Chapter 7

오브젝티브-C 2.0의 데이터 타입

금융 애플리케이션들부터 그래픽이 집약된 게임들까지 컴퓨터 시스템에서 실행되는 다양한 종류의 소프트웨어들을 이용하다 보면, 컴퓨터가 단지 2진수로 동작하는 기계라는 사실을 쉽게 잊게 된다. 이진(binary) 시스템은 0과 1, 참과 거짓, 그리고 설정과 해제로 동작한다. 디스크 드라이브에 저장되어 RAM에 올라가는 데이터와 회로판과 버스를 통해 흐르는 모든 데이터는 일련의 1과 0일 뿐이다. 각각의 1과 0은 비트(bit)로 불리며, 비트는 8개가 하나로 묶이며, 각각의 묶음은 바이트(byte)로 불린다. 사람들이 32비트 컴퓨터와 64비트 컴퓨터에 대해 이야기할 때 CPU 버스가 동시에 처리할 수 있는 비트 수에 대해 이야기를 한다. 예를 들어, 64비트 CPU는 64비트 블록의 데이터를 처리할 수 있어서 32비트 기반의 시스템보다 더 빠른 성능의 결과를 낼 수 있다.

물론 사람들은 2진수로 생각하지 않는다. 우리는 10진수와, 문자, 그리고 단어로 생각하고 이해한다. 그래서 사람들이 쉽게 프로그래밍을 할 수 있도록 사람이 생각하는 방법과 컴퓨터가 생각하는 방법 사이에 어떤 중간 그룹이 필요하다. 이것이 바로 오브젝티브-C와 같은 프로그래밍 언어가 활동하는 위치다. 프로그래밍 언어는 사람이 컴퓨터에게 할 명령어들을 표현할 수 있도록 해주고, 우리가 이해할 수 있는 구조로 구성할 수 있게 해준다. 그런 다음, 컴퓨터가 실행할 수 있는 형태로 컴파일할 수 있게 해준다.

모든 프로그램의 기본 중에 하나는 데이터이며, 오브젝티브-C와 같은 프로그래밍 언어

는 프로그램을 개발할 때 우리가 알 수 있는 형태로 데이터를 사용할 수 있도록 하는 데이터 타입들을 가지고 있다. 예를 들어, 오브젝티브–C에서 숫자를 저장할 경우는 다음과 같은 문장을 사용한다.

```
int mynumber = 10;
```

위의 예에서는 int라는 키워드를 사용하여 mynumber라는 이름의 정수형 변수를 생성한다. 그런 다음, 이 변수에 10을 할당한다. int가 정수형 변수를 지정하고 있다는 의미임을 알게 된다면, 우리는 오브젝티브–C 프로그램의 특정 위치에서 어떤 일이 일어나고 있는지를 이해하게 된다. 이 소스 코드를 CPU가 사용하는 코드로 컴파일한다면, 컴퓨터는 10이라는 숫자를 다음과 같은 2진수로 받아들일 것이다.

```
1010
```

마찬가지로 우리는 문자, '0'부터 '9'까지의 문자화된 숫자, 또는 문장부호(punctuation mark)를 다음과 같은 문장으로 표현할 수 있다.

```
char myletter = 'c';
```

다시 한 번 말하자면, 위의 코드는 프로그래머가 이해할 수 있는 코드이지만 CPU가 이해할 수 있도록 2진수로 컴파일돼야 한다. 이 예에서 'c'라는 문자는 ASCII 테이블(사람이 읽을 수 있는 문자에 숫자 값을 할당한 국제적으로 공인된 표준)에서 99라는 숫자를 의미한다. 이 값을 2진수로 변환하면 다음과 같은 값이 된다.

```
1010110011
```

우리는 데이터 타입의 개념과 필요한 이유에 대한 기본적인 이해를 했다. 이제는 오브젝티브–C에서 지원하는 다양한 데이터 타입들과 규정들에 대해 자세히 살펴보자.

7.1 int 데이터 타입

오브젝티브–C에서의 int 데이터 타입은 양수 또는 음수(다시 말해서 소수점이 없는 숫자)를 저장할 수 있다. 정수의 실제 크기(범위)는 컴퓨터와 컴파일에 따라 다를 수 있다. 일반적으로 int 값에 할당되는 저장공간의 크기는 컴파일러가 실행되는 플랫폼이나 CPU에 따라 32비트 또는 64비트가 되며, 운영체제 또한 int 값이 32비트가 될지 아니면 64비

트가 될지를 결정하는 중요한 요소가 된다. 예를 들어, 컴퓨터의 CPU는 64비트이지만 운영체제가 32비트로만 실행될 수도 있다.

32비트로 구현된 것을 예로 들면, 부호없는 정수(unsigned int)의 최대 범위는 0에서 4,294,967,295이지만, 64비트에서는 0에서 18,446,744,073,709,551,615가 된다. 부호있는 정수(signed int)일 경우는 32비트에서는 −2,147,483,648에서 +2,147,483,647이며, 64비트에서는 −9,223,372,036,854,775,808에서 +9,223,372,036,854,775,807이 된다.

오브젝티브-C 프로그램을 개발할 때 여러분에게 보장된 것은 int가 적어도 32비트가 될 것이라는 점이다. 다른 플랫폼에서 코드를 컴파일할 경우에 생길 수 있는 잠재적인 문제를 피하기 위해서는 int 값을 64비트 범위라고 가정하지 말고 32비트 범위로 제한하여 사용하는 것이 더 안전하다.

기본적으로 int 값은 10진수다. int 값을 8진수로 표현하려면 다음의 예제처럼 해당 값 앞에 0을 붙이자.

```
int myoctal = 024;
```

마찬가지로 16진수로 표현하려면 다음의 예제처럼 0x를 붙이자.

```
int myhex = 0xFFA2;
```

7.2 char 데이터 타입

오브젝티브-C의 char 데이터 타입은 단일 문자나 문자화된 숫자, 문장부호, 또는 빈 문자를 저장하는 데 사용된다. 다음의 예제는 다양한 문자들을 char 타입의 변수에 할당하고 있다.

```
char myChar = 'w';
char myChar = '2';
char myChar = ':';
```

7.2.1 특수 문자/이스케이프 문자

위에서 설명한 표준 문자들뿐 만 아니라 줄바꿈 또는 탭과 같은 것들을 지정하는 특수 문자들(이스케이프 문자라고도 함)도 있다. 이러한 특수 문자들은 역슬래시를 문자 앞에 두어 표시한다. 예를 들어, 다음의 예제는 newline이라는 이름의 변수에 줄바꿈을 할당한다.

```
char newline = '\n';
```

근본적으로 역슬래시가 앞에 붙은 모든 문자들은 특수 문자로 간주되며, 또 그렇게 취급된다. 여기서 여러분이 실제 역슬래시 문자를 사용하고 싶다면 어떻게 해야 할까? 이것은 역슬래시 자체에 역슬래시를 붙이는 방식으로 해결된다.

```
char myslash = '\\'; 역슬래시를 변수에 할당
```

오브젝티브–C에서 지원되는 일반적인 특수 문자들은 다음과 같다.

/a – 경고음
/b – 백스페이스
/f – 폼 피드
/n – 뉴 라인
/r – 케리지 리턴
/t – 수평 탭
/v – 수직 탭
// – 역슬래시
/" – 큰따옴표(문자열 내용에 큰따옴표를 쓰려고 할 때 사용됨)
/' – 작은따옴표(문자열 내용에 작은따옴표를 쓰려고 할 때 사용됨)

7.3 float 데이터 타입

오브젝티브-C의 float 데이터 타입은 부동소수점 값을 저장하는 데 사용된다. 즉, 이 값은 소수점 이하의 자릿수를 포함한다. 예를 들어, 456.12는 float 데이터 타입으로 저장될 것이다. 사실, 모든 부동소수점 값은 디폴트로 double이라는 데이터 타입으로 저장된다. 우리는 double 데이터 타입에 대해서는 다음 절에서 다룰 것이다. 만일 여러분이 명확하게 float 데이터 타입을 사용하고 싶다면, 다음의 예제와 같이 값 뒤에 f를 붙여야 한다.

```
float myfloat = 123.432f
```

예상하지 못할 정도로 긴 값을 가지고 작업할 경우에 편하게 하기 위해서 부동소수점 데이터 타입과 double 데이터 타입 모두 과학적인 표기법(표준식 또는 지수 표기법으로 알려진)을 사용하여 지정될 수가 있다. 예를 들어, 오브젝티브-C에서는 67.7×10^4을 다음과 같이 표현할 수가 있다.

```
float myfloat = 67.7e4
```

7.4 double 데이터 타입

오브젝티브-C의 double 데이터 타입은 float 데이터 타입으로 처리될 수 있는 값보다 더 큰 값을 저장하는 데 사용된다. double이라는 용어는 float의 두 배 크기의 값을 처리할 수 있다는 사실에서 비롯된 것이다. 앞에서 말했듯이 모든 부동소수점 값은 float로 지정하기 위한 'f'가 붙지 않았다면 double 데이터 타입으로 저장된다.

7.5 id 데이터 타입

이 책 뒤에 나오는 장에서 살펴보겠지만, 오브젝티브-C는 객체지향 언어다. 이런 방식의 언어들처럼 구조화될 프로그램은 재사용할 수 있는 객체 형태가 된다. 이런 객체들은

작업을 수행하고 결과 값을 반환하기 위해서 호출된다. 보통, 객체에 전달된 정보와 반환된 결과 값은 또 다른 객체의 형태로 된다. id 데이터 타입은 객체의 타입과는 상관없이 모든 객체에 대한 참조체(reference)를 저장하는 데 사용될 수가 있는 일반적인 목적의 데이터 타입이다.

7.6 BOOL 데이터 타입

다른 언어들과 마찬가지로 오브젝티브-C는 참과 거짓(1과 0)을 처리하기 위한 데이터 타입을 가지고 있다. 이 데이터 타입은 _Bool이나 BOOL 키워드를 이용하여 선언된다. 다음과 같은 문장은 모두 사용이 가능하다.

```
_Bool flag = 0;
BOOL secondflag = 1;
```

7.7 오브젝티브-C 데이터 타입 수식자들

지금까지는 오브젝티브-C 프로그래밍 언어의 문장에 제공되는 기본 데이터 타입들을 살펴보았다. 우리는 여러 가지 다양한 데이터 선언을 위해 제공되는 데이터 타입들과 필요한 크기에 대해 알아보았으며, 어떤 종류의 데이터를 담을 수 있는가에 대한 제약들을 가진 각각의 데이터 타입에 대해 배웠다. 사실, 이런 제약들 중에 몇몇은 수식자(qualifier)를 사용하여 바꿀 수도 있다. 사용할 수 있는 수식자들은 많이 있으며, 이번 장의 나머지 부분에서는 그것들을 살펴볼 것이다.

7.7.1 long

long 수식자는 어떤 데이터 타입의 값에 대한 범위를 확장하기 위해 사용된다. 예를 들어, 정수 변수의 범위를 크게 하기 위해서 다음과 같이 수식자를 선언부 앞에 붙인다.

```
long int mylargeint;
```

long 수식자를 사용하여 증가될 데이터 타입의 범위는 시스템에 따라 다르다. 요즘에 나오는 많은 시스템들은 int와 long int 모두 같은 범위를 갖기 때문에 이런 수식자를 사용하는 것은 의미가 없다. long 수식자는 다음의 예제와 같이 double 데이터 타입에도 적용될 수 있다.

```
long double mydouble;
```

7.7.2 long long

long long 식별자는 확장형 long이라고 생각하는 편할 것이다. 다음 예제에서 int 데이터 타입의 경우, long long 수식자를 사용한 애플리케이션은 일반적으로 그 범위를 32비트에서 64비트로 변경할 것이다.

```
long long int mylargeint;
```

7.7.3 short

지금까지는 데이터 타입의 저장공간(즉, 값의 범위)을 증가시켜 주는 수식자들을 살펴보았다. short 수식자는 int 데이터 타입의 저장공간과 범위를 줄이는 데 사용된다. 이것은 정수의 범위를 16비트로 줄이는 데 효과적이며, 부호있는 값의 범위는 -32,768에서 32,767이 된다.

```
short int myshort;
```

7.7.4 signed/unsigned

기본적으로 정수는 부호가 있다고 가정한다. 다시 말해서 컴파일러는 정수 변수가 음수 또는 양수를 저장하기 위해서 호출될 것이라고 가정한다. 이것은 음수나 양수의 범위를 제한한다. 예를 들어, 32비트 int는 4,294,967,295의 범위를 갖지만, 실제로는 양수나 음

수가 가질 수 있는 범위는 –2,147,483,648에서 +2,147,483,647까지다. 만약 음수를 절대로 저장하지 않을 것이라는 것을 우리가 알고 있다면, unsigned로 선언하여 그 양수의 범위를 0에서 +4,294,967,295까지로 확장할 수 있다. unsigned int는 다음과 같이 지정된다.

```
unsigned int myint;
```

이 수식자들은 다른 수식자들과 같이 사용될 수도 있다. 다음의 예제처럼 unsigned short int를 선언할 수 있다.

```
unsigned short int myint = 10;
```

여기서 주목해야 할 점은 정수 값에 unsigned, signed, short, 그리고 long을 사용할 경우에는 int 키워드를 사용하지 않을 수도 있다는 것이다. 다음의 예제들은 모두 가능하다.

```
short myint;
long myint;
unsigned myint;
signed myint;
```

7.8 요약

데이터 타입은 모든 프로그래밍 언어에서의 기본적인 구성 요소이며, 오브젝티브–C도 예외는 아니다. 지금까지 이러한 기본에 대해 알아보았으니 다음 장으로 가서 변수의 사용에 대해 이야기해보자.