.

```
vector<int> v = {1, 2, 3, 5, 8, 13};
```

C++98 에서는, 배열에서만 이런 연속된 요소들로 초기화할 수 있었다. C++11 에서는, 원하는 타입에 맞춰 초기화 리스트 {}을 인자로 받는 생성자를 정의할 수 있다.

또한 이 벡터에 범위 기반의 반복문을 사용할 수 있다.

```
for (int x : v) test(x);
```

이 구문은 v 의 각 요소에 대해 test() 함수를 한 번 실행한다.

범위 기반의 반복문은 모든 시퀀스(Sequence)에 사용할 수 있으므로, 초기화 리스트를 직접 사용해 해당 예제를 간단하게 만들 수 있다.

```
for (int x : {1, 2, 3, 5, 8, 13}) test(x);
```

C++11 의 목적 중 하나는 간단한 일을 간단하게 만드는 것이다. 물론, 성능이 저하되진 않는다.

성능

C

C++ 사용

라이브러리

가르침