Chapter 23

오브젝티브-C 함수의 개요

함수는 잘 짜인 구조와 효과적인 코드를 만드는 데 매우 중요한 것이다. 오브젝티브-C 함수는 프로그램을 체계화하며 코드의 반복 사용을 피하는 방법을 제공한다. 이번 장에서는 함수를 선언하는 방법과 사용하는 방법을 살펴볼 것이다.

23.1 함수란 무엇인가?

함수(function)는 특정 작업을 수행하기 위해 호출되는 코드 블록의 이름이다. 함수는 작업을 수행하기 위한 데이터를 받을 수도 있으며, 함수를 호출하는 코드에 수행한 결과를 반환할 수도 있다. 예를 들어, 오브젝티브-C 프로그램 내에서 어떤 산술 계산을 해야 한다면, 그 계산을 하기 위한 코드는 함수 안에 위치할 수가 있다. 이 함수는 산술 계산을 수행하기 위한 값들을 받고(인자를 이용하여) 계산의 결과를 반환하도록 프로그래밍될 수 있다. 프로그램 안에서 이 계산이 필요한 곳에서는 이 함수를 쉽게 호출하여 그 결과를 반환받게 된다.

23.2 오브젝티브-C 함수를 선언하는 방법

오브젝티브-C 함수는 다음과 같은 문법으로 선언된다.

```
<반환 타입> <함수 이름> (<인자1 타입> <인자1 이름>, <인자2 타입> <인자2 이름>, ... )
{
// 함수 코드
}
```

함수 선언에 필요한 각 부분에 대한 설명은 다음과 같다.

- **〈반환 타입〉** – 함수에 의해 반환되는 결과의 데이터 타입을 지정한다. 만약 함수가 아무런 결과를 반환하지 않는다면, void를 사용해야 한다. 만일 지정하지 않는다면, 함수는 int가 반환될 것이라고 가정한다.
- **〈함수 이름〉** – 해당 함수에 주어진 이름이다. 이것은 애플리케이션 내의 코드에서 함수를 호출할 때 불리게 될 이름이다. 만약 static 지정자를 사용하지 않는다면 함수 이름은 global로 간주되며, 컴파일 에러를 피하기 위하여 애플리케이션 내에서 고유한 이름이어야 한다.
- **〈인자n 타입〉** – 함수에 전달될 인자의 타입이다.
- **〈인자n 이름〉** – 함수에서 참조될 인자의 이름이다.
- **함수 코드** – 실제로 동작할 함수의 코드다.

예를 들어, 다음의 함수는 인자를 받지 않으며, 결과를 반환하지도 않고, 단순히 메시지만을 표시한다.

```
void sayhello ()
{
    NSLog (@"Hello");
}
```

반대로, 다음의 샘플 함수는 두 개의 정수 인자를 받아서 그 숫자를 곱한 결과를 반환한다.

```
int multiply (int x, int y)
{
    return x * y;
}
```

23.3 main() 함수

이 책을 순서대로 읽어 왔다면 이전에 있던 많은 예제에 main이라고 불리는 함수가 있었던 것을 기억할 것이다. 이것은 오브젝티브–C 컴파일러에게 프로그램이 실행되는 시작점이 어디인지를 알려 주는 특별한 함수 이름이다. 만일 여러분의 코드에 main 함수가 없다면, 빌드 과정에서 에러가 날 것이다. main 함수의 문법은 다음과 같다.

```
int main (int argc, const char * argv[])
{
    // 코드가 있어야 할 위치
}
```

argc 인자는 프로그램이 실행될 때 커맨드 라인에 있는 인자의 개수를 담으며, argv는 그 인자들을 가지고 있는 배열의 포인터다.

23.4 오브젝티브 함수 호출하기

함수가 한번 선언되면, 동일한 소스 파일에 내에 있든지 아니면 다른 파일에 있든지와는 상관없이 어디서든 호출될 수 있게 된다. 그 이유는 디폴트로 함수는 global이기 때문이다. 각각의 소스 파일들이 객체 코드로 컴파일되면, 링커(linker)라고 불리는 도구는 각각의 객체 파일 내에 정의되지 않은 기호(symbol)들을 분석하는 작업을 수행한다. 이 과정에서 어떤 객체 파일에 있는 정의되지 않은 함수에 대한 호출이 발견되면, 그 코드를 구성하는 다른 객체 파일과 지정된 라이브러리를 찾을 때까지 검색한다. 만약 일치하는 것을 찾지 못한다면, 링커는 정의되지 않은 기호에 대한 오류를 낼 것이다.

함수는 다음과 같은 문법으로 호출된다.

```
<함수 이름> (<인자1>, <인자2>, ... )
```

예를 들어, 인자를 받지 않으며 어떠한 값도 반환하지 않는 sayHello라는 이름의 함수를 호출하려면, 다음과 같은 코드를 쓸 것이다.

```
sayhello();
```

두 개의 인자를 받으며 정수를 반환하는 multiply라는 함수를 호출하려면, 다음과 같은 코드를 쓸 것이다.

```
int result;
result = multiply (10, 20);
```

위의 예제에서는 result라는 새로운 변수를 생성하였고, 인자로 10과 20을 전달한 multiply 함수에서 반환하는 결과를 저장하기 위하여 할당 연산자(=)를 사용하였다.

23.5 함수 프로토타입

함수가 호출되는 위치에 따라 그 함수가 선언되는 위치는 중요하다. 책의 내용을 위에서 아래로 읽는 것과 같은 식으로, 컴파일러도 오브젝티브-C 소스 파일을 위에서 아래로 읽는다. 만약 컴파일러가 어떤 함수에 대한 선언을 발견하기 전에 그 함수를 호출하는 코드를 먼저 발견한다면, 컴파일러는 그 함수가 반환하는 결과에 대한 타입을 추측해야 한다. 디폴트로 정해진 것은 정수(int)를 반환한다고 가정하는 것이다. 이런 가정을 가지고 진행하다가, 컴파일러가 그 함수에 대한 선언을 발견하여 그 함수를 반환하는 데이터 타입이 int가 아니면 에러를 발생할 것이다. 이에 대한 것을 확인하기 위해서 다음의 코드를 컴파일해보자.

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

int main (int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    float result;

    result = multiply( 10, 20 );
    [pool drain];
    return 0;
}

float multiply (int x, int y)
{
    return x * y;
}
```

위의 코드를 컴파일하면 다음과 같은 메시지와 함께 에러가 날 것이다.

```
sample.m:18: error: conflicting types for 'multiply'
sample.m:10: error: previous implicit declaration of 'multiply' was here
```

컴파일러는 에러를 표시하고 있다. 왜냐하면 10번 줄에서 함수가 호출될 때 int를 반환한다고 가정하기 때문이다. 그런 이유는 그 위치에 올 때까지 함수 선언을 발견하지 못해서 그렇다. 컴파일러는 함수를 선언한 부분이 없었기 때문에 디폴트로 정수를 반환한다고 가정하였는데, 실제 함수 선언에서 사실은 float를 반환하고 있기 때문에 혼란이 생기게 되었다.

이런 문제를 해결하기 위한 방법은 두 가지가 있다. 첫 번째 방법은 항상 함수에 대한 선언을 호출되기 전에 하는 것이다.

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

float multiply (int x, int y)
{
    return x * y;
}

int main (int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    float result;
    result = multiply( 10, 20 );
    [pool drain];
    return 0;
}
```

이것은 간단한 경우에 해결하는 방법이지만, 여러 개의 파일과 다른 함수들을 호출하는 함수들을 가진 큰 규모의 애플리케이션 코드에서는 관리하기가 어려워질 수가 있다. 더 좋은 해결책은 함수의 프로토타입을 사용하는 것이다. 이것은 반환 타입과 함수의 인자에 대한 정보를 파일의 상단에 두어 미리 선언되게 할 수 있는 선언부이다. 예를 들어, multiply 함수에 대한 함수 프로토타입은 다음과 같다.

```
float multiply (int x, int y);
```

함수 프로토타입에 있는 인자 이름은 마음대로 정할 수 있으며, 인자에 대한 타입만 써도 동일한 결과를 얻을 수 있다.

```
float multiply (int, int);
```

세미콜론(;)은 반드시 써야 한다.

이것을 앞의 예제에 적용하면 다음과 같다.

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

float multiply (int x, int y);

int main (int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    float result;

    result = multiply( 10, 20 );
    [pool drain];
    return 0;
}

float multiply (int x, int y)
{
    return x * y;
}
```

여러 개의 인자들을 받는 함수를 위해서 함수 프로토타입에 '...' 지시자를 사용할 수 있다.

```
int addAll (int, ...);
```

23.6 함수의 범위와 static 지정자

오브젝티브-C 함수들은 기본적으로 그 범위가 전역으로 간주된다. 이 말은 어떤 프로그램 내의 파일에 선언된 함수는 그 프로그램의 다른 파일들의 코드에서 호출될 수 있다는 것을 의미한다. 이 의미는 함수 이름들은 각각 유일해야 한다는 뜻이다. 동일한 이름으로 된 두 개의 함수가 있다면 빌드(build) 과정에서 에러가 발생할 것이다. 함수의 범위를 선언된 파일로 제한하려면 함수 선언부 앞에 static 키워드를 두자.

```
static float multiply (int x, int y)
{
    return x * y;
}
```

23.7 함수의 정적 변수

프로그램이 실행되는 일반적인 과정에서 그 함수가 종료되고 그 함수를 호출한 위치의 코드에 값을 반환한 다음에는 함수 내에서 선언된 모든 변수들은 없어진다. 예를 들어, 다음의 코드에서 displayit 함수가 호출될 때마다 변수 y는 0으로 다시 초기화된다.

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

void displayit (int);

int main (int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int i;
    for (i=0; i<5; i++)
    {
        displayit( i );
    }
    [pool drain];
    return 0;
}
void displayit (int i)
{
    int y = 0;
    y += i;
    NSLog (@"y + i = %i", y);
}
```

위의 코드를 실행하면 다음과 같은 결과를 얻게 된다.

```
2009-10-20 15:39:12.306 t[10128:10b] y + i = 0
2009-10-20 15:39:12.308 t[10128:10b] y + i = 1
2009-10-20 15:39:12.308 t[10128:10b] y + i = 2
2009-10-20 15:39:12.309 t[10128:10b] y + i = 3
2009-10-20 15:39:12.309 t[10128:10b] y + i = 4
```

만약 함수가 실행된 다음에도 y에 대한 값을 유지하고 싶다면, 이 변수를 static으로 선언하면 된다.

```
static int y = 0;
```

이제 이 코드를 다시 컴파일하여 실행하면, 다음과 같은 결과를 얻을 수가 있다. 왜냐하면 이 함수가 호출될 때마다 y가 0으로 다시 설정되지 않기 때문이다.

```
2011-09-04 17:24:48.194 t[10136:10b] y + i = 0
2011-09-04 17:24:48.195 t[10136:10b] y + i = 1
2011-09-04 17:24:48.195 t[10136:10b] y + i = 3
2011-09-04 17:24:48.196 t[10136:10b] y + i = 6
2011-09-04 17:24:48.196 t[10136:10b] y + i = 10
```