Chapter

01

Hello Objective-C

『아이폰과 맥 OS 개발을 위한 오브젝티브-C 2.0』을 읽는 독자에게 환영의 말을 전한다. 이 책은 오브젝티브-C 언어의 기본을 알려주기 위한 책이다. 오브젝티브-C는 C언어의 수퍼셋(superset)으로 진정한 맥 OS X의 룩앤필(Look & Feel)을 갖는 다양한 애플리케이션에 사용되는 언어이다.

이 책은 오브젝티브-C 언어에 대해 설명하고 그와 관련된 애플의 코코아 툴킷(Cocoa ToolKit)을 소개한다. 코코아는 오브젝티브-C로 만들어졌으며 맥 OS X의 사용자 인터페이스에 대한 모든 요소를 포함하고 있다. 일단 이 책에서 설명하는 오브젝티브-C를 배우고 나면 여러분은 코코아로 진행되는 프로젝트에 빠져들 준비가 된 것이다. 또한 데이브 마크와 제프 라마르쉬가 집필한 『Learn Cocoa on the Mac』 또는 『Beginning iPhone Development』와 같은 책을 볼 준비가 되었다고 할 수 있다.

이 장에서는 이 책을 공부하기 전에 알아야 할 기본 정보를 살펴본 후 오브젝티브-C의 역사에 대해 살짝 알아보고 앞으로 배울 내용에 대해 간단히 살펴보도록 하자.

시작하기 전에

이 책을 읽으려면 C와 같은 프로그래밍 언어(C++, 자바 등)에 약간의 경험이 있어야 좋다. 어떤 언어이든 해당 언어의 기본 원리를 이해해야 한다. 변수와 함수가 무엇인지 알고 제어문과 반복문을 사용해서 프로그램의 흐름을 제어할 수 있는 방법을 이해하고 있어야 하는 것이다. 이 책은

이런 기본 언어를 바탕으로 오브젝티브-C의 특징과 애플의 코코아 툴킷을 추가적으로 이해하는 데 초점이 맞춰져 있다.

C가 아닌 다른 언어를 알고 있는 상태에서도 이 책을 통해 무리 없이 오브젝티브-C를 배울 수는 있겠지만, 이 책의 부록을 살펴보거나 다른 C언어 기초 도서를 읽어보는 것도 도움이 될 것이다.

미래는 어제로 만들어진다

오브젝티브-C와 코코아는 애플의 맥 OS X 운영체제의 핵심이다. 맥 OS X는 비교적 최근에 만들어졌지만 오브젝티브-C와 코코아는 훨씬 오래되었다. 1980년대 초에 브래드 콕스(Brad Cox)는 대중적이고 이식성이 뛰어난 C언어를 우아한 스몰토크(Smalltalk) 언어에 결합시키려는 목적으로 오브젝티브-C를 만들었다. 1985년에 스티브 잡스(Steave Jobs)는 강력하고 가격이 비싸지 않은 워크스테이션을 만들기 위해 NeXT를 설립하였다. NeXT는 운영체제로 유닉스(UNIX)를 선택하였고 오브젝티브-C로 만들어진 강력한 사용자 인터페이스 툴킷을 가진 NextSTEP을 만들게 된다. 그러나 가볍고 멋진 특징을 가지고 있었음에도 불구하고 상업적으로는 성공하지 못했다.

1996년에 애플이 NeXT를 인수했을 때(혹은 그 반대로 NeXT가 애플을 인수했다고 하는 이들도 있음) NextSTEP은 코코아로 이름이 바뀌었고 더 폭넓게 매킨토시 프로그래머를 불러들이게 된다. 애플은 코코아를 포함한 개발 툴을 무료로 배포하였기 때문에 맥 프로그래머들은 이런 개발 툴을 무료로 사용하는 덕을 보게 되었다. 따라서 여러분은 약간의 프로그래밍 경험, 오브젝티브-C의 기본 지식 그리고 배우려는 열정만 있으면 된다.

어쩌면 여러분은 「오래된 유닉스는 둘째치더라도 오브젝티브-C와 코코아는 외계인 알프와 A특공대(외화 시리즈) 시절인 80년대에 개발됐는데 너무 오래되고 진부한 도구는 아닐까?」라고 생각할 수도 있겠다. 그러나 결코 그렇지 않다. 오브젝티브-C와 코코아는 뛰어난 프로그래머로 구성된 팀이 수년간 노력하여 만들어낸 결과물이며 지속적으로 업데이트되면서 발전하고 있다. 시간이 지날수록 오브젝티브-C와 코코아는 놀랍도록 우아하고 강력한 툴로 진화하고 있다. 또한, 오브젝티브-C는 아이폰용 애플리케이션을 개발하기 위한 핵심 도구이기도 하다. 그렇기 때문에 NeXT가 오브젝티브-C를 적용한 후로 20여 년간 모든 훌륭한 개발자들이 오브젝티브-C와 코코아를 사용하고 있는 것이다.

앞으로 배울 것들

오브젝티브-C는 C언어의 수퍼셋이다. 오브젝티브-C는 C에서 시작하였지만 다른 여러 중요한 특징을 C에 추가한 형태이다. C++ 또는 자바를 살펴본 경험이 있다면 오브젝티브-C가 실제로 얼마나 작은지 보고 놀라게 될 것이다. 이 책의 각 장에서는 C에 추가된 오브젝티브-C의 내용에 대해 자세히 알아본다.

- **2장 C의 확장**은 오브젝티브-C를 소개하는 기본 내용에 중점을 두고 있다.
- **3장 객체 지향 프로그래밍의 소개**는 객체 지향 프로그래밍의 기본적인 내용을 설명한다.
- **4장 상속**은 부모 클래스의 특징을 가지고 있는 클래스를 만드는 방법에 대해 설명한다.
- **5장 컴포지션**은 객체들이 결합돼서 함께 동작할 수 있도록 하는 방법을 알아본다.
- **6장 소스 파일 구성**에서는 프로그램 소스를 만들기 위한 실제 전략을 알아본다.
- **7장 Xcode에 대하여**에서는 여러분이 프로그래밍을 할 때 도움을 주기 위해 몇 가지 팁과 Xcode에 능숙해지기 위한 방법을 보여준다.
- **8장 Foundation Kit 소개**는 코코아의 주요 프레임워크 두 개 중 하나를 사용해서 코코아의 멋진 특징을 알아본다.
- **9장 메모리 관리**에서는 코코아 애플리케이션을 다루는 데 중점을 둔다.
- **10장 객체 초기화**에서는 객체가 만들어질 때 어떠한 일이 일어나는지를 살펴본다.
- **11장 프로퍼티**에서는 오브젝티브-C에서 새롭게 사용하는 점(.) 표기법의 비밀을 알아보고 객체 접근자를 쉽게 만드는 방법을 살펴본다.
- **12장 카테고리**에서는 이미 존재하고 있는 클래스(여러분이 작성하지 않은 클래스일지라도)에 새로운 메서드를 추가할 수 있는 오브젝티브-C의 특이한 특징을 설명한다.
- **13장 프로토콜**에서는 클래스를 구현하는데 필요한 내용을 알려주도록 하는 오브젝티브-C에서 상속의 한 형태에 대해 설명한다.
- **14장 AppKit 소개**에서는 또 다른 주요 프레임워크를 사용해서 코코아로 멋진 애플리케이션을 개발할 수 있는 방법을 알아본다.
- **15장 파일 불러오기와 저장하기**에서는 데이터를 저장하고 가져오는 방법을 알아본다.
- **16장 키-밸류 코딩**은 여러분의 데이터를 간접적으로 다루는 방법을 알아본다.
- **17장 NSPredicate**에서는 데이터를 어떻게 자르는지 알아본다.

자바나 C++같은 다른 종류의 언어만 경험해 봤거나 윈도우나 리눅스 같은 다른 플랫폼의 경험만 있다면 **부록**의 **다른 언어에서 오브젝티브-C로**를 살펴보도록 하자. 이 내용으로 오브젝티브-C를 알기 위해 필요한 것들을 정리할 수 있다.

요약

맥 OS X 프로그램은 1980년대부터 사용되어 지금은 강력한 툴로 진화된 오브젝티브-C로 만들어진다. 이 책은 여러분이 C 프로그래밍을 어느 정도 알고 있다는 가정 하에 시작한다. 그럼 즐거운 여행이 되길 바란다!

Chapter

02

C의 확장

오브젝티브-C는 C언어에 약간의 내용이 추가된 것이 전부다. 그래서 멋지다! 이 장에서는 우선 간단한 오브젝티브-C 프로그램을 만들어 보면서 C언어에서 추가된 내용을 알아본다.

가장 간단한 오브젝티브-C 프로그램

여러분은 이미 전통적인 Hello World 프로그램(Hello World! 또는 그 비슷한 간단한 단어를 출력하는 기본 프로그램)의 C버전에 접해 보았을 것이다. Hello World는 보통 초보 C 프로그래머가 배우는 첫 번째 프로그램이다. 우리는 전통을 무시할 생각이 없으므로 여기서도 Hello Objective-C라고 하는 비슷한 프로그램을 작성해 보겠다.

Hello Objective-C 빌드하기

이 책에서는 여러분이 애플의 Xcode 툴을 설치했다고 가정하겠다. 만일 Xcode가 없거나 사용해 본 경험이 없다면 홈페이지에서 제공하는 Objective-C.zip 파일의 Xcode. pdf를 참조하여 Xcode를 얻는 법과 설치하는 법을 배울 수 있다(http://www.bjpublic.co.kr의 **도서자료 ▶ 다운로드**).

여기서는 여러분의 첫 번째 오브젝티브-C 프로젝트를 만들기 위해 Xcode를 사용하는 단계에 대해 설명하므로 이미 Xcode에 익숙하다면 넘어가도 좋다. 시작하기 전에, 이 책의 사이트에서

Objective-C.zip 파일을 내려 받아 푼다. 압축을 풀고 나면 Learn Objective-C Samples라는 폴더가 생긴다. 이 프로젝트는 예제 폴더 02.01 Hello Objective-C에 들어 있다.

이제 프로젝트를 만들기 위해 Xcode를 시작하자. Xcode는 /Developer/Applications에서 찾을 수 있다. 쉽게 사용하기 위해 Dock에 Xcode 아이콘을 넣어놓도록 하겠다.

일단 Xcode가 시작됐으면 File ▶ New Project를 선택한다. Xcode에서 만들 수 있는 다양한 종류의 프로젝트 목록이 나올 것이다. 다른 프로젝트 타입은 상관하지 말고 그림 2-1에 나타낸 대로 윈도우의 왼편에 있는 Command Line Utility를 선택하고 나서 오른편에 있는 Foundation Tool을 선택한다. 그리고 Choose 버튼을 클릭한다.

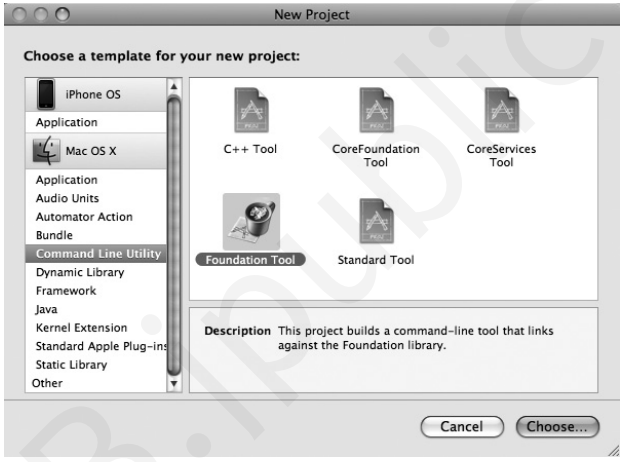


그림 2-1 새로운 Foundation Tool 만들기

Xcode는 시트(sheet)를 보여주고 프로젝트의 이름을 무엇으로 저장할지 물어본다. 원하는 아무 이름이나 넣어도 되지만 우리는 그림 2-2에 보이는 것처럼 「Hello Objective-C」라고 하겠다. 여기서는 이 프로젝트를 깔끔하게 관리하기 위해 Projects 디렉터리에 넣었지만 여러분은 원하는 폴더 아무 곳에나 넣어도 된다.

Save 버튼을 클릭하고나면 Xcode는 프로젝트 윈도우(그림 2-3 참조)라고 하는 메인 윈도우를 보여준다. 이 윈도우는 소스 편집 창을 포함한 프로젝트를 구성하는 여러 부분을 보여준다. 선택되어 있는 Hello Objective-C.m 파일이 Hello Objective-C를 위한 코드를 포함하고 있는 소스 파일이다.

Hello Objective-C.m은 어디서 많이 본 코드를 담고 있는데, Xcode에는 친절하게도 새로운 프로젝트마다 제공하는 기본 코드가 들어 있다. 우리는 Hello Objective-C 프로그램을 Xcode가 제공하는 이 샘플 코드보다 더 알아보기 쉽게 만들 수 있다. Hello Objective-C.m의 모든 코드를 지우고 다음의 코드로 바꾸도록 하자.

```
#import <Foundation/Foundation.h>

int main (int argc, const char *argv[])
{
  NSLog (@"Hello, Objective-C!");

  return (0);

} // main
```

그림 2-2 새로운 Foundation Tool의 이름을 정한다.

지금은 여기에 있는 모든 코드가 이해되지 않는다고 걱정하지는 말자. 곧 이 프로그램을 한 줄씩 자세히 살펴볼 것이다.

소스 코드는 실제로 프로그램을 실행해서 확인할 수 없다면 재미가 없다. Build and Go 버튼을 클릭하거나 ⌘+R을 눌러서 프로그램을 빌드하고 실행하자. 오타가 없다면 Xcode는 이 프로그램을 컴파일하고 링크하고 나서 실행까지 한다. Run ▶ Console을 클릭하거나 ⌘+Shift+R을 눌러서 Xcode의 콘솔 윈도우를 실행해 보자. 콘솔은 그림 2-4 처럼 이 프로그램의 출력을 보여준다.

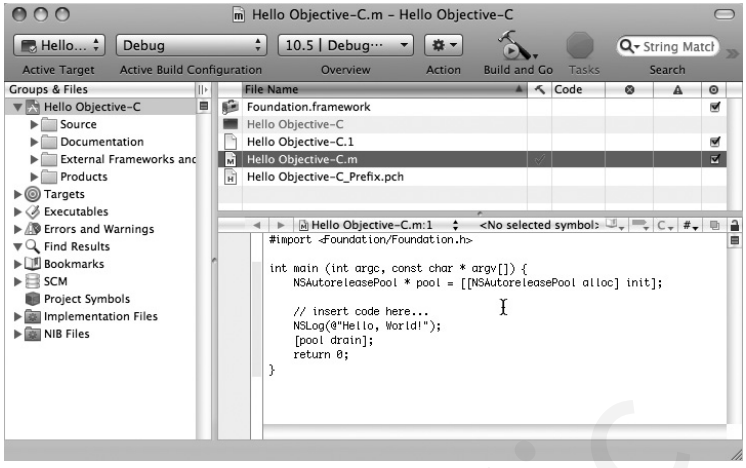


그림 2-3 Xcode의 메인 윈도우

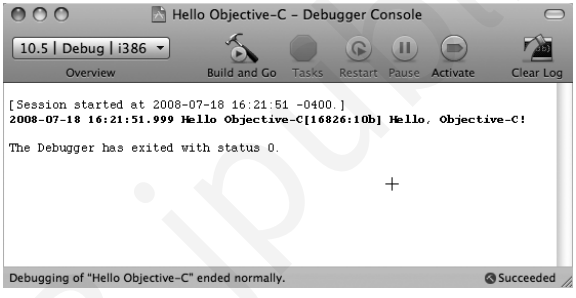


그림 2-4 Hello Objective-C의 실행 모습

드디어 결과를 보게 됐다. 여러분의 첫 번째 오브젝티브-C 프로그램이다. 축하한다! 자 이제 각 부분을 나눠서 어떻게 동작하는지 살펴보자.

Hello Objective-C 분해하기

자, 다시 Hello Objective-C.m의 내용을 보자.

```
#import <Foundation/Foundation.h>

int main (int argc, const char *argv[])
{
  NSLog (@"Hello, Objective-C!");
```

```
    return (0);
}  // main
```

Xcode는 오브젝티브-C 코드가 들어 있는 파일임을 나타낼 때 .m이라는 확장자를 사용한다. 이 파일은 오브젝티브-C 컴파일러에 의해 처리된다. .c로 끝나는 파일이름은 C 컴파일러가, .cpp 파일은 C++ 컴파일러가 처리한다. Xcode에서는 이 모든 컴파일이 GNU Compiler Collection(GCC)에 의해 처리되는데, 하나의 컴파일러로 이 세 가지 다른 언어를 모두 처리할 수 있다.

main.m 파일은 코드가 두 줄 들어있는데 C를 알고 있다면 어렵지 않을 것이다. 앞의 코드를 보면 `main()` 선언으로 시작해서 `return(0)` 구문으로 끝난다. 오브젝티브-C는 그 속이 C와 똑같기 때문에 `main()` 함수를 선언하고 값을 반환하는 문법은 C와 동일하다. 이 부분을 제외한 나머지 코드가 일반적인 C와 약간 달라 보일 것이다. 예를 들어, 저 낯선 `#import`라는 것은 무엇일까? 궁금한가? 계속 읽어보자!

> **NOTE**
> .m 확장자는 원래 오브젝티브-C가 처음 소개됐을 때 메시지(message)를 나타냈는데, 메시지는 앞으로 우리가 공부할 오브젝티브-C의 중요 특징을 나타낸다. 요즘은 메시지라고 읽지 않고 그냥 .m(점엠) 파일이라고 부른다.

낯선 #import

오브젝티브-C도 C처럼 구조체, 상수, 함수 원형과 같은 요소에 대한 선언을 담고 있는 **헤더 파일**(header file)을 사용한다. C에서는 필요한 정의에 대한 헤더 파일을 컴파일러에게 알려주기 위해 `#include` 문을 사용한다. 오브젝티브-C에서도 이런 목적으로 `#include`를 사용할 수 있지만 실제로는 사용하지 않게 될 것이다. 대신 다음과 같이 `#import`를 사용한다.

```
#import <Foundation/Foundation.h>
```

`#import`는 Xcode가 오브젝티브-C나 C 또는 C++ 프로그램을 컴파일할 때 사용하는 GCC 컴파일러가 제공하는 특징이다. `#import`는 그 파일에서 실제로 `#import`가 같은 헤더 파일을 여러 번 포함해도 헤더 파일이 한 번만 포함된다는 것을 보장한다.

NOTE

C에서 프로그래머는 하나의 파일이 두 번째 파일을 포함하고 두 번째 파일이 다시 첫 번째 파일을 포함하는 경우에 재선언을 방지하기 위해서 보통 `#ifdef` 지시자를 사용하도록 구성한다.
오브젝티브-C에서는 프로그래머가 `#import`를 사용함으로써 이런 문제를 해결하고 있다.

`#import <Foundation/Foundation.h>`문은 컴파일러에게 Foundation 프레임워크에서 `Foundation.h` 헤더 파일을 찾아보라고 알려준다.

그럼 프레임워크란 무엇인가? 궁금해 할 줄 알았다. 프레임워크는 여러 부분(헤더 파일, 라이브러리, 이미지, 사운드 등)이 모여서 하나의 단위로 묶여 있는 컬렉션이다. 애플은 코코아, 카본(Carbon), 퀵타임(QuickTime), OpenGL 등의 기술을 프레임워크로 배포한다. 코코아는 Foundation과 Application Kit(AppKit이라고도 한다)의 두 프레임워크로 구성되어 있으며 코코아를 더 멋지게 만들어 주는 코어 애니메이션(Core Animation)과 코어 이미지(Core Image)를 포함하는 프레임워크도 들어있다.

Foundation 프레임워크는 자료구조와 통신 메커니즘 등과 같이 사용자 인터페이스의 하위에 있는 내용을 다룬다. 이 책의 모든 프로그램은 Foundation 프레임워크를 기반으로 한다.

NOTE

이 책을 다 읽은 독자가 코코아의 달인이 되기 위한 다음 과업은 코코아의 상위 레벨 기능인 사용자 인터페이스, 프린팅, 컬러와 사운드 관리, 애플스크립트(AppleScript) 지원 등을 담고 있는 Application Kit에 통달하는 것이다. 더 많이 알고 싶은 사람에게는 곧 번역서로도 출간될 예정인 데이브 마크와 제프 라마르쉬의 『Learn Cocoa on the Mac』을 추천한다.

각 프레임워크는 여러 기술의 커다란 집합체이며 종종 수십 또는 수백 개의 헤더 파일을 포함하고 있고 프레임워크의 각 헤더 파일 모두를 포함하는 마스터 헤더 파일을 가지고 있다. 마스터 헤더 파일에 `#import`를 사용하면 프레임워크의 모든 기능에 접근할 수 있다.

Foundation 프레임워크를 위한 헤더 파일은 거의 1MB의 디스크 용량을 차지하고 수백 개의 파일에 걸쳐있는 14,000줄 이상의 소스 코드를 포함한다. 여러분이 `#import <Foundation/Foundation.h>`로 Foundation 프레임워크의 마스터 헤더 파일을 포함했다면 이 거대한 내용을 사용할 수 있게 된다. 어쩌면 각 파일의 모든 텍스트를 포함하는데 컴파일러가 상당한 시간이 들 거라고 생각할 수도 있겠지만 Xcode는 똑똑하다. Xcode는 `#import`할 때 빠르게 로드되는 헤

더의 형태로 압축하고 군더더기를 빼서 만든 미리 컴파일된 헤더를 사용해서 작업의 속도를 높인다.

만일 Foundation 프레임워크로 어떤 헤더 파일들을 포함하게 되는지 궁금하다면 헤더 폴더(/System/Library/Frameworks/Foundation.framework/Headers/) 안의 내용을 살펴보도록 하자. 이 폴더의 파일을 살펴볼 때는 파일을 지우거나 변경하지 않는 것이 좋다.

NSLog( )와 @"문자열"

Foundation 프레임워크를 위한 마스터 헤더 파일에 `#import`를 사용해서 포함했으니 이제는 코코아 기능의 일부를 이용하는 코드를 만들 준비가 된 것이다. Hello Objective-C에서 실제로 사용된 처음이자 유일한 라인이 다음과 같은 `NSLog()` 함수이다.

```
NSLog (@"Hello, Objective-C!");
```

이 함수는 「Hello, Objective- C!」를 콘솔에 출력한다. C에 대한 경험이 조금이라도 있다면 아마도 `printf()` 함수에 대한 경험이 있을 것이다. `NSLog()`는 `printf()`와 아주 비슷한 기능을 하는 코코아 함수이다.

`printf()`와 마찬가지로 `NSLog()`도 첫번째 인수로 문자열을 받아들인다. 이 문자열은 `%d`와 같은 형식 기술자를 포함할 수 있고 함수는 형식 기술자와 같은 추가 파라미터를 받는다. `printf()`는 출력하기 전에 문자열에 이 추가 파라미터를 끼워 넣는다.

이전에 이야기했듯이 오브젝티브-C는 아주 약간 특별한 양념을 친 C와 같기 때문에 원한다면 `NSLog()` 대신 `printf()`를 그냥 쓸 수도 있다. 그러나 출력되는 시간과 날짜가 함께 출력되는 기능이나 새 라인 문자(\n)를 자동으로 넣어주는 등 부가적인 기능이 있으니 `NSLog()`를 사용하기를 추천한다.

`NSLog()`라는 함수 이름이 좀 낯설게 느껴질 수 있다. 함수 이름의 「NS」는 무슨 의미일까? 읽다 보면 코코아의 모든 함수, 상수, 타입(type) 이름에 NS가 앞에 붙는 것을 볼 수 있다. 이 접두사는 함수가 다른 툴킷에서 온 것이 아니라 코코아에서 왔다는 것을 알려준다.

이런 접두사는 같은 식별자가 두 개의 다른 대상에 쓰일 때 결과적으로 큰 문제가 되는 **이름 충돌**(name collisions) 사태를 막는데 도움이 된다. 만일 코코아가 `Log()`라는 이름의 함수를 가지고 있다면, 이를 잘 알지 못하는 프로그래머가 어딘가에서 만든 `Log()`라는 함수와 이름이 충돌할 수 있다. `Log()` 함수를 가지고 있는 프로그램이 코코아를 포함하여 빌드되면 Xcode는 `Log()`가 여러 번

정의되었다고 투덜댈 것이고 좋지 않은 결과, 즉 에러가 나타난다.

이제 접두사를 사용하는 것이 왜 좋은지 알게 되었다. 그런데 이번에는 다른 의문이 생길 것이다. 예를 들어, 왜 Cocoa를 접두사로 쓰지않고 NS를 사용했을까? NS 접두사는 이 툴킷이 NextSTEP이라고 불리며 NeXT Software(공식적으로는 NeXT, Inc.로서 1996년에 애플에 합병됨)의 제품이었던 때로 거슬러 올라간다. NextSTEP을 위해 이미 작성된 코드와의 호환을 유지하기 위해, 애플은 NS 접두사를 계속 쓰고 있다.

코코아는 접두사 NS에 대해서 엄격하게 관리하기 때문에, 당연히 여러분이 작성한 변수나 함수의 이름에는 NS라는 접두사를 사용하지 말아야한다. 만약 사용하게 되면 여러분이 작성한 코드를 읽는 사람들이 그 코드가 원래 코코아에 들어 있는 것인지 아닌지 헷갈리게 된다. 아울러 애플에서 코코아에 여러분이 만든 함수와 같은 이름의 함수를 추가하게 되면 그 코드는 앞으로 제대로 동작하지 않게 될 수도 있다. 코코아는 그 외의 접두사는 따로 관리하지 않으니 여러분이 원하는 접두사를 골라서 쓸 수 있다. 많은 사람들은 코드 작성자의 이름이나 회사의 이름 이니셜을 접두사로 사용한다. 우리의 예제를 조금 더 단순하게 만들기 위해서 이 책에서는 접두사를 사용하지 않을 것이다.

자, `NSLog()`문의 다른 부분에 대해서도 살펴보자.

```
NSLog (@"Hello, Objective-C!");
```

문자열 앞에 있는 기호에 주목해 보자. 책을 만들 때 실수로 들어간 글자가 아니다. 골뱅이(@, at) 기호는 표준 C언어에 추가된 오브젝티브-C의 특징 중 하나이다. 큰따옴표 사이의 문자열 앞에 골뱅이가 있는데, 이는 큰따옴표 안의 문자열을 코코아 `NSString`의 요소로 처리해야 한다는 의미이다.

그럼 `NSString` 요소라는 것은 무슨 뜻일까? `NSString`에서 NS 접두사를 떼고 보면 그냥 String이라는 친숙한 단어가 남는다. 여러분이 이미 알고 있는 것처럼 String은 문자열이고, 문자열은 보통 사람이 읽을 수 있는 문자의 나열이다. 따라서 `NSString`은 코코아에서 문자의 나열을 의미함을 알 수 있을 것이다.

`NSString` 요소는 상당히 많은 수의 기능을 담고 있으며 코코아에서 문자열이 필요한 곳이면 언제 어디서나 사용된다. 다음은 `NSString`의 기능을 몇 가지 간추려본 것이다.

- 문자열의 길이를 알려줌
- 다른 문자열과의 비교

■ 정수 또는 실수로의 값 변환

NSString은 C의 문자열보다 더 많은 일을 할 수 있다. NSString의 설명 및 사용법은 8장에서 자세히 알아보도록 하겠다.

NSString이 좋은 점이 여기서도 드러난다. 바로 이름 자체가 코코아의 멋진 특징 중 하나라는 것이다. 대부분 코코아의 구성요소들은 매우 직관적인 방식으로 이름이 정해지는데, 해당 구성요소가 구현된 특징을 설명하도록 노력하고 있다. 예를 들어 NSArray는 배열(array)을 제공하고 NSDateFormatter는 여러 방식으로 날짜의 형식을 바꿀 수 있도록 해 주며, NSThread는 멀티 스레드 프로그래밍을 하는 도구를 제공하고 NSSpeechSynthesizer는 말소리를 들을 수 있도록 한다.*

이제, 우리의 첫 번째 프로그램으로 다시 돌아가자. 프로그램의 마지막 라인은 main() 함수의 실행을 끝내는 return 문이고 프로그램을 종료한다.

```
return (0);
```

0 값은 우리의 프로그램이 성공적으로 끝났다는 것을 나타내는 반환값이다. 이 부분은 C의 return 문이 동작하는 것과 같다.

다시 한 번 축하한다! 여러분은 방금 첫 번째 오브젝티브-C 프로그램을 작성하였고 컴파일을 했으며 실행해 보고 분석까지 해 봤다.

불리언 타입을 알고 있는가?

다양한 언어가 불리언(Boolean) 타입을 가지고 있다. 이 불리언은 참(true)과 거짓(false) 값을 저장하는 변수를 위한 타입이다. 오브젝티브-C라고 해서 다를 것은 없다.

C에는 true와 false 값을 담을 수 있는 불리언 데이터 타입으로 bool이 있다. 오브젝티브-C도 YES와 NO를 저장할 수 있는 BOOL이라는 비슷한 타입을 제공한다. 그런데 오브젝티브-C의 BOOL 타입은 C의 bool 타입보다 10년은 먼저 만들어졌다. 이 두 개의 다른 불리언 타입은 같은 프로그램에서 같이 쓸 수 있지만 코코아 코드를 작성하는 경우에는 BOOL을 사용하게 될 것이다.

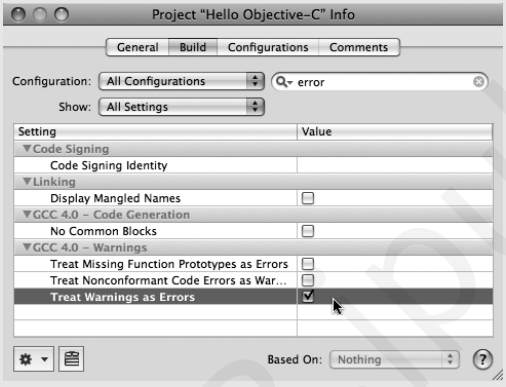
* 역주 : 맥은 기본적으로 TTS(텍스트 음성 변환, Text to Speech)가 들어있으며 제공 소프트웨어인 기본 「텍스트 편집기」에서 문장을 입력한 후 읽을 수 있다.

NOTE 다음 문자열을 살펴보자

자주 하는 실수 중 하나는 `NSLog()`에 `NSString @"strings"` 대신에 C 스타일의 문자열을 넣는 경우이다. 그렇게 하면 컴파일러는 다음과 같은 경고를 내보낸다.

```
main.m:46: warning: passing arg 1 of 'NSLog' from incompatible pointer type
```

이 프로그램을 실행하면 제대로 실행되지 않는다. 이런 문제를 잡기 위해서는 Xcode가 경고를 에러로 간주해서 메시지를 뿌리도록 설정하는 것이 좋다. 그렇게 설정하려면 Xcode의 왼쪽 창 Group & Files에 있는 목록에서 가장 위에 있는 아이템을 선택하고 메뉴의 **File ▶ Get Info**를 선택한 후 Build 탭을 선택한다.* 그리고 나서 아래 그림처럼 검색 필드에 error라고 입력하고 가장 아래 Treat Warning as Errors의 체크박스를 선택한다. 그리고 좀 전에 error를 검색한 필드 왼쪽 Configuration에서 가장 위에 있는 All Configurations를 선택하면 된다.



실전에 강한 BOOL

실전에 강한 `BOOL`을 보여주기 위해 두 정수 값이 다른지 같은지 비교하는 프로젝트(예제 폴더 02.02 BOOL Party)로 넘어가 보자. `main()` 함수 이외에도 이 프로그램은 두 개의 함수를 정의하고 있다. 우선 `areIntsDifferent()` 함수는 두 개의 정수 값을 받아서 `BOOL` 값을 반환하는데, 두 정수가 다르다면 `YES`, 두 정수가 같으면 `NO`를 반환한다. 두 번째 함수 `boolString()`은 `BOOL` 파라미터를 받아서 파라미터가 `YES`인 경우는 `@"YES"`를, `NO`인 경우는 `@"NO"`를 반환한다. 이런 함수는 `BOOL` 값을 사람이 읽을 수 있는 형태로 출력하고 싶을 때 쓸 수 있는 함수이다. `main()`

* 역주 : 이 과정은 Xcode에서 왼쪽 창의 Group & Files의 바로 아래 있는 Hello Objective-C 프로젝트를 더블 클릭하여 동일하게 실행할 수 있다.

NOTE

오브젝티브-C에서 BOOL은 실제로 8비트의 저장 공간을 갖는 부호 있는 문자(signed char) 타입을 단지 형 정의(typedef)한 것이다. YES는 1로 정의되어 있고 NO는 0으로 정의되어 있다(#define 사용). 오브젝티브-C는 BOOL을 YES나 NO의 값만을 담을 수 있는 진정한 불리언 타입으로 여기지 않는다. 컴파일러는 BOOL을 8비트 숫자로 인식하고 YES와 NO의 값은 단지 관습이다. 이는 미묘한 결과를 낳는다. 만일 무심코 1바이트보다 큰 값의 정수(예를 들어 short나 int 값 등)를 BOOL 변수에 넣는다면, BOOL 변수의 값에서 하위 바이트만 사용한다. 하위 바이트가 0인 경우(예를 들어 8960, 16진수로 0x2300 등) BOOL 값은 0, 즉 NO값이 된다.

함수는 두 정수를 비교하고 그 결과를 출력하는데 이 두 함수를 사용한다.

Bool Party를 위한 프로젝트 생성도 Hello Objective-C 프로젝트를 만드는 과정과 동일하다.

1. Xcode가 실행되고 있지 않은 상태라면 Xcode를 실행한다.
2. File ▶ New Project를 선택한다.
3. 왼쪽에서 Command Line Utility를 선택하고 나서 오른쪽에 Foundation Tool을 선택한다.
4. Choose를 선택한다.
5. Project Name을 Bool Party라고 입력하고 Save 버튼을 클릭한다.

Bool party.m 파일을 다음과 같이 수정한다.

```
#import <Foundation/Foundation.h>

// returns NO if the two integers have the same
// value, YES otherwise

BOOL areIntsDifferent (int thing1, int thing2)
{
  if (thing1 == thing2) {
     return (NO);
  } else {
     return (YES);
  }


} // areIntsDifferent
// given a YES value, return the  human-readable
// string "YES". Otherwise return "NO"

NSString *boolString (BOOL yesNo)
```

```
{
  if (yesNo == NO) {
     return (@"NO");
  } else {
     return (@"YES");
  }

} // boolString


int main (int argc, const char *argv[])
{
  BOOL areTheyDifferent;

  areTheyDifferent = areIntsDifferent (5, 5);

  NSLog (@"are %d and %d different? %@",
            5, 5, boolString(areTheyDifferent));

  areTheyDifferent = areIntsDifferent (23, 42);

  NSLog (@"are %d and %d different? %@",
            23, 42, boolString(areTheyDifferent));

  return (0);

} // main
```

방금 작성한 프로그램을 빌드해서 실행해 보자. 출력을 보기 위해 Run ▶ Console을 선택하거나 키보드 단축키 ⌘+Shift+R을 사용해서 나오는 Console 윈도우를 띄울 필요가 있다. Run Debugger Console 윈도우에 다음과 같이 출력되는 것을 볼 수 있을 것이다.

```
2008-07-20 16:47:09.528 02 BOOL Party[16991:10b] are 5 and 5
different? NO
2008-07-20 16:47:09.542 02 BOOL Party[16991:10b] are 23 and 42
different? YES
The Debugger has exited with status 0.
```

다시 한 번 이 프로그램을 함수 단위로 나눠서 어떻게 동작하는지 알아보자. 우리가 알아볼 첫 번째 함수는 `areIntsDifferent()` 함수이다.

```
BOOL areIntsDifferent (int thing1, int thing2)
{
  if (thing1 == thing2) {
     return (NO);
  } else {
```

```
        return (YES);
    }

} // areIntsDifferent
```

areIntsDifferent() 함수는 두 정수 파라미터를 받아서 BOOL 값을 반환한다. C에 경험이 있다면 문법은 익숙할 것이다. thing1을 thing2와 비교하는 부분을 볼 수 있을 것이다. 만일 같으면, 다르지 않기 때문에 NO를 반환한다. 다른 경우에는 YES를 반환한다. 상당히 직관적이다. 그렇지 않은가?

두 번째 함수는 boolString()이다. 이 함수는 숫자 BOOL 값을 사람이 읽을 수 있는 문자열로 바꿔준다.

NOTE 쉽게 오해할 수 있는 불리언

경험 많은 C 프로그래머라면 areIntsDifferent() 함수를 다음과 같이 한 줄로 쓸 수도 있다.

```
BOOL areIntsDifferent_faulty (int thing1, int thing2)
{
  return (thing1 - thing2);
} // areIntsDifferent_faulty
```

이들이 이렇게 작성할 수 있는 것은 0이 아닌 값이 YES라는 가정을 하기 때문이다. 그러나 이 경우는 다르다. 이 함수가 반환하는 값은 C에서는 참 또는 거짓이지만 BOOL을 반환하는 함수의 호출자는 YES나 NO가 반환되기를 기대한다. 만일 프로그래머가 이 함수를 다음과 같이 사용한다면 23 빼기 5가 18이기 때문에 실패한다.

```
if (areIntsDifferent_faulty(23, 5) == YES) {
  // ....
}
```

위 함수가 C에서는 참 값일 수 있겠지만, 오브젝티브-C에서는 YES(1의 값)와는 다르다. BOOL 값을 바로 YES와 비교하지 않는 것이 좋다(비교하려면 상당히 주의를 기울여 처리해야 한다). 왜냐하면 너무나 영리한 프로그래머들도 가끔은 areIntsDifferent_faulty()와 비슷한 위험한 시도를 하기 때문이다. 대신 다음과 같이 if문을 쓰자.

```
if (areIntsDifferent_faulty(5, 23)) {
  // ....
}
```

이에 반해 NO를 바로 비교하는 것은 안전한데, C에서 false는 0 하나밖에는 없기 때문이다.

```
NSString *boolString (BOOL yesNo)
{
  if (yesNo == NO) {
     return (@"NO");
  } else {
     return (@"YES");
  }

} // boolString
```

함수의 가운데 있는 if 문은 이제 놀랄 것이 못 된다. 단지 yesNo를 상수 NO와 비교해서 같은 경우 @"NO"를 반환한다. 그렇지 않으면 yesNo가 반드시 참 값이 되어야 하기 때문에 @"YES"를 반환한다.

boolString() 함수의 반환 타입이 NSString을 가리키는 포인터라는 사실에 주의하자. 이는 전에 NSLog()를 처음 봤던 코코아 문자열의 하나를 반환하는 함수라는 것을 의미한다. 반환문을 살펴보면 반환 값의 바로 앞에 골뱅이 기호가 있는 것을 볼 수 있다. 이것이 NSString 값이라는 결정적인 증거이다.

main() 함수는 마지막 함수이다. main() 함수의 반환 타입과 인수의 선언을 준비한 다음에 로컬 BOOL 변수가 있다.

```
int main (int argc, const char *argv[])
{
  BOOL areTheyDifferent;
```

areTheyDifferent 변수는 areIntsDifferent()가 반환하는 YES 또는 NO 값을 담는다. 우리는 단순하게 함수의 BOOL 반환 값을 if 문처럼 직접 사용할 수 있지만 코드를 읽기 쉽게 하기 위해서 여기처럼 추가변수를 사용하는 것도 좋은 방법이다. 깊게 중첩된 구조는 이해하기 힘든 경우가 많다. 아울러 이런 곳은 버그가 숨어 있기 좋은 장소이기도 하다.

자기 자신의 비교

다음 두 줄의 코드는 areIntsDifferent()로 두 정수를 비교해서 areTheyDifferent 변수에 반환값을 저장한다. NSLog()는 숫자 값과 boolString() 함수에 의해 반환되는, 사람이 읽을 수 있는 형태의 문자열을 출력한다.

```
    areTheyDifferent = areIntsDifferent (5, 5);
```

```
    NSLog (@"are %d and %d different? %@",
        5, 5, boolString(areTheyDifferent));
```

이미 이전에 본 것처럼 NSLog()는 형식 문자열을 받아서 형식 식별자에 꽂아 넣을 수 있는 추가 파라미터를 사용하는 기본적인 코코아식 printf() 함수이다. 위에서 NSLog() 호출에서 두 개의 5로 치환되는 두 개의 %d를 볼 수 있다.

NSLog()에 전달한 문자열의 끝에 또 하나의 골뱅이 기호가 있다. 여기서는 바로 %@이다. 무슨 의미일까? boolString() 함수는 NSString 포인터를 반환한다. printf()는 NSString이 어떻게 동작하는지 모르기 때문에 우리가 사용할 수 있는 포맷 식별자가 없다. NSLog()를 만든 사람들은 문자열에서 문자를 사용하고 콘솔에 출력하도록 보내는 NSString처럼 적절한 인수를 받아들이는 NSLog() 명령에 %@ 포맷 식별자를 추가했다.

NOTE

아직 객체에 대해 설명하지는 않았지만, 여기서 간단히 살펴보자. NSLog()를 사용해서 어떤 객체의 값을 출력할 때, %@ 포맷 명세를 사용한다. 이 식별자를 사용할 때, 객체는 description이라는 이름의 메서드를 통해 자신의 NSLog() 포맷을 제공한다. NSString의 description 메서드는 단순히 문자열의 문자들을 출력한다.

다음 두 줄은 방금 본 것과 상당히 비슷하다.

```
    areTheyDifferent = areIntsDifferent (23, 42);

    NSLog (@"are %d and %d different? %@",
        23, 42, boolString(areTheyDifferent));
```

함수가 23과 42 값을 비교한다. 이번에는 두 값이 다르기 때문에 areIntsDifferent()는 YES를 반환하고 사용자는 23과 42가 다른 값이라는 사실을 나타내는 문자열을 보게 된다.

다음은 BOOL Party의 최종 결론을 보여주는 return 문이다.

```
    return (0);

}  // main
```

이 프로그램에서 여러분은 오브젝티브-C의 BOOL 타입과 참과 거짓 값을 나타내는 상수인 YES와 NO를 살펴보았다. int와 float 등의 타입을 사용하듯 BOOL을 변수, 함수의 파라미터, 함수의

반환값으로 사용할 수 있다.

요약

이 장에서 여러분은 처음으로 두 개의 오브젝티브-C 프로그램을 만들어 보고 재미있다는 사실을 알게 됐다. 또한 컴파일러에게 같은 헤더를 한 번만 포함하도록 하도록 알려주는 `#import`와 같은 언어의 오브젝티브-C 확장도 약간 알아보았다. `NSString` 문자열에 대해서도 배웠는데, `@"hello"`처럼 문자열 앞에 골뱅이가 붙는다. `NSLog()` 함수라는 중요하고 여러 용도로 쓰일 수 있는 도구도 보았는데, 이는 콘솔에 텍스트를 출력하기위해 코코아가 제공하는 함수이다. 또한, `NSLog()`의 특수한 포맷 식별자 `%@`는 `NSLog()` 출력에 `NSString`을 출력할 수 있도록 한다. 그리고 코드에서 골뱅이 기호가 나오면 이는 바로 C언어의 오브젝티브-C 확장을 의미하는 암호라는 것도 알게 되었다.

그럼, 한눈 팔지 말고 객체 지향 프로그래밍의 신비한 세계를 다루게 될 다음 장을 기대하시라!