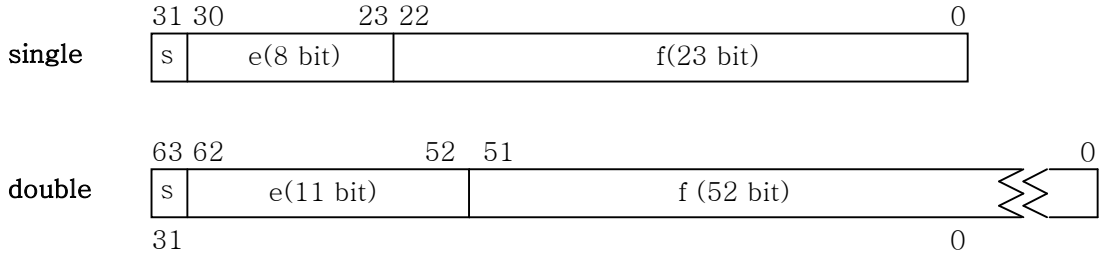
◈ 실수값 저장 방법(IEEE754 표준)

C언어에서 정수값을 저장하는 방법은 양수는 2진수 형태로 저장 되고 음수는 2의 보수 형태로 저장 됩니다. 그러나 정수형 자료형의 크기는 한계가 있으므로 아주 큰 수나 아주 작은 수는 표현할 수 없습니다. 또한 소수점을 포함하는 실수 값을 저장하는 데는 부적합할 것입니다. 따라서 한정된 기억공간에 이와 같은 값들을 효율적으로 저장하기 위해서 특별한 규칙을 정하였는데, 그것이 IEEE(아이트리플이) 754 표준입니다. C컴파일러는 이 표준에 따라 구현되었으므로 그 내용을 이해하면 프로그래밍 과정에서 수치 계산의 오류를 줄일 수 있고 보다 합리적인 자료형의 선택에 도움이 될 것입니다.

▷ 실수값을 표현하는 세 가지 형식

IEEE 754 표준은 실수 값을 저장하기 위해 4바이트, 8바이트, 16바이트 등의 저장형식을 정의 하였는데, 각각 single, double, quad라고 합니다. single형식은 4바이트 자료형으로 31번 비트는 부호를 나타내고, 23~30번까지 8개의 비트는 지수값을 저장하며, 나머지 23개 비트는 소수부분을 저장합니다. double형식은 전체크기가 8바이트 인데 지수부분을 11비트로 늘린 것이며, quad형식은 16바이트의 길이에 지수부분을 15비트 사용한 것입니다. 대게 8바이트 보다 큰 실수형은 quad의 형식으로 구현하기도 하지만 시스템에 따라 다를 수 있으므로, 여기서는 공통적인 형식인 single과 double만을 살펴봅니다.



s – 부호비트(sign), e – 지수부분(exponent), f – 소수부분(fraction)

C언어의 자료형 중에서 float는 single형태로, double은 double형식의 표준에 따라 구현 되었으며, long double의 경우는 시스템에 따라 여러 가지 형식으로 구현 되었습니다(터보C의 경우는 10바이트, 유닉스는 16바이트 등, VC++은 double과 같음).

▷ 실수값이 비트열로 저장되는 과정

이제 float형 변수에 실수값을 저장할 때 어떤 비트열로 저장 되는지 구체적으로 살펴봅시다. 소수점 형태로 된 수를 밑수를 이용해서 밑수의 거듭제곱 형태로 표현한 것을 과학적 표기법이라고 합니다. 예를 들어 1230000.0을 과학적 표기법으로 바꾸면 1230.0×10^3 또는 1.23×10^6 등과 같이 되고 소수점의 위치에 따라 얼마든지 다양한 표현 방법이 있을 것입니다. 과학적 표기법 중에 특히 소수점 앞에 0이 아닌 한자리 수만으로 나타낸 것을 정규화(normalization)라고 합니다. 즉, 1.23×10^6이 정규화된 표현법이 되겠지요.

모든 실수 값은 IEEE 754 표준에 의해 컴퓨터에 저장될 때 일단 2진수로 바꾸고 정규화합니다. 그리고 정규화한 표현법에서 다음과 같은 세 가지 정보를 추려냅니다.

- 부호
- 지수
- 유효숫자

예를 들어 6.5의 값이 float형 변수에 저장되는 과정을 살펴봅시다. 6.5를 이진수로 바꾸고 정규화하면 다음과 같습니다.

$6.5 \Rightarrow 110.1_2 \Rightarrow 1.101\times2^2$

여기서 부호는 양수, 지수는 2, 유효숫자는 1101입니다. 유효숫자는 무효의 0을 제외한 값입니다. 만약 십진수 00100.500의 값에서 왼쪽의 두 개의 0과 오른쪽의 두 개의 0은 의미가 없으므로 유효숫자는 1005 네 개가 될 것입니다.

이제 이 세 가지 정보에 따라 각 형식의 비트열을 채워주면 됩니다. 부호가 양수일 때는 부호비트에 0을, 음수일 때는 1을 저장합니다. 지수값 2는 지수부분에 저장을 하게 되는데 2에 127을 더해서 이진수 형태로 저장합니다. 이 때 더해주는 값을 바이어스(bias)라 하고 바이어스가 더해진 지수값을 biased exponent라고 합니다. single 형식의 지수부분 8비트에는 이 biased exponent가 이진수의 형태로 저장됩니다. 즉, 2+ 127=129의 값이 10000001의 형태로 저장이 되는 것이지요.

마지막으로 유효숫자는 1101중 맨 앞의 1을 제외한 나머지 부분, 즉 정규화 표현에서 소수부분(fractional part)만을 저장하게 됩니다. 이렇게 하는 이유는 정규화된 모든 이진수는 유효숫자가 반드시 1로 시작하게 되므로 저장할 때는 1을 제외시켜서 한정된 비트의 저장공간에 유효숫자를 하나라도 더 저장하기 위한 것입니다. 결국 실수값 6.5는 컴퓨터 내부에서

32비트 기억공간에 다음과 같이 저장될 것입니다.

(부호) (지수부분) (소수부분)
0 10000001 10100000000000000000000 (16진수로 0x40d00000)

▷ 바이어스는 왜 더하는가?

지수에 바이어스값을 더하는 이유는 음의 지수값을 양수로 표현하고자 하는 것입니다. 이렇게 하면 실수값을 대소비교 할 때 정수값처럼 비교할 수 있으므로 실행속도를 빠르게 할 수 있습니다. 예를 들어, 음수와 양수는 부호비트만을 검사하여 쉽게 대소를 구분할 수 있습니다. 그런데 두 수의 부호가 같다면 그 다음에는 지수가 큰 값이 더 큰 값이 될 것입니다. 예를 들어, 0.0062와 123.0의 대소비교를 생각해 봅시다. 일단 두 값을 이진수로 바꾸고 정규화하면 다음과 같습니다.

(십진수) (이진수) (정규화)
0.0062 => $0.0000000011001011\ldots_2$ => $1.1001011\ldots\times2^{-8}$
123.0 => 1111011.0_2 => 1.111011×2^{6}

여기서 0.0062의 지수값은 -8이고 123.0은 6이 됩니다. 이 두 지수값에 각각 바이어스 127을 더합니다.
0.0062의 경우 : -8 + 127 = 119
123.0의 경우 : 6 + 127 = 133

그리고 바이어스가 더해진 값을 8비트의 이진수로 바꾸면 다음과 같습니다.
119 => 01110111
133 => 10000101

결국 이 8비트가 single형식의 지수부분에 표현되는 것입니다.

0.0062 => 0 01110111 10010110010100101011111

123.0 => 0 10000101 11101100000000000000000

이렇게 바이어스가 더해진 지수의 비트열은 모두 양수이므로 가장 왼쪽의 비트부터 차례로 확인해 가면 대소를 쉽게 판단할 수 있습니다.

single 형식의 경우 바이어스를 127로 정한 이유는 single의 지수부분이 8비트이므로 256가지의 상태표현이 가능한데, 음수와 양수를 절반씩 나눠서 표현하기 위한 것입니다. 0의 지수가 포함되므로 바이어스를 127로 잡으면 양의 지수를 하나 더 표현할 수 있게 됩니다.

(지수)	(바이어스가 더해진 지수)	(8비트의 비트열)
-127	0	00000000
-126	1	00000001
-125	2	00000010
-124	3	00000011
-123	4	00000100
⋮		
0	127	01111111
1	128	10000000
2	129	10000001
3	130	10000010
⋮		
127	254	11111110
128	255	11111111

결국 바이어스가 127이면 8비트에 표현할 수 있는 지수의 범위는 -127~128이 될 것입니다.

▷ 비트열을 보고 저장된 값을 계산해 보자

반대로 IEEE 754 표준의 single에 의해서 표기된 16진수 0xbea00000는 출력할 때 어떤 실수값이 되는지 살펴봅시다.

16진수 0xbea00000 => 1 **01111101** 01000000000000000000000

- 부호비트는 1이므로 **음수**

- biased exponent가 125이므로 원래의 지수값은 125-127 = **-2**
- 소수부분이 01이므로 유효숫자는 **101**

이 세가지 값으로 다시 원래의 2진수 값을 계산하면 -0.0101이 되고 10진수로 바꾸면 -0.3125가 될 것입니다. 간단하게(?) 식으로 표현하면 다음과 같습니다.

$$N = (-1)^s \times 2^{e-127} \times 1.f = (-1)^1 \times 2^{-2} \times 1.01 = -0.0101_2 = -0.3125$$

(s는 지수 비트, e는 바이어스가 더해진 10진수 지수값, f는 소수부분(fractional part)의 이진 비트열)

IEEE 754 표준의 double 형식도 같은 방법으로 저장되며 단지 지수부분과 소수부분을 표현하는 비트수가 늘어남으로서 값의 크기와 유효숫자의 표현범위가 늘어난 것 뿐입니다. 예를 들어, double 형식의 바이어스는 1023인데 이것은 지수부분이 11비트 이므로 반으로 나눈 것에서 1을 뺀 값입니다. 결국 double형식은 지수값을 -1023부터 1024까지 표현할 수 있겠지요(주의할 것은 2진수의 지수값입니다. 10진수로 바꾸면 훨씬 적어집니다. 그래도 계산해 보면 어마어마합니다^^).

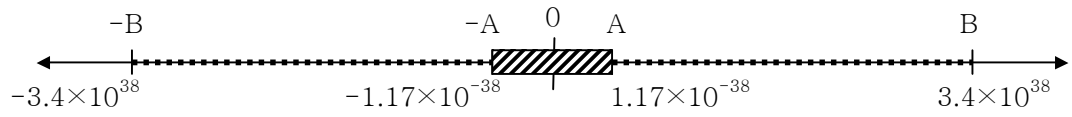
▷ 실수를 표현할 수 있는 값의 범위와 유효숫자

이제 single과 double의 형식으로 표현할 수 있는 값의 범위를 살펴봅시다. 우선 부호비트를 제외한 절대값의 범위를 계산한 후에 음수값의 범위를 더하면 전체 범위를 알 수 있을 것입니다.

single 형식에서 표현 가능한 지수의 범위는 -127~128까지 입니다. 이 중에서 -127과 128은 특별한 용도로 쓰이므로 이들을 제외하면 결국 지수값 –126~127까지 표현할 수 있습니다. 따라서 최소값은 지수가 –126이고 소수부분이 전부 0일 때입니다. 최대값은 지수가 127이고 소수부분이 전부 1인 값을 가질 때입니다. 이 값들을 위의 식에 따라 구해보면 $2^{-126} \times 1.0 \sim 2^{127} \times (2-\varepsilon)$의 범위가 됩니다. 여기서 2-ε은 1.11111111111111111111111이며 10진수로 계산하면 2에서 아주 작은 값을 뺀 것을 의미합니다. single 형식에서 소수부분의 23비트가 전부 1이라면 그 값은 십진수로 계산하면 거의 1에 가까운 수가 될 것이고, 이 때 1과의 차이가 나는 값을 ε(입실론)으로 표현한 것입니다. 물론 double 형식과 같이 소수부분의 비트수가 늘어나면 ε값은 더욱 작아질 것이고 유효숫자는 늘어날 것입니다.

결국 single형식에서 표현할 수 있는 값의 범위를 10진수 바꾸면 $1.17549435 \times 10^{-38} \sim 3.40282347 \times 10^{38}$까지의 값이 됩니다. 여기에 부호비트가 음수이면 범위는 반대로 되므로 음수값의 표현범위는 $-3.40282347 \times 10^{38} \sim -1.17549435 \times 10^{-38}$ 가 될 것입니다. 이 실수

값들의 표현 범위를 수직선 상에 나타내면 다음과 같습니다.



이 수직선상에서 보면 single 형식에 의해 표현되는 값은 결국 –B~-A, A~B 사이의 값이 됨을 알 수 있습니다. 그런데 컴퓨터의 기억공간은 한정되어 있으므로 이 사이의 실수값 모두를 표현할 수는 없습니다. 따라서 점선과 같이 사이 사이의 값들 만을 표현할 수 있는 것이고 single 형식의 경우 소수부분이 23비트 이므로 유효숫자가 2^{-23} 보다 작은 정밀도의 값은 나타낼 수 없습니다.

좀더 쉽게 생각해 보면 만약에 single 형식의 소수부분이 2비트라고 하면 표현할 수 있는 10진수는 다음과 같이 제한적입니다.

(10진수 값)	(소수부분 비트열)
1	00
1.25	01
1.5	10
1.75	11

결국 1.25의 값이나 1.3의 값은 모두 소수부분이 같은 비트열로 저장되게 되므로 0.25 만큼의 오차가 생기는 것입니다. 이것을 입실론 오차(epsilon error)라고 하는 것이지요. 이 오차 때문에 표현할 수 있는 유효숫자가 제한되는 것입니다. 2^{-23} 은 0.0000001192의 값이 되므로 따라서 10진수로 바꾸었을 때 정수자리까지 포함해서 7자리 정도는 오차 없이 표현할 수 있게 되는 것입니다.

그런데 지금까지의 single 형식에서 보면, 0을 포함해서 –A에서 A 사이의 수는 표현 할 수 없음을 알 수 있습니다. 이 사이의 수를 subnormal number라고 하며, 우리가 아껴두었던 지수값 -127을 사용하여 표현합니다. 즉, 바이어스를 더해서 0이 될 때(지수부분의 비트열이 전부 0일 때)는 소수부분의 비트열을 1을 더하지 않고 소수 그 자체로 계산함으로서 최소값과 0사이에 표현하지 못했던 수를 표현하는 것입니다.

$$N = (-1)^s \times 2^{-126} \times 0.f$$

(1.f가 아니라 0.f이다! 2^{-126}은 수직선 상에서 최소값(A위치))

여기서 물론 소수부분이 0이면 전체 비트열이 전부 0이 되며 0값을 나타내게 됩니다.

결론적으로 subnormal number값을 포함하면 float형의 표현 범위는 $-3.4\times10^{38} \sim 3.4\times10^{38}$ 까지가 되는 것입니다.

double형의 표현 범위는 지수부분을 11비트, 소수부분을 52비트로 잡고 같은 방식으로 계산하면 대략 $-1.79\times10^{308} \sim 1.79\times10^{308}$ 까지의 값이 나옵니다. 유효숫자는 10진수로 float는 7자리, double이 15자리까지 표현할 수 있습니다.

마지막으로 single 형식에서 표현하지 않았던 지수값 128, 즉, biased exponent의 비트열이 전부 1인 경우는 무한대의 값이나 무리수를 표현하기 위해 사용합니다. 소수부분이 전부 0이고 부호비트가 0이면 양의 무한대이고, 부호비트가 1이면 음의 무한대가 됩니다.

```
0 11111111 00000000000000000000000   => 양의 무한대
1 11111111 00000000000000000000000   => 음의 무한대
그 외 지수부분이 전부 1인 경우              => 무리수 표현
```