시작하면서

C++ 프로그래밍에 대한 새로운 접근

이 책을 읽는 독자들은 단기간 내에 유용한 C++ 프로그램을 작성하길 원할 것입니다. 따라서, C++에서 가장 유용한 부분들을 먼저 설명하겠습니다. 이러한 전략은 얼핏 보면 당연하다고 생각할지 모릅니다. 하지만, C++가 C를 기반으로 만들어졌다는 사실을 생각해 보면, C를 먼저 가르치지 않는다는 것은 가히 혁신적이라 할 수 있습니다. 대신, 상위 레벨의 데이터 구조체들을 먼저 사용합니다. 그리고, 나중에 그러한 데이터 구조체들의 기본적인 내용에 대해서 설명합니다. 이러한 접근 방식은 여러분들이 자신만의 C++ 프로그램을 즉시 작성할 수 있도록 해줄 것입니다.

우리의 접근 방식에 있어서 남다른 면이 또 하나 있습니다. C++ 언어나 라이브러리 기능에 대해 설명하기 보다는 문제를 푸는 데에 더 신경을 썼다는 것입니다. 물론, 기능에 대해서도 설명하긴 하지만, 기능을 설명하기 위하여 프로그램을 사용하는 것이 아니라, 프로그램을 설명하기 위하여 기능을 설명합니다.

이 책은 C++ 프로그래밍을 가르쳐 주는 책이지, C++ 기능만을 가르쳐 주는 책은 아닙니다. 따라서, 이미 어느 정도 C++를 알고 있거나, 언어를 더 자연스럽고 효과적으로 사용하려는 독자들에게 특히 도움이 됩니다. C++를 새로 배우는 사람들은 언어의 구조적인 내용만을 배우고, 정작 언어를 사용하여 여러 문제를 어떻게 해결할지에 대해서는 배우지 못하는 경우가 너무나도 많습니다.

이러한 접근 방식은 초보자와 경력 프로그래머 모두에게 통합니다

우리는 매년 여름 스탠포드 대학에서 일주일의 C++ 집중 강좌를 진행했습니다. 초기에는 전통적인 접근 방식을 사용했습니다. 즉, 학생들이 이미 C를 알고 있다고 가정하고, 먼저 클래스를 정의하는 법을 가르쳐 준 다음, 나머지 부분에 대하여 체계적으로 진행해 나갔습니다. 약 이틀 정도가 지나자, 학생들이 혼란스러워하거나 좌절감을 느낀다는 것을 알게 되었습니다. 하지만, 그들이 유용한 프로그램을 작성할 수 있게 되자, 그러한 문제가 사라졌습니다. 어느 정도 수준에 다다르자, 배우는 속도도 빨라졌습니다.

그 당시 C++ 구현시스템 중에 최신 표준 라이브러리를 쓸만하게 구현한 것이 있었는데, 그 구현 시스템을 얻게 된 뒤, 우리는 강좌를 재정비했습니다. 새로운 강좌는 처음부터 그 라이브러리를 사용했고, 먼저 유용한 프로그램을 작성하는 데에 집중했습니다. 그리고 나서, 학생들이 세부적인 것들을 활용할 수 있는 수준이 되었을 때, 세부적인 사항들을 설명했습니다.

결과는 대만족이었습니다. 강의한지 하루만에, 학생들은 예전 스타일의 강좌였다면 일주일이 걸려야 만들 수 있는 프로그램을 작성할 수 있게 되었습니다. 게다가, 그들은 좌절감도 느끼지 않았습니다.

추상화

이러한 접근 방식이 가능했던 이유는, C++와 그에 대한 우리의 이해가 날로 성숙해졌기 때문입니다. 그 결과, 초기의 C++ 프로그램과 프로그래머들이 의지해왔던 많은 하위 레벨의 개념들을 생략할 수 있었습니다.

세세한 것들을 생략할 수 있다는 것은 그 기술이 성숙화되고 있다는 증거입니다. 예를 들어, 초기의 자동차는 자주 고장이 났기 때문에, 모든 운전자들은 아마추어 정비공의 자질을 갖추어야 했습니다. 언제 고장날지 모르는 상황에서, 집에 돌아오는 방법을 모르고서 드라이브를 한다는 것은 무모한 일이었습니다. 하지만, 오늘날의 운전자들이 운송수단으로서 자동차를 운전할 때, 기계공학의 세부적인 내용을 알 필요는 없습니다. 다른 이유 때문에 기계공학을 공부할 필요는 있겠지만, 그것은 이것과는 전혀 별개의 이야기입니다.

우리는 추상화(abstraction)를 선택적 생략이라고 정의하고, 그것을 현대 프로그래밍에 있어서 가장 중요한 개념이라고 여깁니다. 여기에서 선택적 생략이란, 주어진 작업에 비교적 더 관련있는 아이디어에 집중하고, 나머지 부분들은 무시한다는 개념입니다. 성공적인 프로그램을 작성하는 중요한 열쇠는 문제의 어떤 부분은 고려하고, 어떤 부분은 고려하지 말아야 할지를 결정하는 것입니다. 모든 프로그래밍 언어는 유용한 추상화를 만들어 주는 툴을 제공합니다. 그리고, 모든 성공적인 프로그래머들은 그러한 툴들을 어떻게 사용할지 알고 있습니다.

추상화는 매우 유용한 것이라고 생각합니다. 따라서, 이 책의 내용을 추상화로 가득 채웠습니다. 물론, 그것들을 추상화라고 직접적으로 이야기하지는 않습니다. 왜냐하면, 추상화는 매우 다양한 형태로 존재하기 때문입니다. 추상화라는 용어 대신 함수, 자료구조, 클래스 및 상속이라는 용어를 사용할 것입니다. 이러한 모든 것들이 바로 추상화이기 때문입니다. 그러한 용어를 사용하기만 하는 것이 아니라, 책 전반에 걸쳐 이용할 것입니다.

만약, 추상화가 잘 설계되었고 적절한 추상화를 선택했다면, 그것들이 어떻게 동작하는지에 대한 세부적인 내용은 몰라도, 사용할 수는 있습니다. 자동차 정비공만이 자동차를 운전할 수 있는 것이 아닌 것과 마찬가지로, C++가 어떻게 동작하는지를 모두 이해해야만 C++를 사용할 수 있는 것은 아닙니다.

다루는 영역

만약, 여러분에게 C++ 프로그래밍이 절실하다면, 이 책이 여러분이 알고자 하는 모든 내용을 가르쳐 줄 수는 없겠지만, 여러분은 이 책의 모든 내용을 다 알아야 합니다.

이 말은 사실 그리 모순된 말은 아닙니다. 이만한 두께의 어떤 책도 여러분이 C++에 대해 알고자 하는 모든 것을 알려주지는 못합니다. 왜냐하면, 다른 프로그래머와 다른 애플리케이션은 다른 지식을 필요로 하기 때문입니다. 따라서, Stroustrup의 The C++ Programming Language (Addison-Wesley, 2000)와 같이 C++의 모든 내용을 다루는 책은 정작 자신에게는 필요 없는 내용까지도 모두 설명하고 있습니다. 여러분에게는 필요치 않을지 몰라도, 다른 사람에게는 필요할 수 있기 때문입니다.

한편, C++의 많은 부분은 두루두루 중요하기 때문에, 그 내용들을 이해하지 못하면 효율적으로 프로그래밍할 수 없습니다. 이 책에서는 그러한 부분들에 중점을 두고 있습니다. 이 책에서 제공하는 정보만으로도 여러 분야의 유용한 프로그램을 작성할 수 있습니다. 실제로, 현장에서 C++로 작성된 상용 시스템의 선임 프로그래머인 한 독자가 서평하기를, '이 책은 자신의 업무에 꼭 필요한 모든 기능들을 다루고 있다'라고 하기도 했습니다.

이러한 기능들을 사용하면, 진정한 C++ 프로그램을 작성할 수 있습니다. C++를 C나 다른 언어 스타일로 프로그래밍하는 것을 말하는 것이 아닙니다. 이 책의 내용들을 모두 소화해 낸다면, 여러분은 자신에게 무엇이 필요한지, 또 어떻게 해나가야 할지를 알게 됩니다. 아마추어 망원경 제작자들 사이에서 전해지는 격언 중에 이런 말이 있습니다. 3인치 짜리 반사경을 만들어본 다음에 6인치 짜리 반사경을 만들기는 쉬워도, 처음부터 6인치 짜리 반사경을 만들기는 어렵다.

여기에서는 표준 C++만을 다룹니다. 다른 독자적인 확장은 무시합니다. 이러한 접근법의 장점은 여러분들에게 가르치는 프로그램은 어디에서든 동작할 수 있다는 것입니다. 하지만, 그것은 윈도우 환경에서 실행되는 프로그램을 작성하는 법은 다루지 않겠다는 뜻을 내포합니다. 왜냐하면, 그러한 프로그램들은 특정 환경이나 특정 제품에 종속되어 있기 때문입니다. 만약 특정 환경에서만 동작할 프로그램을 작성하고자 한다면, 그 내용은 이 책말고 다른 책에서 찾아보기 바랍니다.

하지만, 그렇다고 해서 이 책을 덮으라는 말은 아닙니다! 우리의 접근 방식은 보편적인 것들입니다. 따라서, 나중에 어떠한 환경에서 작업하더라도 써먹을 수 있습니다. 아무튼, 원하는 GUI 애플리케이션에 관한 책을 사서 읽되, 단 이 책은 꼭 먼저 읽어야 합니다.

C 및 C++ 경력 프로그래머들에게 한마디

새로운 프로그래밍 언어를 배울 때, 여러분이 기존에 알고 있는 언어와 유사한 스타일로 프로그램을 작성하고 싶은 유혹을 느낄 것입니다. 우리의 접근 방식은 그러한 유혹을 뿌리칠 수 있도록, C++ 표준 라이브러리의 하이-레벨 추상화를 처음부터 사용합니다. 만약, 여러분이 이미 C나 C++ 경력 프로그래머라면, 이러한 접근 방식은 좋은 뉴스인 동시에 다소 나쁜 뉴스일 수 있습니다. 실은, 같은 내용이긴 합니다.

그 뉴스란, 우리가 설명하는 C++을 이해하는 데에 있어, 여러분의 지식은 별로 도움이 되지 않는다는 사실에 놀라게 된다는 것입니다. 기대했던 것보다 많은 것을 배워야 할 것이지만(나쁜 뉴스), 기대했던 것보다 빨리 배울 수 있을 것입니다(좋은 뉴스). 특히, 이미 C++를 알고 있다면, 아마 C를 먼저 배웠을 것입니다. 이 말은 여러분의 C++ 프로그래밍 스타일은 C에 기초하고 있다는 것을 의미합니다. 그러한 접근 방식에도 문제는 없습니다. 하지만 우리의 접근 방식은 좀 달라서, 여러분이 지금껏 보지 못했던 C++의 면모를 보게될 것입니다.

물론, 많은 문법적 세부 설명에도 익숙해지겠지만, 그것은 말 그대로 세부 설명일 뿐입니다. 중요한 개념들은 지금껏 알고 있던 것과는 다른 순서로 진행할 것입니다. 예를 들어, 포인터나 배열에 대해서는 제 10장까지 언급하지 않습니다. 그리고, 예전에 주로 사용하던 printf나 malloc과 같은 것은 전혀 논의하지 않습니다. 대신, 제 1장부터 표준 라이브러리인 string 클래스를 시작할 것입니다. 새로운 접근법을 적용했다는 것이 바로 이것입니다!

이 책의 구조

이 책은 두 부분으로 나뉘어 있다고 생각하면 편할 것입니다. 그 첫 번째 부분은 제 7장까지의 내용으로, 표준 라이브러리 추상화를 사용하는 프로그램을 집중적으로 다루고 있습니다. 두 번째 부분은 제 8장부터의 내용으로, 여러분 자신만의 추상화를 정의하는 방법을 다루고 있습니다.

라이브러리를 먼저 설명한다는 것은 일반적인 방법은 아니지만, 우리는 그 방법이 옳다고 생각합니다. C++ 언어의 많은 부분, 특히 그 중에서 어려운 부분들은, 라이브러리 작성자들을 위한 것입니다. 라이브러리 사용자가 언어의 그러한 부분들까지 알 필요는 전혀 없습니다. 책의 두 번째

파트까지 언어의 어려운 부분들은 무시하기 때문에, 일반적인 접근 방식을 적용하는 것보다 훨씬 더 빠르게 유용한 C++ 프로그램을 작성할 수 있습니다.

일단, 라이브러리를 사용하는 법을 이해했다면, 라이브러리의 저수준 기능들과 이러한 기능들을 사용한 자신만의 라이브러리 작성법에 대해서도 공부할 준비가 된 것입니다. 뿐만 아니라, 어떻게 하면 라이브러리를 유용하게 만들 수 있을지, 그리고 새로운 라이브러리 코드를 전혀 작성하지 말아야 할 때가 언제인지에 대한 영감을 받을 수 있을 것입니다.

비록 이 책은 대부분의 C++ 책보다 얇긴 하지만, 모든 중요한 개념들을 최소한 두 번씩은 사용하려고 했으며, 핵심 개념들은 그 이상 사용하려고 노력했습니다. 그 결과, 책의 많은 부분은 다른 부분을 참조합니다. 참조 시에는 해당 절의 번호를 언급할 것입니다. 각 개념을 처음 소개할 때에는, 그 부분을 쉽게 찾고 주의 깊게 살펴보라는 뜻에서 볼드 이탤릭체를 사용할 것입니다.

모든 장(마지막 장은 제외)은 '내용 정리'라는 절을 가집니다. 이 절은 두 가지 목적을 갖습니다. 해당 장에서 소개한 개념들을 쉽게 기억하라는 의미를 가지기도 하며, 나중에는 필요하게 될 것 같은 추가적인 내용이나 관련 내용들을 다룹니다. 처음 읽을 때에는 이 절은 건너 뛰고, 나중에 필요할 때에 다시 살펴보는 방법을 권장합니다.

부록 A, B는 프로그램을 작성할 때 유용할 만큼 세세하게, 언어 및 라이브러리의 중요한 파트들을 요약하고 설명합니다.

이 책의 최신판 얻기

프로그래밍 언어에 관한 모든 책들은 예제 프로그램을 포함합니다. 이 책도 예외는 아닙니다. 이러한 프로그램들이 어떻게 동작되는지 알고 싶다면, 컴퓨터 상에서 실행해 보는 수밖에는 없습니다. 이 프로그램을 실행해 볼 수 있는 컴퓨터는 널려 있습니다. 하지만, 지금도 새 컴퓨터가 출시되고 있습니다. 즉, 여러분이 이 글자를 읽는 순간에도 컴퓨터는 새로 출시되고 있기 때문에, 어떤 컴퓨터에서 이 프로그램을 실행시킬 수 있다라는 말은 부정확한 표현이 될 수 있습니다. 따라서, C++ 프로그램을 어떻게 컴파일하고 실행하는지 모른다면, http://www.acceleratedcpp.com 에 방문하여 어떤 내용이 있는지 구경해 보세요. C++ 프로그램을 실행하는 메커니즘에 대한 정보 및 권고를 틈날 때마다 올려놓을 예정입니다. 이 사이트에는 일부 예제 프로그램의 실행파일 및 관심 가질만한 다른 정보들도 제공합니다.

감사의 글

이 책의 출간에 도움을 주신 분들께 감사드립니다. 이분들이 없었더라면, 이 책의 출간이 불가능했을지도 모릅니다. 이 책의 형식은 리뷰해 주신 분들의 영향을 많이 받았습니다. 그 분들은 바로 Robert Berger, Dag Bruck, Adam Buchsbaum, Stephen Clamage, John Kalb, Jeffrey Oldham, David Slayton, Bjarne Stroustrup, Albert Tenbusch, Bruce Tetelman, Clovis Tondo 씨입니다. Addison-Wesley의 많은 분들이 출간에 참여해 주셨는데, 그 중에서 이름을 알고 있는 분들은 Tyrrell Albaugh, Bunny Ames, Mike Hendrickson, Deborah Lafferty, Cathy Ohala, Simone Payment 씨입니다. Alexander Tsiris 씨는 13.2.2 절의 그리스어 어원을 확인해 주셨습니다. 마지막으로, 강좌를 들은 수 백 명의 학생들과 연설에 참가한 수 천명의 참석자들에 영향력에 힘입어, 고수준 프로그램으로 시작한다는 아이디어가 수 년 동안 널리 퍼질 수 있게 되었습니다.

Chapter 0

시작 전 몸풀기

이 장에서는...

먼저, 작은 C++ 프로그램 하나를 살펴보면서 시작해 보기로 하겠습니다.

```
// 작은 프로그램
#include <iostream>

int main()
{
        std::cout << "Hello, world!" << std::endl;
        return 0;
}
```

프로그래머들은 보통 이런 프로그램을 헬로우 월드(Hello, world!) 프로그램이라고 부릅니다. 비록 작은 프로그램이기는 하지만, 이 책을 계속 진행하기에 앞서 반드시 자신의 컴퓨터에서 컴파일하고 실행해 보도록 합시다. 이 프로그램은 분명

```
Hello, world!
```

라는 결과를 표준 출력(standard output)에 표시할 것입니다. 여기에서, 표준 출력은 일반적으로 디스플레이 스크린 상의 윈도우를 말합니다. 만약 이렇게 해보다가 문제에 부딪히게 된다면, 이미 C++를 알고 있는 사람을 찾아 도움을 구하거나, 우리의 웹 사이트, http://www.acceleratedcpp.com에 조언을 요청해 보기 바랍니다.

이 프로그램은 아주 유용한 프로그램입니다. 왜냐하면, 너무나 단순해서 만약 문제가 발생한다고 해도, 고작 타이핑 실수, 아니면 구현시스템(implementation)[1] 사용법에 대한 오해 외에는 달리 문제될 것이 없기 때문입니다. 더욱이, 비록 작은 프로그램이긴 하지만 이 프로그램을 철저히 이해한다면, C++의 기초에 대하여 상당히 많은 것들을 배울 수도 있습니다. 자, 그렇다면 이제 이 프로그램을 한 줄씩 자세하게 살펴보면서 철저히 이해해 보도록 합시다.

0.1 주석

우리가 살펴볼 프로그램의 첫 줄은 다음과 같습니다.

```
// 작은 C++ 프로그램
```

이와 같이 // 문자부터 그 줄의 끝까지를 *주석(comment)*이라고 합니다. 컴파일러는 이 주석들을 무시해 버립니다. 말하자면, 주석은 단지 어떤 프로그램을 읽는 사람들에게 그 프로그램에 대해 설명하는 목적으로만 사용될 뿐입니다. 이 책에서는 주석을 *이탤릭체*로 표시하여 프로그램의 다른 부분들과 주석을 쉽게 구별할 수 있도록 하겠습니다.

0.2 #include

C++에서, 입·출력과 같은 수많은 기본 기능들은, ***기본 언어**(Core language)*가 아니라, ***표준 라이브러리**(standard library)*에 속합니다. 이 둘은 꼭 구별되어야 합니다. 왜냐하면, 기본 언어

1) 이 책에서는 C++ 언어를 구현한 컴파일러 및 링커를 포함한 패키지 시스템을 일컫는 용어로 사용됨. 예를 들면, Microsoft사의 Visual C/C++ 및 볼랜드 사의 C++빌더, GNU GCC 등이 이에 해당된다.

는 모든 C++ 프로그램에서 언제든지 사용할 수 있지만, 표준 라이브러리는 사용하기 위해서 반드시 사용할 부분을 미리 요청해 두어야 하기 때문입니다.

프로그램이 표준 라이브러리의 기능을 사용할 때에는 **#include** ***지시문****(directives)*을 사용합니다. 이러한 지시문은 보통 프로그램의 처음에 나타납니다. 우리의 헬로우 월드 프로그램에서 유일하게 사용되는 표준 라이브러리는 입·출력 라이브러리이며, 다음과 같이 요청할 수 있습니다.

```
#include <iostream>
```

이 iostream이라는 이름은 글자 그대로 임의 접근 입·출력이나 그래픽 입·출력이 아니라, 순차적 입·출력 또는 스트림 입·출력에 대한 기능을 제공하는 라이브러리를 말합니다. 또한, 이 iostream이라는 이름은 #include 지시문에 나타나며, ***각괄호****(angled bracket,* "<"와 ">")로 닫혀 있기 때문에, 표준 헤더(standard header)라는 C++ 라이브러리라고 할 수도 있습니다.

C++ 표준 헤더가 정확히 어떤 것인지를 알려 주지는 않지만, 각각의 헤더 이름과 작동 방식을 정의해 주고 있습니다. 표준 헤더를 인클루드(include)하면 해당 라이브러리의 기능을 프로그램에서 사용할 수 있게 됩니다. 그러나 구현시스템이 정확하게 이를 어떻게 처리하는지는 우리가 알지 못해도 상관 없습니다.

0.3 main 함수

함수*(function)*란 자신의 이름을 가지며, 다른 곳에서 이를 ***호출****(call)*하거나 실행시킬 수 있는, 프로그램의 한 조각을 말합니다. 모든 C++ 프로그램은 반드시 main이라는 이름의 함수를 갖고 있어야 합니다. 우리가 C++ 구현시스템에게 프로그램의 실행을 요청하면, 구현시스템은 이 함수를 호출하여 그 요청을 수행하는 것입니다.

main 함수는 그 결과로 하나의 정수(integer) 값을 되돌려야 하는데, 그 목적은 구현시스템에게 프로그램이 성공적으로 실행되었는지 여부를 알리는 데 있습니다. 영(0) 값은 성공을 의미하며, 다른 값은 문제가 발생했음을 의미합니다. 따라서, 우리는 int 타입의 값을 리턴하는 main이라는 이름의 함수를 정의하기 위하여

```
int main()
```

라고 작성합니다. 여기에서, int란 기본 언어(core language)가 정수를 표현하기 위하여 사용하는 이름입니다. main 다음의 괄호는 우리의 함수가 구현시스템으로부터 받는 매개변수들(parameters)을

담습니다. 이 예제는 좀 특수한 경우로, 매개변수가 전혀 없으며, 따라서 괄호 사이에 아무 것도 적지 않습니다. main의 매개변수 사용법은 10.4 절에서 살펴보겠습니다.

0.4 중괄호

괄호 다음에 *중괄호(curly braces)*로 묶인 일련의 *구문(statements, 또는 문)*을 삽입하여 main 함수의 정의를 계속해 보기로 합시다.

```
int main()
{       // 왼쪽 중괄호
        // 여기에 구문(statement)들을 적습니다.
}       // 오른쪽 중괄호
```

C++에서, 중괄호는 구현시스템에게 그 안에 무엇이 오든지 하나의 단위로 처리하라고 말합니다. 이 예제에서 보면, 왼쪽 중괄호는 main 함수 안에서 구문들의 시작을 표시하고, 오른쪽 중괄호는 구문들의 끝을 표시합니다. 달리 말하자면, 중괄호는 그 사이에 있는 구문들이 모두 같은 함수의 부분임을 알려준다고 하겠습니다.

이 함수에서처럼, 중괄호 안에 두 개 이상의 구문들이 존재하면, 구현시스템은 이들을 나타난 순서대로 처리합니다.

0.5 표준 라이브러리를 사용하여 출력하기

중괄호 안에 있는 첫 번째 구문이 우리의 프로그램이 실질적인 작업을 하는 부분입니다.

```
std::cout << "Hello, world!" << std::endl;
```

이 구문은 표준 라이브러리의 *출력 연산자(output operator)*, <<를 사용하여 표준 출력에 Hello, world!를 출력하고, 그 다음에 std::endl 값을 출력합니다.

이름 앞에 std::가 붙으면, 그 이름은 std라고 불리는 *네임스페이스(namespace)*의 일부라는 뜻이 됩니다. 여기서 네임스페이스란, 서로 관계가 있는 이름들의 집합을 말합니다. 그리고 표준 라이브러리는 std 안에 자신이 정의하고 있는 모든 이름들을 담습니다. 예를 들면, iostream 표준 헤더에는 cout과 endl이라는 이름들이 정의되어 있고, 이 이름들을 각각 std::cout과 std:endl의 형태로 사용할 수 있습니다.

std::cout 이라는 이름은 *표준 출력 스트림(standard output stream)*을 나타내며, 어떠한 C++ 구현시스템이라도 이 이름은 프로그램으로부터의 일반적인 출력을 의미합니다. 윈도우 기반 운영체제 하의 전형적인 C++ 구현시스템에서, std::cout은 구현시스템이 프로그램에 연결시킨 해당 윈도우를 가리킵니다. 그러한 경우, std::cout에 대한 출력은 연결된 윈도우에 나타날 것입니다.

std::endl 이라는 값을 출력하면 현재 라인의 출력을 끝내게 됩니다. 따라서, 만약 이 프로그램이 조금 더 출력하는 프로그램이었다면, 그 뒤의 내용들은 그 다음 라인에 출력될 것입니다.

0.6 return 문

다음과 같은 return 문은,

```
return 0;
```

그 구문이 포함된 해당 함수의 실행을 종료하고, return과 세미콜론(;) 기호 사이에 있는 값(이 예제에서는 0)을 이 함수를 호출한 프로그램에 리턴합니다. 리턴되는 값은 그 함수가 리턴할 것이라고 명시한 바로 그 타입이어야 합니다. main의 경우에 리턴 타입은 int 타입이며, main이 리턴되는 곳은 C++ 구현시스템 그 자체입니다. 따라서, main의 return 문은 정수 값의 표현식을 포함해야 하며, 이 값은 다시 구현시스템에게 보내집니다.

물론, 프로그램을 종료시킬 수 있는 곳은 한 군데 이상일 수도 있습니다. 그러한 프로그램은 하나 이상의 return 문을 가질 것입니다. 함수 정의 시 그 함수가 어떤 타입의 값을 리턴한다고 하면, 함수 내의 모든 return 문들은 그 타입의 값을 리턴해야만 합니다.

0.7 살짝 깊이 들여다 보기

이 프로그램은 C++에 널리 사용되는 두 가지 추가적인 개념을 사용합니다. 그것은 표현식(expression)과 스코프(scope)입니다. 이 책에서는 앞으로 이러한 개념에 대해서 자주 설명할 것이지만, 여기서 어느 정도 기본적인 내용은 짚고 넘어갈 필요가 있습니다.

*표현식*은 구현시스템에게 어떤 것을 계산하도록 요청합니다. 계산을 하면 결과(result)가 나오고, *부수 효과(side effect)*도 발생합니다. 여기서 부수 효과란, 결과와는 직접적으로 상관없이 구현시스템이나 프로그램의 상태에 영향을 주는 것을 말합니다. 예를 들어, 3+4는 7이라는 결과를 내며 아무런 부수 효과도 없지만,

```
std::cout << "Hello, world!" << std::endl
```

앞의 표현식은 표준 출력 스트림에 Hello, world!를 출력하고 현재 라인을 끝마치는 부수 효과를 갖습니다.

표현식은 연산자(operator)와 피연산자(operand)를 포함하는데, 이 둘은 모두 여러 가지 형태를 취할 수 있습니다. 우리의 Hello, world! 표현식에서, 두 개의 << 기호들은 연산자이며 std::cout, "Hello, world!"와 std::endl 은 피연산자입니다.

모든 피연산자는 *타입(type)*을 갖습니다. 타입에 대해서도 할말이 많지만, 기본적인 것만 이야기한다면, 타입은 자료구조와 그 자료구조에 대한 연산(operation)의 의미를 나타내는 것입니다. 연산자가 어떠한 영향을 미칠지는 그 피연산자들의 타입에 의존합니다.

타입은 보통 이름을 가집니다. 예를 들어, 기본 언어는 int 를 정수를 표현하는 타입의 이름으로 정의하며, 라이브러리는 std::ostream을 스트림 기반의 출력을 제공하는 타입으로 정의합니다. 우리의 프로그램에서, std::cout 은 std::ostream 타입을 가집니다.

<< 연산자는 두 개의 피연산자들을 가집니다. 하지만, 여기에서는 두 개의 << 연산자와 세 개의 피연산자를 사용하였습니다. 어떻게 그럴 수 있을까요? 그 답은 <<이 *왼쪽 우선 결합(left associative)*이기 때문입니다. 그 말을 쉽게 다시 설명하자면, <<가 같은 표현식에서 두 번 이상 나오면, 각각의 <<는 가능한 많은 표현식을 왼쪽 피연산자로 사용하고, 가능한 한 적은 표현식을 오른쪽 피연산자로 사용한다는 것입니다. 예제에서, 첫 번째 << 연산자는 오른쪽 피연산자로 "Hello, world!"를, 왼쪽 연산자로 std::cout 을 가집니다. 두 번째 << 연산자는 오른쪽 피연산자로 std::endl 을, 왼쪽 연산자로 std::cout << "Hello, world!"를 가집니다. 만약 연산자와 피연산자와의 관계를 명확히 하기 위하여 괄호로 묶어 주면, 출력 표현식은 다음과 같을 것입니다.

```
(std::cout << "Hello, world!") << std::endl
```

각각의 <<는 해당 피연산자들의 타입에 따라 동작합니다. 첫 번째 <<는 왼쪽 피연산자로 std::cout 을 가집니다. 이 때, std::cout 은 std::ostream 타입입니다. 오른쪽 피연산자는 문자열 리터럴(literal)입니다. 문자열 리터럴에 대해서는 10.2 절까지는 언급하지 않을 것입니다. 이러한 피연산자 타입들을 사용하여, <<는 오른쪽 연산자의 문자들을 왼쪽 연산자가 나타내는 스트림에 기록합니다. 그러면, 그 결과는 왼쪽 피연산자가 됩니다.

두 번째 <<의 왼쪽 피연산자는 std::ostream 타입을 갖는 std::cout 의 표현식이 됩니다. 오른쪽 피연산자는 std::endl 로서, *조작자(manipulator)*라고 부릅니다. 스트림에 조작자를 쓴

다는 것은 단순히 문자를 쓰는 것과는 약간 다른데, 이를 통해 스트림을 조작할 수 있다는 것이 조작자의 주요 속성이라 할 수 있습니다. <<의 왼쪽 피연산자가 std::ostream 타입이며, 오른쪽 피연산자가 조작자일 때, <<는 주어진 스트림에 대해 조작자가 의미하는 어떤 작업을 수행하고, 그 결과를 스트림에 되돌립니다. std::endl의 경우에는, 현재 출력 라인을 끝마치는 작업을 수행합니다.

그러므로, 전체 표현식은 그 결과로 std::cout을 돌려주며, 부수 효과로 Hello, world!를 표준 출력 스트림에 쓴 다음, 출력 라인을 종료시킵니다. 표현식 다음에 세미콜론을 사용하면, 구현시스템에게 그 값을 무시하라고 요청하게 됩니다. 그 값을 무시해도 되는 이유는, 우리가 원하는 것은 그 부수 효과뿐이기 때문입니다.

어떤 이름에 대한 ***스코프***(*scope*)란 그 이름이 프로그램 상에서 의미를 갖는 범위를 말합니다. C++는 여러 종류의 스코프들을 갖는데, 여러분들은 그 중의 둘은 이미 이 프로그램에서 보았습니다.

우리가 사용한 첫 번째 스코프는 네임스페이스입니다. 이미 말했듯이, 네임스페이스란 관련된 이름들의 집합을 의미합니다. 표준 라이브러리는 모든 이름을 std라는 네임스페이스에 정의합니다. 따라서, 여러분이 직접 작성한 것들과의 충돌을 피할 수 있습니다. 단, 실수로 std라는 이름을 정의하지만 않는다면 말입니다. 표준 라이브러리에 있는 이름을 사용하려면, 그 이름이 라이브러리로부터 온 것이라는 것을 명시해야 합니다. 예를 들어, std::cout으로 cout이 std라는 이름의 네임스페이스에 정의되어 있다는 것을 알 수 있습니다.

std::cout이라는 이름은 ***한정된 이름***(*qualified name*)으로서, :: 연산자를 사용합니다. 이 연산자는 ***스코프 연산자***(*scope operator*)라고도 합니다. ::의 왼쪽에는 스코프의 이름(아마 한정된)이 오게 됩니다. std::cout의 경우에는 std라는 네임스페이스 이름이 옵니다. ::의 오른쪽에는 왼쪽에 적힌 스코프에 정의되어 있는 이름이 옵니다. 따라서, std::cout이란 "cout이라는 이름이 std라는 스코프(네임스페이스)에 있다"라는 것을 의미합니다.

중괄호는 이와는 다른 종류의 스코프를 나타냅니다. main의 본체(그리고 모든 함수의 본체)는 그 자체가 스코프입니다. 이것은 이처럼 작은 프로그램에 있어서는 큰 의미가 없지만, 우리가 작성할 다른 거의 모든 함수에 있어서는 중요합니다.

0.8 내용 정리

비록 작성한 프로그램은 작지만, 이번 장에서 많은 영역을 다루었습니다. 앞으로 여기에서 소개한 내용들을 기반으로 설명할 것이므로, 이 장을 확실히 이해하고 다음 장으로 진행하기 바랍니다.

이해를 돕고자, 이번 장(및 16장을 제외한 다른 모든 장)은 '내용 정리'라고 하는 절과 연습문제들로 끝마칠 것입니다. *내용 정리* 절은 문서의 내용을 요약하거나, 때로는 부연 설명을 하게 됩니다. 각 장에서 설명한 아이디어를 다시 되살펴보기 위하여 각 *내용 정리* 절을 살펴보는 것도 바람직하겠습니다.

■ ***프로그램 구조*** : C++ 프로그램들은 보통 ***자유형식****(free form)*을 가집니다. 그 말은 스페이스(space)는 서로 인접한 기호들을 분리하기 위해서 사용될 뿐이라는 것입니다. 특히, 개행문자(즉, 구현시스템이 프로그램의 한 라인과 다음 라인을 구별하기 위해 사용하는)는 단지 스페이스의 일종일 뿐, 다른 특별한 의미는 없습니다. 프로그램에 스페이스를 어떻게 사용하느냐에 따라 프로그램을 더 읽기 쉽게 하거나, 더 읽기 어렵게 만들 수도 있습니다. 일반적으로, 프로그램은 가독성이 좋아야 합니다.

자유형식이 아닌 세 가지 요소들은 다음과 같습니다.

스트링 리터럴	이중 인용부호로 둘러싸인 문자. 여러 라인에 걸치지 않습니다.
`#include` ***이름***	모두 한 라인에 나타내야 합니다(주석은 제외).
`//` ***주석***	`//` 다음에 어떤 내용들이 오고, 현재 라인의 끝에서 종료됩니다.

`/*`로 시작하는 주석은 자유형식입니다. 가장 먼저 나오는 `*/`에서 종료되며, 여러 라인에 걸칠 수 있습니다.

타입은 자료구조 및 해당 자료구조에 대한 연산을 정의합니다. C++는 두 종류의 타입을 가집니다. `int`와 같은 기본 언어 내부에 정의된 것들과 `std::ostream`과 같이 기본 언어 외부에서 정의된 것들이 있습니다.

네임스페이스는 관련된 이름들을 그룹화시키는 도구입니다. 표준 라이브러리의 이름들은 `std`라고 하는 네임스페이스 안에 정의되어 있습니다.

문자열 리터럴은 이중 인용부호(")로 시작하며 이중 인용부호로 끝납니다. 각 문자열 리터럴은 프로그램의 한 라인에 모두 나타내야 합니다. 문자열 리터럴에 사용되는 일부 문자들은 백슬래쉬(\) 다음에 사용하면 특별한 의미를 가집니다.

\n	개행(newline) 문자
\t	탭(tab) 문자
\b	백스페이스(backspace) 문자
\"	문자열 종료를 위한 것이 아니라 문자열의 일부로서 이 기호(") 자체를 의미
\'	문자 리터럴(1.2 절 참조)에서처럼 문자열 리터럴에서 '의 의미
\\	문자열에 \ 를 포함시킴. 그 다음 문자는 일반 문자로 취급

문자열 리터럴에 대한 내용은 10.2 절 및 A.2.1.3 절에서 더 자세히 살펴보도록 하겠습니다.

■ ***정의(definitions) 및 헤더(headers)*** : C++ 프로그램이 사용하는 모든 이름은 그 이름에 대한 정의를 해야 합니다. 표준 라이브러리는 헤더 안에 그 이름을 정의하며, 다른 프로그램들은 #include를 통해 접근할 수 있습니다. 이름은 사용하기 전에 미리 정의되어 있어야 합니다. 따라서, 헤더에 있는 이름을 사용하기 전에 #include를 먼저 사용하는 것입니다. <iostream> 헤더는 라이브러리의 입·출력 기능을 정의합니다.

■ ***main 함수*** : 모든 C++ 프로그램은 int를 리턴하는 main이라는 이름의 함수를 꼭 정의해야 합니다. 구현시스템은 main을 호출함으로써 프로그램을 실행시킵니다. main에서 0을 리턴하면 성공을, 0이 아닌 다른 값을 리턴하면 실패를 의미합니다. 일반적으로, 함수는 적어도 하나의 return 문을 포함해야 하며, 갑자기 함수를 끝내버려서는 안됩니다. 하지만, main 함수는 좀 특별합니다. return을 생략할 수 있습니다. 그렇게 하면, 구현시스템은 0인 리턴 값을 가정합니다. 그러나 main 함수에 명시적으로 리턴문을 포함시키는 것이 바람직합니다.

■ ***괄호 및 세미콜론*** : 눈에 잘 띄지 않는 이러한 기호들은 C++ 프로그램에서 중요합니다. 그것들은 작기 때문에 간과하기 쉽지만, 이러한 기호들을 생략하면, 이해하기 힘든 컴파일러 오류 메시지들을 남발시키기 때문에, 주의 깊게 확인해야 합니다.

괄호 안에 하나의 구문(statement)도 없거나 하나 이상의 구문을 나열한 것도 구문이며, ***블록(block)***이라고 부릅니다. 블록 안의 구문은 그 순서대로 수행됩니다. 함수의 본체는 단 하나의 구문만 존재한다고 하더라도 괄호로 둘러싸여야 합니다. 쌍으로 대응하는 괄호 안에 있는 구문들은 하나의 스코프를 형성합니다.

표현식 다음에 세미콜론이 있으면 ***표현식 구문(expression statement)***이라고 부릅니다. 이때, 그 결과는 무시되고 부수 효과만 수행됩니다. 표현식은 선택적이기 때문에, 생략하면 아무런 일도 일어나지 않는 ***빈 구문(null statement)***이 됩니다.

■ **출력(*output*)** : std::cout << e를 실행하면, e의 값을 표준-출력 스트림에 씁니다. 그리고 std::cout을 수행합니다. std::cout은 ostream의 타입을 가지며, 그 값은 꼬리에 꼬리를 무는 출력에 사용될 수 있습니다.

연습문제

0-0. Hello, world! 프로그램을 컴파일하고 실행해 보세요.

0-1. 다음 문장의 의미는 무엇인가요?

```
3 + 4;
```

0-2. 실행하면 다음과 같은 출력을 하는 프로그램을 작성해 보세요.

```
This (") is a quote, and this (\) is a backslash.
```

0-3. 문자열 리터럴 "\t"는 탭 문자를 나타내는데, C++ 구현시스템마다 다른 방식으로 탭을 표현합니다. 여러분의 구현시스템이 어떻게 탭을 표시하는지 시험해 보세요.

0-4. 실행하면 그 출력으로 Hello, world! 프로그램 소스 코드를 출력하는 프로그램을 작성해 보세요.

0-5. 이 프로그램이 올바른가요? 왜 그런가요? 혹은, 왜 그렇지 않은가요?

```
#include <iostream>
int main() std::cout << "Hello, world!" << std::endl;
```

0-6. 이 프로그램이 올바른가요? 왜 그런가요? 혹은, 왜 그렇지 않은가요?

```
#include <iostream>
int main() {{{{{{ std::cout << "Hello, world!" << std::endl; }}}}}}
```

0-7. 이 프로그램은 어떤가요?

```
#include <iostream>
int main()
{
        /* 이것은 여러 라인에 걸친 주석입니다.
        왜냐하면, /*과 */를 시작 및 종료 표시로 사용하기 때문입니다.
        */
std::cout << "Does this work?" << std::endl;
return 0;
}
```

0-8. **그렇다면, 이 프로그램은 어떤가요?**

```
#include <iostream>
int main()
{
        // 이것은 /*나 */를 사용하는 대신,
        // 각 라인의 시작부분에 //를 사용하여
        // 여러 라인에 걸치게 한 주석입니다.
        std::cout << "Does this work?" << std::endl;
        return 0;
}
```

0-9. **온전한 프로그램으로서 가장 짧은 프로그램을 작성해 보세요.**

0-10. **공백문자(whitespace)가 들어갈 수 있는 모든 부분에 개행문자를 넣어** Hello, world!**를 다시 작성해 보세요.**

Chapter 1

문자열 처리하기

이 장에서는...

0장에서는 아주 작은 프로그램을 자세히 살펴봄으로써, 여러 가지 기본적인 C++ 개념들을 맛보았습니다. 주석, 표준 헤더, 스코프, 네임스페이스, 표현식, 구문, 문자열 리터럴, 그리고 출력 등에 대해서 살펴보았죠. 이번 장에서는 문자열을 사용하는 간단한 프로그램을 작성함으로써, 기본적인 내용들을 계속해서 살펴볼 것입니다. 앞으로 C++ string 라이브러리와 입력에 대한 것뿐만 아니라, 선언(declaration), 변수(variable), 그리고 초기화에 대한 내용을 살펴보겠습니다. 이 장의 프로그램은 너무나 간단해서 2장에서 다룰 내용인 제어 구조(control structure)도 필요 없습니다.

1.1 입력

텍스트를 화면에 쓸 수 있게 되었다면, 그 다음 단계는 텍스트를 읽어들이는 것입니다. 예를 들어, Hello, world! 프로그램을 변형시켜 특정 사람에게 Hello라고 말하는 프로그램을 작성할 수 있습니다.

```
// 사람의 이름을 묻고, 그 사람에게 인사합니다.
#include <iostream>
#include <string>
int main()
{
    // 이름을 물어봅니다.
    std::cout << "Please enter your first name: ";
    // 이름을 읽음
    std::string name; // name을 정의
    std::cin >> name; // name에 읽어들입니다.
    // 인사말을 출력합니다.
    std::cout << "Hello, " << name << "!" << std::endl;
    return 0;
}
```

이 프로그램을 실행시키면, 다음과 같은 글을 표준 출력에 표시할 것입니다.

```
Please enter your first name:
```

예를 들어, 다음과 같이 응답하면,

```
Vladimir
```

프로그램은 다음과 같이 출력할 것입니다.

```
Hello, Vladimir!
```

어떤 일이 일어나는지 살펴봅시다. 입력된 내용을 읽어들이기 위하여, 그것을 저장할 공간을 마련해야 합니다. 그러한 공간을 *변수(variable)*라고 부릅니다. 변수는 이름을 갖는 *객체(object)*입니다. 객체는 타입을 갖는 컴퓨터 메모리의 일부를 말합니다. 객체와 변수의 차이는 중요합니다. 그 이유에 대해서는 3.2.2 절, 4.2.3 절, 그리고 10.6.1 절에서 다루도록 하겠습니다. 참고로, 이름을 갖지 않는 객체도 가능합니다.

변수를 사용하고자 한다면, 구현시스템에게 사용하려고 하는 이름과 타입을 알려주어야 합니다.

이름과 타입을 같이 명시하면 구현시스템이 더 효과적으로 머신 코드(machine code)를 생성할 수 있게 됩니다. 또한, 그렇게 함으로써 구현시스템이 잘못 타이핑된 변수명을 자동적으로 감지할 수도 있습니다. 만약, 잘못 타이핑된 변수명이 우연히도 프로그램에 사용된 다른 변수명과 일치하지만 않는다면 말입니다.

이 예제에서, 변수의 이름은 name이며, 그 타입은 std::string입니다. 0.5 절과 0.7 절에서 보았듯이, std::를 string이라는 이름 앞에 사용하면 그것은 기본 언어의 일부도 아닌, 비표준 라이브러리도 아닌, 즉 표준 라이브러리의 일부라는 것을 의미합니다. 표준 라이브러리의 모든 부분과 마찬가지로, std::string도 관련된 헤더를 가지고 있습니다. 그 이름은 <string>입니다. 따라서, 적절한 #include 지시문을 프로그램에 추가해야 합니다.

다음과 같은 첫 번째 구문은

```
std::cout << "Please enter your first name: ";
```

이제 익숙해져 있어야 합니다. 이 구문은 사용자의 이름을 묻는 메시지를 출력합니다. 이 구문의 중요한 점은 여기에 없는 내용, 즉 std::endl 조작자에 관한 것입니다. 우리는 std::endl을 사용하지 않았기 때문에, 프로그램이 메시지를 출력한 다음 새 라인을 시작하지 않습니다. 대신, 프롬프트를 표시하자마자 컴퓨터는 같은 라인에서 입력을 기다리게 됩니다.

다음 구문은

```
std::string name; // name을 정의
```

로서 ***정의***(*definition*)에 해당합니다. 이것은 name이라는 변수를 정의하는데, 그 변수의 타입은 std::string입니다. 이 정의는 함수 본체에 위치하기 때문에, name은 ***지역 변수***(*local variable*)가 됩니다. 이 지역 변수는 괄호 안에 있는 부분이 실행되는 동안만 존재합니다. 컴퓨터가 }에 도달하는 순간 name 변수를 ***소멸시키고***(*destroy*), 변수가 사용했던 메모리를 다른 용도로 사용할 수 있도록 시스템에 반환합니다. 지역 변수가 제한된 생명주기를 갖는 이유는 변수와 다른 객체들 간에 구별하기 위함입니다.

객체의 타입에 묵시적으로 내포되어 있는 것은 ***인터페이스***(*interface*)로서, 해당 타입의 객체에 사용 가능한 연산(operation)들의 집합을 말합니다. name을 string 타입의 변수(이름 있는 객체)로 정의하게 되면, 우리는 string으로 할 수 있는 모든 일들을 name으로 하고 싶다는 뜻을 묵시적으로 내포하게 됩니다.

그러한 연산들에 string을 *초기화*하는 것도 포함됩니다. string 변수를 정의하면 내부적으로 그것을 초기화합니다. 왜냐하면, 표준 라이브러리는 모든 string 객체가 하나의 값을 가진 채로 생성된다고 생각하기 때문입니다. 우리는 string을 생성할 때에 하나의 값을 부여할 수 있다는 사실에 대해서는 조만간 배우게 될 것입니다. 만약 그런 식으로라면, string은 아무 문자도 갖지 않은 채 생성될 수도 있습니다. 우리는 그러한 string을 *빈 문자열* 또는 *널(null)문자열*이라고 부릅니다.

name을 정의했다면, 다음 문장을 실행할 수 있습니다.

```
std::cin >> name; // name에 읽어들입니다
```

이것은 std::cin을 name에 읽어들이는 구문입니다. << 연산자와 std::cout을 출력에 사용한 것과 마찬가지로, 라이브러리는 >> *연산자*와 std::cin을 입력에 사용합니다. 이 예제에서, >>는 표준 입력으로부터 string을 읽어들여 그 내용을 name이라는 이름의 객체에 저장합니다. 라이브러리에 string을 읽어들이라고 요청할 때에, 라이브러리는 공백문자(스페이스, 탭, 백스페이스, 또는 라인끝 문자)를 입력에서 먼저 제외시키고, 다음 *공백*문자 또는 파일의 끝(end-of-file)을 만날 때까지 문자들을 name에 읽어들입니다. 따라서, std::cin >> name을 실행시킨 결과는 표준 입력으로부터 한 낱말을 읽어들이고, 그 낱말을 구성하는 문자들을 name에 저장하는 것입니다.

입력 연산은 또 다른 부수 효과를 가집니다. 사용자의 이름을 요청하는 프롬프트를 컴퓨터의 출력 장치에 나타나게 합니다. 일반적으로, 입·출력 라이브러리는 그 출력을 내부적인 자료구조인 *버퍼(buffer)*에 저장하는데, 그 버퍼라는 것은 출력 연산을 위해 최적화하기 위한 용도로 사용됩니다. 대부분의 시스템은 출력장치에 쓰는 문자들이 많든 적든 간에 많은 시간을 할애하게 됩니다. 각 출력 요청이 있을 때마다 쓰게 된다면 오버헤드가 발생합니다. 그러한 오버헤드를 피하기 위하여, 라이브러리는 버퍼를 사용하여 쓸 문자들을 누적시킨 다음, 꼭 필요한 때에만 버퍼를 *플러쉬(flush)*하여 그 내용들을 출력 장치에 씁니다. 그렇게 하면, 여러 번의 출력 연산들을 한 번의 쓰기로 묶을 수 있습니다.

시스템이 버퍼를 플러쉬하도록 하는 상황은 여러 가지입니다. 첫 번째로, 버퍼가 꽉 차는 경우가 발생하는데, 그때 라이브러리는 자동적으로 버퍼를 플러쉬합니다. 두 번째로, 라이브러리는 입력 스트림으로부터 읽으라는 요청을 받을 수 있는데, 이 경우에 라이브러리는 버퍼가 꽉 차게 될 때까지 기다리지 않고, 그 즉시 출력 버퍼를 플러쉬합니다. 버퍼를 플러쉬하는 세 번째 경우는 명시적으로 플러쉬하도록 요청하는 경우입니다.

이 프로그램이 cout에 입력 요청 메세지를 출력할 때, 그 출력은 표준 출력 스트림에 연관된 버퍼 속으로 들어가게 됩니다. 그 다음, cin으로부터 읽기를 시도합니다. 이러한 읽기 시도는 cout 버퍼를 플러쉬하기 때문에, 사용자는 그 입력 요청 메시지를 보게 될 것입니다.

그 다음 구문은 출력하는 구문으로서, 라이브러리에게 버퍼를 플러쉬하도록 명시적으로 요청합니다. 이 구문은 입력 요청 메시지를 출력한 앞의 구문보다는 약간 더 복잡합니다. 여기서 우리는 문자열 리터럴 "Hello, "를 출력하고, 그 다음에 name이라는 이름의 string 값을, 그리고 마지막으로 std::endl을 출력합니다. std::endl을 쓰면 출력 라인을 끝내고 버퍼를 플러쉬하게 됩니다. 따라서, 시스템은 그 즉시 출력 스트림에 쓰게 되는 것입니다.

버퍼를 적당한 순간에 플러쉬하는 것은 실행 시간이 오래 걸리는 프로그램을 작성할 때 좋은 습관이라고 할 수 있습니다. 그렇지 않으면, 출력한 다음 여러분이 그것을 보게 될 때까지 시스템 버퍼에 너무 오랫동안 머물러 있게 됩니다.

1.2 이름에 테두리 만들기

지금까지, 이 프로그램은 단순히 인사말을 출력하는 것뿐이었습니다. 이제, 그것을 조금 변형시키고 싶습니다. 인사말을 더 보기 좋게 장식하여, 입력과 출력을 다음과 같이 만들고 싶습니다.

```
Please enter your first name: Estragon
**********************
*                    *
* Hello, Estragon!   *
*                    *
**********************
```

이 프로그램은 출력으로 다섯 줄을 출력합니다. 첫 번째 라인은 테두리를 시작합니다. 테두리는 이름의 길이에 인사말("Hello, ")의 길이를 더하고, 각 끝에 공백 및 * 문자만큼을 더한 길이에 해당하는 일련의 * 문자들로 이루어집니다. 그 다음의 라인은 양쪽 끝에 *를 표시하고 나머지를 적당한 수의 공백으로 채웁니다. 세 번째 라인은 *, 공백, 메시지, 공백, 그리고 *입니다. 마지막 두 라인은 각각 두 번째 및 첫 번째 라인과 동일합니다.

여기에 있어서 현명한 전략은 한 번에 한 조각의 출력을 구성하는 것입니다. 먼저, 이름을 읽어 그 이름을 인사말을 구성하는 데에 사용할 것입니다. 그리고, 인사말을 사용하여 출력의 각 라인을 구성할 것입니다. 다음은 그러한 전략을 사용한 프로그램입니다.

```
// 이름을 요청하고 테두리 있는 인사말을 생성합니다.
#include <iostream>
#include <string>
int main()
{
    std::cout << "Please enter your first name: ";
    std::string name;
    std::cin >> name;

    // 출력할 메시지를 구성
    const std::string greeting = "Hello, " + name + "!";

    // 두 번째 및 세 번째 출력 라인을 구성
    const std::string spaces(greeting.size(), ' ');
    const std::string second = "* " + spaces + " *";

    // 첫 번째 및 다섯 번째 출력 라인을 구성
    const std::string first(second.size(), '*');

    // 전부 출력
    std::cout << std::endl;
    std::cout << first << std::endl;
    std::cout << second << std::endl;
    std::cout << "* " << greeting << " *" << std::endl;
    std::cout << second << std::endl;
    std::cout << first << std::endl;
    return 0;
}
```

먼저, 우리의 프로그램은 사용자의 이름을 요청합니다. 그리고 그 이름을 `name`이라는 변수에 저장합니다. 그 다음, 출력할 메시지를 담을 `greeting`이라는 변수를 정의합니다. 그 다음, `spaces`라는 변수를 정의하여 `greeting`에 포함된 문자수만큼의 공백을 담도록 합니다. `spaces` 변수를 사용하여 `second`라는 변수를 정의합니다. 이 `second`라는 변수는 두 번째 출력 라인을 저장할 것입니다. 그 다음, `first`라는 변수에 `second`에 있는 문자수만큼의 `*` 문자를 담도록 합니다. 마지막으로, 한 번에 한 라인씩 출력합니다.

`#include` 지시문과 처음 세 개의 구문은 이제 친숙해져 있어야 합니다. 한편, `greeting`의 정의는 세 가지 새로운 개념을 소개합니다.

첫 번째 개념은, 우리가 변수를 정의할 때 그 값을 부여할 수 있다는 것입니다. 변수의 이름과 세미콜론 사이에, = 기호와 부여하고자 하는 값을 위치시키면 됩니다. 만약 변수 및 값이 다른 타입을 가졌다면, 10.2 절의 string과 문자열 리터럴에서처럼, 구현시스템이 그 변수의 타입에 맞추어 초기 값을 ***변환***시킬 것입니다.

두 번째 새로운 개념은 +를 사용하여 문자열과 문자열 리터럴, 또는 두 개의 문자열(두 개의 문자열 리터럴은 안됨)을 ***결합***시킬 수 있다는 것입니다. 우리는 0장에서 3+4는 7이라는 것을 언급한 바가 있습니다. 이 예제에서는 +가 완전히 다른 의미를 가집니다. 각 경우에 있어서, + 연산자가 어떤 일을 해야 하는지는 그 피연산자들의 타입을 살펴보면 알 수 있습니다. 연산자가 서로 다른 타입의 피연산자들에 대하여 다른 의미를 가진다면, 그 연산자는 ***오버로드****(overload)*되었다고 이야기할 수 있습니다.

세 번째 개념은 변수의 정의에 **const**를 사용하는 것입니다. const를 사용하면 변수의 값을 그 생명주기 동안 변화시키지 않을 것을 약속합니다. 엄밀히 말하자면, 이 프로그램은 const를 사용하는 이점이 없습니다. 하지만 어떤 변수의 값이 변하지 않는 사실을 알린다면, 프로그램을 더 이해하기 쉽게 만들 수 있습니다.

만약 변수가 const라고 하면, 우리는 그 변수를 즉시 초기화해야 합니다. 왜냐하면, 나중에 초기화할 기회가 없기 때문입니다. 또한, const 변수를 초기화하기 위해 사용하는 값 자체가 상수일 필요는 없습니다. 이 예제에서, 우리는 값을 name에 읽어들이기 전까지는 greeting의 값을 알 수 없습니다. 또한, 프로그램을 실행시키기 전까지는 값을 읽어들일 수 없습니다. 우리는 값을 읽어들여 name을 변경하기 때문에, name을 const로 정의할 수는 없습니다.

연산자의 변하지 않는 속성은 그 결합방식(associativity)입니다. 0장에서 <<는 왼쪽 결합 우선이기 때문에 std::cout << s << t는 (std::cout << s) << t와 같은 의미라는 것을 배웠습니다. 이와 유사하게, + 연산자(>> 연산자도 마찬가지)도 왼쪽 결합 우선입니다. 따라서, "Hello, "+name+"!"는 "Hello, "와 name을 결합한 결과에 "!"를 결합한 결과입니다. 결국, 변수 name이 Estragon이라는 값을 갖는다면, "Hello, "+name+"!"의 값은 Hello, Estragon!이 됩니다.

이제, 우리가 표시하려는 메시지가 무엇인지를 알 수 있게 되었으며, 그 정보를 greeting이라는 변수에 저장했습니다. 그 다음 단계는 인사말을 둘러싸는 테두리를 생성하는 것입니다. 이를 위해, 세 가지 이상의 개념을 하나의 구문을 통해 소개하겠습니다.

```
const std::string spaces(greeting.size(), ' ');
```

greeting을 정의할 때, 그 값을 초기화하기 위하여 = 기호를 사용했습니다. 여기에서, spaces 다음에 콤마로 분리된 두 개의 표현식을 괄호로 묶었습니다. = 기호를 사용한다면, 그 변수가 가질 값을 명시적으로 표시하는 것입니다. 여기에서처럼 정의에 괄호를 사용하면, 구현시스템에게 변수의 타입에 의존하여 표현식으로부터 변수를 *구성*하라고 지시하는 것입니다. 이 경우에는 spaces 변수를 구성합니다. 다시 말해서, 이 정의를 이해하려면 두 표현식을 사용하여 string을 구성한다는 의미를 먼저 이해해야 합니다.

변수가 어떻게 구성되는지는 전적으로 그 타입에 의존합니다. 이와 같이 특별한 경우에, 우리는 string을 어떠한 것으로부터 구성합니다. 하지만, 그 어떤 무엇에 해당하는 두 표현식 모두 우리가 아직까지 보지 못한 형식입니다. 과연, 그것이 어떤 의미일까요?

첫 번째 표현식은 greeting.size()로서, ***멤버 함수****(member function)*를 호출하는 경우입니다. 사실상, greeting이라는 객체는 size라는 컴포넌트를 갖는데, 이것은 함수로서 값을 얻기 위해 호출할 수 있습니다. greeting이라는 변수는 std::string이라는 타입으로 정의되었기 때문에, greeting.size()를 호출하면 greeting에 들어있는 문자들의 개수를 나타내는 정수를 얻습니다.

두 번째 표현식은 ' '로서 ***문자 리터럴****(character literal)*을 나타냅니다. 문자 리터럴은 문자열 리터럴과 완전히 구별됩니다. 하나의 문자 리터럴은 항상 단일 인용부호로 둘러싸입니다. 문자열 리터럴은 항상 이중 인용부호로 둘러싸입니다. 문자 리터럴의 타입은 기본 내장 타입인 char입니다. 문자열 리터럴의 타입은 더 복잡하며, 10.2 절까지는 설명하지 않을 것입니다. 문자 리터럴은 단일 문자를 나타냅니다. 문자열 리터럴 안에 특별한 의미를 갖는 문자들은 문자 리터럴에서도 같은 특별한 의미를 가집니다. 따라서, 만약 ' 또는 \를 원한다면, \ 기호를 먼저 적어야 합니다. 덧붙여서, '\n', '\t', '\"' 및 기타 등등의 형식은 0장 문자열 리터럴과 동일한 방식으로 동작합니다.

spaces에 대한 나머지 내용을 짚고 넘어가도록 합시다. string을 정수 값과 char 값으로 구성한다면, char 문자들이 정수 값만큼 복사된 결과를 얻을 수 있습니다. 예를 들어, 다음과 같이 정의한다면,

```
std::string stars(10, '*');
```

stars.size()는 10이 되며, stars 자체는 **********를 갖게 될 것입니다.

따라서, spaces는 greeting과 같은 개수의 문자들을 포함하지만, 그 문자들은 모두 빈칸입니다.

second의 의미를 이해하는 데 있어서 특별히 새로운 것은 없습니다. "* "와 공백문자열, 그리고 " *"를 결합하여 메시지 테두리의 두 번째 라인을 만듭니다. first를 이해하는 데에도 특별한 것은 없습니다. second에 있는 문자 개수만큼 * 문자들을 포함하면 됩니다.

프로그램의 나머지 부분은 익숙할 것입니다. 1.1 절에서 한 것과 마찬가지로 문자열들을 출력하는 것이 전부입니다.

1.3 내용 정리

타입

char 구현시스템에 의해 정의되는 일반 문자들을 담는 기본 타입(built-in type)

wchar_t 한글, 한자, 일본어처럼 많은 수의 문자를 담기 위한 "와이드 문자(wide characters)" 용 내장 타입

string 타입은 <string> 헤더에 정의되어 있습니다. string 타입의 객체는 0 또는 그 이상의 연속적인 문자들을 담을 수 있습니다. n은 정수, c는 char, is는 입력 스트림, os는 출력 스트림이라면, string 연산은 다음과 같습니다.

std::string s;

빈 std::string 타입의 변수 s를 정의합니다.

std::string t = s;

std::string 타입의 변수로 t를 정의하며, 최초에 s의 문자들이 복사됩니다. 여기에서, s는 string일 수도 있고, 문자열 리터럴일 수도 있습니다.

std::string z(n, c);

std::string 타입의 변수로 z를 정의하며, 최초에 문자 c를 n 개 복사한 내용을 담습니다. 여기에서 c는 char이어야 하며, string이나 문자열 리터럴이어서는 안 됩니다.

os << s s 안에 있는 문자들을 출력합니다. 이때, os에 의해 지정된 출력 스트림의 어떠한 포매팅(formatting)도 변하지 않습니다. 표현식의 결과는 os가 됩니다.

is >> s 공백문자를 만날 때까지 is에 의해 지정된 스트림으로부터 문자들을 읽어서 버립니다. 그 다음 계속되는 문자들은 is로부터 s에 읽어들입니다. 이 때, s에 값이 있었더라도 덮어씁니다. 다음 문자가 공백문자일 때까지 계속합니다. 결과는 is입니다.

`s + t` 이 표현식의 결과는 `std::string`이며, `s`에 있는 문자들에 대한 복사본에 `t`에 있는 문자들의 복사본을 합한 내용을 담습니다. `s` 또는 `t` 둘 중의 하나는 `char` 타입의 값 또는 문자열 리터럴이 될 수 있습니다. 하지만, 둘 다는 안됩니다.

`s.size()` `s`에 있는 문자들의 개수

변수는 다음 세 가지 방법 중의 하나로 정의할 수 있습니다.

```
std::string hello = "Hello";    // 초기 값을 명시하여 변수를 정의
std::string stars(100, '*');    // 타입 및 주어진 표현식으로 변수를 구성
std::string name;               // 타입에 의존하는 묵시적 초기화를 통해 변수를 정의
```

중괄호 쌍 안에 정의된 변수들은 지역 변수입니다. 지역 변수는 괄호 안의 프로그램이 실행되는 동안에만 존재합니다. 구현시스템이 }에 도달하면, 변수들을 제거하고 사용된 메모리를 시스템에 반환합니다.

변수를 `const`로 정의하는 것은 변수의 값이 그 생명주기 동안 변하지 않는다는 것을 보장합니다. 이러한 변수는 정의와 동시에 초기화되어야 합니다. 나중에 값을 초기화할 수 없기 때문입니다.

■ **입력** : `std::cin >> v`를 실행하면 표준 입력 스트림에서 공백문자들을 읽어서 버리고, 표준 입력으로부터 변수 `v`에 값을 읽어들입니다. 결과로 `std::cin`을 리턴하는데, 그 타입은 `istream`입니다. 따라서, 입력 연산을 계속해서 연결할 수 있습니다.

연습문제

1-0. 이 장에서 소개한 프로그램을 컴파일하고, 실행하고 테스트해 보세요.

1-1. 다음과 같은 정의가 올바른가요? 왜 그런가요? 또는 왜 그렇지 않은가요?

```
const std::string hello = "Hello";
const std::string message = hello + ", world" + "!";
```

1-2. 다음과 같은 정의가 올바른가요? 왜 그런가요? 또는 왜 그렇지 않은가요?

```
const std::string exclam = "!";
const std::string message = "Hello" + ", world" + exclam;
```

1-3. **다음 프로그램이 올바른가요? 만약 그렇다면, 무엇을 하는 프로그램인가요? 그렇지 않다면, 무엇 때문에 그렇지 않은가요?**

```
#include <iostream>
#include <string>
int main()
{
   { const std::string s = "a string";
     std::cout << s << std::endl; }
   { const std::string s = "another string";
     std::cout << s << std::endl; }
   return 0;
}
```

1-4. **다음 프로그램은 어떤가요? 끝에서 세 번째 라인에서** }}**를** };}**로 바꾸면 어떤가요?**

```
#include <iostream>
#include <string>
int main()
{
   { const std::string s = "a string";
     std::cout << s << std::endl;
   { const std::string s = "another string";
     std::cout << s << std::endl; }}
   return 0;
}
```

1-5. **이 프로그램이 올바른가요? 만약 그렇다면, 어떤 프로그램인가요? 만약 그렇지 않다면, 왜 그렇지 않은지 말해보고, 올바르게 다시 써 보도록 합니다.**

```
#include <iostream>
#include <string>
int main()
{
   { std::string s = "a string";
   { std::string x = s + ", really";
     std::cout << s << std::endl; }
     std::cout << x << std::endl;
   }
   return 0;
}
```

1-6. **다음 프로그램은 무엇을 하는 프로그램인가요? 입력을 요청받으면, 두 개의 이름을 적어 보세요(예, Samuel Beckett). 결과를 미리 예측해 본 다음 실행해 보세요.**

```
#include <iostream>
#include <string>
int main()
{
    std::cout << "What is your name? ";
    std::string name;
    std::cin >> name;
    std::cout << "Hello, " << name
              << std::endl << "And what is yours? ";
    std::cin >> name;
    std::cout << "Hello, " << name
              << "; nice to meet you too!" << std::endl;
    return 0;
}
```