## 1.1 컴퓨터에 수가 기억되는 방법

**이 절의 목적**
컴퓨터에 수를 기억시키는 방법과 표현하는 방법을 알아본다.

**고정소수점**
정수를 표현하는 방법으로 한 비트는 부호를 나머지는 숫자를 표현한다.

**부동소수점**
소수를 표현하는 방법

$$x = (0.a_1a_2a_3\cdots a_r)_b \times b^c,\ 0 \le a_i < b,\ m \le c \le M$$

**일배정도, 이배정도**
32비트를 한 단어로 사용, 64비트를 한 단어로 사용

컴퓨터는 전기를 이용하여 움직이는 기계이므로 전기가 켜질 때와 꺼질 때의 두 가지 현상을 이용하여 수를 표현할 수 있다. 컴퓨터에 있는 하나의 기억소자(memory chip)는 전기가 나갈 때와 들어올 때 두 가지를 표현할 수 있으므로 이것을 수에 비교하면 0과 1을 대신할 수 있다. 이러한 기억소자 여러 개를 묶어 하나로 사용하면 더욱 다양한 수를 표현할 수 있다. 예를 들어, 두 개의 소자로 표현할 수 있는 현상은 $00, 01, 10, 11$으로 이것을 이용하여 나타낼 수 있는 수는 모두 4개이다. 일반적으로 많이 사용하는 개인용 컴퓨터는 32개의 소자를 사용하는 데 이것을 32비트 기억소자라 한다. 이 경우 얼마나 많은 수를 어떻게 표현하는 지 알아보자.

### 수의 표현방법

그림 1.1과 같이 한 단어를 이루고 있는 32비트 기억소자의 0번 비트는 수의 부호를 나타내고 1번 비트부터 31번 비트까지 31개의 기억소자가 수를 나타낸다. 이것은 최대 $2^{32} = 4294967296$개의 수를 표현할 수 있고 이중 2147483647개는 양수를 2147483648개는 음수를 나머지 하나는 0을 표현하여 표현 가능한 정수는 $-2147483648$부터 2147483647까지 모두 4294967296개이다. 항상 31번 비트 다음에 소수점이 고정되어 있는 것으로 보아 이를 **고정소수점**(fixed point) 표현이라고 한다.

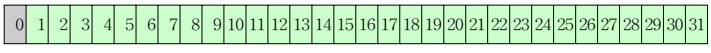


그림 1.1 32비트 기억소자

소수를 표현하는 데는 다음과 같은 **부동소수점**(floating point) 표현을 사용한다.

$$x = (0.a_1a_2a_3\cdots a_r)_b \times b^c,\ 0 \le a_i < b,\ m \le c \le M \qquad (1.1)$$

여기서 $0.a_1a_2a_3\cdots a_r$을 **가수**(mantissa), $b$를 **기저**(base), $c$를 **지수**(exponent)라고 하며 상수 $m$과 $M$은 컴퓨터 기종에 따라 다르다. 특히 $a_1$이 0이 아니면 이 표현은 **정규화**(normalized)되었다고 한다. 식 (1.1)과 같은 표현을 사용하는 경우 $b$**-진법**을 사용한다고 한다.

### 일배정도의 최대(소)실수

32비트를 한 단위로 사용하는 컴퓨터에서 소수의 표현 방법을 알아보자. 그림 1.1에서 0번 비트는 부호를 나타내며 1번 비트부터 7번 비트까지는 지수를, 8번 비트부터 31번 비트까지 24개 비트는 가수를 나타낸다. 소수점은 7번과 8번 비트 사이에 있는 것으로 생각한다. 지수를 나타내는 7개의 비트가 표현할 수 있는 수는 $2^7 = 128$개로 0을 포함하여 127까지이며 이중 64개는 음수를 나타내고 나머지 64개는 0과 양수를 나타낸다. 따라서 큰 수는 양수로 최대 63, 작은 수는 음수로 최소 $-64$이다. 이 경우 $m = -64$, $M = 63$, $b = 16$이 된다. 가수를 표현할 수 있는 비트는 모두 24개이다. 4비트를 수의 한 자리로 사용하면 한 자리가 나타낼 수 있는 수는 $2^4 = 16$으로 0부터 15까지 모두 16개의 수를 표현할 수 있고 자리 수는 $r = 6$이 되어 다음과 같이 나타난다.

$$x = (0.a_1a_2a_3a_4a_5a_6)_{16} \times 16^c,\ 0 \le a_i < 16,\ -64 \le c \le 63 \qquad (1.2)$$

단, $a_1 \neq 0$이다. 식 (1.2)와 같은 표현은 16진법을 사용한다고 한다. 가수가 표현할 수 있는 최대의 수는 16진법으로 $0.FFFFFF(\approx 1)$이므로( $F$는 15를 뜻함) 이 컴퓨터가 나타낼 수 있는 최대의 실수는

$$(0.FFFFFF)_{16} \times 16^{63} \approx 16^{63} \approx 0.72 \times 10^{76}$$

이고, 가수가 표현할 수 있는 최소의 16진법 수는 $(0.100000)_{16} = 16^{-1}$이므로 최소의 양의 실수는

$$(0.100000)_{16} \times 16^{-64} = 16^{-65} \approx 0.54 \times 10^{-78}$$

가 된다. 절대값이 최대실수보다 크면(overflow) 올바른 연산을 할 수 없고 최소실수보다 작으면(underflow) 일반적으로 0으로 처리하여 계산한다. 이와 같은 컴퓨터에서 32비트를 한 단어로 사용하면 **일배정도**(single precision), 64비트를 한 단어로 사용하면 **이배정도**(double precision)의 표현이라 한다.

## 1.2 마무리오차

**이 절의 목적**
자리수의 제한으로 생기는 컴퓨터에 기억시키는 수의 오차에 대하여 알아본다.

**마무리오차**
컴퓨터에 표현되는 수와 실제의 수의 차이로 생기는 오차

**버림**
표현되는 자리수를 넘는 자리의 숫자를 모두 무시하는 방법,

**반올림**
표현되는 자리수를 넘는 경우 다음 자리의 수가 사용되는 진수의 반 이상이면 직전 자리의 수에 1을 더하는 방법

컴퓨터는 유한개의 기억소자를 사용하여 수를 표현하므로 연산 과정 중에 생긴 수는 그에 맞는 자리 수만 기억된다. 따라서 컴퓨터에 표현되는 수와 실제의 수와는 차이가 있을 수 있는데 이 차이를 **마무리오차**(rounding-off error)라 한다. 예를 들어, 네 자리만 기억되면 수 12.345는 12.35가 되든지 아니면 12.34로 전환해야 하므로 마무리오차 $0.005$ 또는 $-0.005$가 발생된다. 16진법과 6자리 가수로 표현되는 컴퓨터에서 실수 $x$를 기억시키는 방법은

$$\bar{x} = (0.a_1 a_2 a_3 a_4 a_5 a_6)_{16} \times 16^c, \quad 0 \le a_i < 16 \qquad (1.2)$$

임을 알았다. 이 경우 7개 이상의 자리는 표현할 수가 없으므로 컴퓨터에서 $x$를 표현하는 수 $\bar{x}$와 참값 $x$는 차이가 날 수 있다. 7개 이상의 자리가 있는 수를 6자리로 기억시키는 데는 두 가지 방법이 있다. 하나는 7번 째 자리 이하를 모두 버리는 **버림**(chopping)이고 다른 하나는 7번 째 자리가 8(10진법의 경우는 5)보다 작으면 버리고, 그 이상이면 6번 째 자리에 1을 더해주는 **반올림**(symmetric rounding)이다.

**예제 1.1** 가수가 6자리이고 16진법인 컴퓨터에서 $x = 0.1234568 \times 16^3$를 나타내는 수는 다음과 같다.

(1) $\bar{x} = 0.123456 \times 16^3$ 버림

(2) $\bar{x} = 0.123457 \times 16^3$ 반올림

컴퓨터에서 더하는 순서를 바꾸면 마무리오차의 영향으로 그 결과가 달라질 수 있다. 예를 들면, 가수가 5자리이고 10진법을 사용하는 컴퓨터에서 반올림을 사용할 때 0.12345에 0.000002를 더하면 0.123452는 반올림하여 다시 0.12345가 되어

$$\begin{aligned} & (((0.12345+0.000002)+0.000002)+0.000002) \\ = \ & ((0.12345+0.000002)+0.000002) \\ = \ & (0.12345+0.000002) \\ = \ & 0.12345 \end{aligned}$$

이 되나, 더하는 순서를 바꾸어 작은 수끼리 먼저 더하면

$$\begin{aligned} & 0.12345+(0.000002+(0.000002+0.000002)) \\ = \ & 0.12345+(0.000002+0.000004) \\ = \ & 0.12345+0.000006 \\ = \ & 0.12346 \end{aligned}$$

이 되어 서로 같지 않다. 이것은 0.000006을 0.12345에 더하면 0.123456는 반올림되어 0.12346이 되기 때문이다. 덧셈을 할 때는 비슷한 크기의 수끼리 먼저 합하여 그 결과들을 다시 합하는 것이 바람직하다.

### 알고리즘 1.1 연속적인 수의 덧셈

| | |
|---|---|
| **목적** | $a_1+\cdots+a_n$ |
| **입력** | $n, a_i$ |
| **출력** | $sum$ |
| | sum=0.0<br>for $i=1$ to $n$<br>sum = sum + $a_i$<br>end |

### 프로그램 1.1 rounding_off.cpp

**프로그램예제 1.1** 알고리즘 1.1을 이용하여 $1.0$에 $1.0\times10^{-6}$을 $10^4$번 더하는 프로그램을 일배정도와 이배정도를 사용하여 만든 rounding_off.cpp로 결과를 비교해 보자. 단, 더하는 순서를

$$1.0+10^{-6}+\cdots+10^{-6}\text{과}\quad 1.0+(10^{-6}+\cdots+10^{-6})$$

으로 하여 각각을 비교해 보자.

**실행.** rounding_off.cpp을 실행하면 다음과 같이 출력된다.

```
Sum1 = 1.009537, Sum0 = 1.010000
dSum1 = 1.010000, dSum0 = 1.010000
```

이 합은 1.01이 되어야 한다. 그러나 1에 상대적으로 아주 작은 수를 계속 더한 Sum1은 마무리오차가 누적되어 1.01과 다름을 알 수 있다. 그러나 작은 수를 먼저 더하고 나중에 1을 더한 Sum0는 정확하다. 한편 이배정도를 선언하면 결과는 더 정확해져 1에 상대적으로 아주 작은 수를 계속 더한 dSum1도 정확하게 구해졌다.

## 1.3 Taylor 전개와 절단오차

**이 절의 목적**

컴퓨터의 유한성으로 인해 발생되는 오차와 오차를 측정하는 $O(\cdot)$에 대하여 알아본다.

**절단오차**

무한합의 수식에서 유한개의 합을 택할 때 발생되는 오는 오차

$O(h)$

$h$가 0으로 수렴하는 정도를 나타냄

무리수 $e$의 값은 얼마인가? 계산기(Casio fx-115)을 사용하니 2.718281828이 나온다. 이 수는 물론 정확한 수가 아니라 $e$의 근사값이다. 이 값을 살펴보기 위해 적당한 수 $x$에서 $e^x$의 값을 구하여 보자. $e^x$의 근사값은 Taylor 전개 $e^x = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + \cdots$을 이용하면 사칙연산을 통하여 계산할 수 있다. 이 무한합을 계산할 수 없으므로 이 중 유한합 $1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!}$을 이 값으로 대신한다. 이 때 오차가 발생하며 이와 같이 수식의 계산에서 무한합을 계산할 수 없어 유한개의 합을 그 근사값으로 하여 발생되는 오차를 **절단오차**(truncation error)라고 한다.

Taylor 정리에 의하면 함수 $f$가 $x = a$에서 계속 미분가능할 때 $x = a$을 중심으로 함수 $f(x)$의 값은 다음과 같이 표현할 수 있다.

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''(a) + \cdots + \frac{(x-a)^n}{n!}f^{(n)}(a) + \cdots$$

이중 처음 유한개 항의 합

$$f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''(a) + \cdots + \frac{(x-a)^n}{n!}f^{(n)}(a)$$

을 $f(x)$의 $x = a$에서 $n$**차 Taylor 전개식**이라 한다.

**예제 1.2** Taylor 전개를 이용하여 $f(x)=e^x$의 $a=0$에서의 전개식을 구하고 $f(1)=e$의 근사값을 $n=2,4,6,8,10$일 때 구하여 보자.

**풀이.** $f(0)=f'(0)=f''(0)=\cdots=f^{(n)}(0)=\cdots=1$이므로

$$e^x = 1+x+\frac{x^2}{2!}+\cdots+\frac{x^n}{n!}+\cdots$$
$$\approx 1+x+\frac{x^2}{2!}+\cdots+\frac{x^n}{n!}$$

이다. 계산기를 사용하여 다음과 같은 표를 얻는다. 참고로 계산기에서 보여주는 $e$의 값은 2.718281828이다.

| n | 2 | 4 | 6 | 8 | 10 |
|---|---|---|---|---|---|
| 근사값 | 2.5 | 2.708333333 | 2.718055555 | 2.718278769 | 2.7182818 |

**알고리즘 1.2 Taylor 전개식**

| | |
|---|---|
| 목적 | $\sum_{k=0}^{n}\frac{(x-a)^k}{k!}f^{(k)}(a)$의 $x=x_0$에서 함수값 |
| 입력 | $n, x_0, a, f(x), f'(x), f''(x), \cdots, f^{(n)}(x)$ |
| 출력 | sum |
| | sum=0.0<br>for $k=1$ to $n$<br>$\text{sum} = \text{sum} + \frac{(x_0-a)^k}{k!}f^{(k)}(a)$<br>end |

**프로그램 1.2** taylor_exp.cpp

**프로그램예제 1.2** 알고리즘 1.2를 이용하여 함수 $f(x)=e^x$의 $x=a$에서 $n$차 Taylor 전개식으로 $f(x_0)$의 근사값을 계산하는 taylor_exp.cpp로 예제 1.2을 확인해 보라.

**실행.** taylor_exp.cpp을 실행하고 해당되는 입력을 주면 다음과 같이 출력된다.

```
차수 n은? 10
a와 x의 값은? 0 1
n=10        Exp(1.000000) = 2.718282
```

$n$차 Taylor 전개식을 근사식으로 사용할 경우 생긴 절단오차는 다음과 같이 **큰 O()**로 나타낸다.

$$\begin{aligned} f(x) &= f(a)+(x-a)f'(a)+\frac{(x-a)^2}{2!}f''(a)+\cdots+\frac{(x-a)^n}{n!}f^{(n)}(a)+\cdots \\ &= f(a)+(x-a)f'(a)+\cdots+\frac{(x-a)^n}{n!}f^{(n)}(a)+O((x-a)^{n+1}) \end{aligned}$$

$n=1$과 $n=2$인 경우 각각의 Taylor 전개식을 함수 $f(x)$의 $x=a$에서 **일차근사식, 이차근사식**이라 하며 다음의 오른쪽 항과 같이 나타낸다.

$$\begin{aligned} f(x) &\approx f(a)+(x-a)f'(a) \\ &\approx f(a)+(x-a)f'(a)+\frac{(x-a)^2}{2}f''(x-a) \end{aligned}$$

각각의 경우 절단오차는 $O((x-a)^2), O((x-a)^3)$이 된다. $f(x)=g(x)+O((x-a)^2)$의 의미는 $x\to a$일 때 $(x-a)^2$이나 $f(x)-g(x)$가 0으로 수렴하는 정도가 비슷함을 말한다. 즉,

$$\lim_{x\to a}\frac{f(x)-g(x)}{(x-a)^2}=\text{상수}$$

**예제 1.3** $f(x)=\ln(x+1)$의 $a=0$에서 일차근사식과 이차근사식을 구하고 $x=1, 0.5, 0.1, 0.05, 0.01$에서 그 값을 비교해 보자.

**풀이.**

$$f(0)=\ln 1=0,\ f'(x)=\frac{1}{x+1}\text{ 이므로 }\ f'(0)=1$$

$$f''(x)=-\frac{1}{(x+1)^2}\text{ 이므로 }\ f''(0)=-1$$

따라서 $\ln(x+1)$의 $a=0$에서 일차식은 $x$, 이차식은 $x-\frac{x^2}{2}$이다. 이 근사식을 이용하여 $x=1, 0.5, 0.1, 0.05, 0.01$에서 근사값을 구하면 다음과 같고 그래프는 그림 1.2와 같다.

| 식 \ $x$ | 1 | 0.5 | 0.1 | 0.05 | 0.01 |
|---|---|---|---|---|---|
| **일차근사식** | 1 | 0.5 | 0.1 | 0.05 | 0.01 |
| **이차근사식** | 0.5 | 0.375 | 0.095 | 0.04875 | 0.00995 |
| **참값** | 0.69314718 | 0.405465108 | 0.095310179 | 0.048790164 | 0.009503308 |

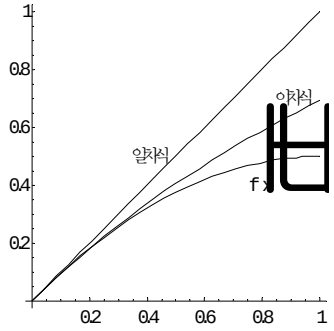


그림 1.2 일차, 이차근사식

## 1.4 다양한 오차의 정의

**이 절의 목적**
상대오차, 절대오차, 오차의 한계 그리고 기계오차에 대하여 알아본다.

**절대오차, 상대오차**
참값과 근사값의 차이, 참값에 대한 절대오차의 비

**오차의 한계**
연산 중에 발생되는 오차의 최대치

**기계오차**
$1+\varepsilon$을 컴퓨터에서 표현하는 수가 $1$보다 크게되는 최소의 양의 실수 $\varepsilon$

컴퓨터에서 수를 저장하는 기억소자의 유한성으로 인하여 수를 표현할 때 필연적으로 오차가 생기는 이유를 설명하였다. 어느 수의 참값이 $x$이고 이 값을 컴퓨터에서 나타내는 수를 $\bar{x}$ 라고 하자. 이 때 $|x-\bar{x}|$을 **절대오차**(absolute error)라 하고 참값이 0이 아닌 경우 $\frac{|x-\bar{x}|}{|x|}$ 을 **상대오차**(relative error)라 한다. 연산과정 중에 발생되는 오차의 최대치를 그 **오차의 한계**(error bound)라고 한다. 10진법에서 소수점 첫 째 자리에서 버림하여 정수를 얻는 경우 절대오차의 한계는 1이 된다. 예를 들어 참값 1.999는 소수점 첫 째 자리에서 버림하면 1이 되기 때문에 그 오차 $|1.999-1|$는 1을 넘을 수 없다. 그러나 반올림의 경우 절대오차의 한계는 0.5가 되어 버림의 경우보다 오차가 반이나 더 작다. 컴퓨터의 반복성을 활용하여 문제를 풀 수 있도록 변형된 **알고리즘**(algorithm)은 발생되는 오차의 한계가 작고 실행속도가 빠르며 기억용량을 가능한 적게 만드는 것이 좋다.

컴퓨터에서 아주 작은 양수 $\varepsilon$에 대하여 $1+\varepsilon$는 1로 저장될 수 있다. 이와 같은 경우 함수 $f(x)$의 $x=1$에서 미분값을 계산할 때 그 수 $\varepsilon$ 보다 작은 모든 양수 $\delta$에 대하여 $1+\delta=1$이 되어 근사값 $\frac{f(1+\delta)-f(1)}{\delta}$ 은 항상 0이 되는 잘못된 결과를 낳게된다. $1+\varepsilon$의 컴퓨터에서 표현하는 수 $\overline{1+\varepsilon}$ 이 $1$보다 크게되는 최소의 양의 실수 $\varepsilon$을 **기계오차**(machine epsilon)라 한다. 즉, 기계오차 $=\min\{\varepsilon>0 \mid \overline{1+\varepsilon}>1\}$.

**예제 1.4** 가수가 5자리이고 16진법을 이용하는 컴퓨터에서 기계오차를 구하여 보자.

**풀이.** 이 컴퓨터에서 $1+16^{-4}=(0.10001)_{16}\times 16>1$이므로 기계오차 $\varepsilon$는 $16^{-4}$보다 클 수 없다. 버림의 경우 그 다음의 작은 수 $0.0000FFF\cdots$는 $\overline{1+0.0000FFF\cdots}=1$이므로 기계오차는 $16^{-4}$이다. 그러나 반올림의 경우 기계오차는 $0.5\times 16^{-4}$이다.

$1+\frac{1}{2^n}$에서 $n$이 점점 커지면 오차로 인하여 $1+\frac{1}{2^n}=1$이 되는 경우가 생긴다. 이 등식이 성립하기 직전의 $\frac{1}{2^n}$ 또는 $1+\frac{1}{2^n}>1$이 되는 최소의 수 $\frac{1}{2^n}$을 구하여 기계오차를 추정해 보자.

**알고리즘 1.3 기계오차**

| | |
|---|---|
| **목적** | 기계오차 계산 |
| **출력** | meps |
| | body=2; test=1<br>while body>1 do<br>  test=test/2; body=1+test<br>  if body>1 then meps=test<br>end |

**프로그램 1.3** machine_eps.cpp

**프로그램예제 1.3** 알고리즘 1.3을 이용하여 일배정도와 이배정도를 사용할 때 학생이 사용하는 컴퓨터의 기계오차를 machine_eps.cpp을 이용하여 구하여 보자.

**실행.** machine_eps.cpp을 실행하면 다음이 출력된다.

Machine epsilon =1.19209290e-07

(double을 사용하면 Machine epsilon = 2.22044605e-16 )

meps는 1+test=1이 되기 직전의 test이다. 즉, 이 계산과정에서 1+test>1이 되는 마지막 test이다. 이 값은 컴퓨터 기종마다 또는 컴파일러마다 달라질 수 있다.