C 言語 010 誤差について

【小数と誤差について】

コンピュータは「整数の演算」は得意だが「小数の演算」は苦手です。

コンピュータで小数を取り扱う場合「誤差」に注意しましょう。

＜丸め誤差＞

人間は数値を 10 進数で扱いますが、コンピュータは 2 進数で扱います。よって、10 進数と 2 進数の変換が随時行われることになります。

10 進数の整数を 2 進数に変換する場合は問題ありませんが、10 進数の小数を 2 進数に変換する場合、誤差が生じる可能性があります。

これは、10 進数から 2 進数に変換する際に循環小数（割り算で割り切れなくなり同じ数値が続くこと）が発生し、強制的に丸められてしまうことがあるからです。

これによって生じる誤差を「丸め誤差」と称します。

＜例文 1＞

```
if (0.1 + 0.2 == 0.3) {
    printf("真");
}
else {
    printf("偽");
}
```

出力結果

```
偽
```

「0.1 + 0.2」と「0.3」が等しくないとなっています。

＜例文 2＞

```
/* 小数 20 桁出力する */
printf("%.20f¥n", 0.1);
printf("%.20f¥n", 0.2);
printf("%.20f¥n", 0.3);
```

出力結果

```
0.10000000000000001000
0.20000000000000001000
0.29999999999999999000
```

10 進数の「0.1」「0.2」「0.3」は、丸め誤差によりコンピュータ上では正確な数値にならず近似値になっています。それゆえ、例文 1 では真にならなかったのです。

＜例文 3＞	出力結果

/* 0.1 を 1000 回足し算する */
double var = 0;
int i;
for (i = 0; i < 1000; i++) {
　　var = var + 0.1;
}
printf("%.20f¥n", var);
	99.99999999999859300000

「0.1 * 1000 = 100」にならず、演算結果に誤差が生じています。

＜例文 4＞	出力結果

if (0.1F == 0.1) {
　　printf("真");
}
else {
　　printf("偽");
}
	偽

float 型の 0.1 と double 型の 0.1 は等しくありません。

＜例文 5＞	出力結果
printf("%.20f¥n", 0.1F); printf("%.20f¥n", 0.1);	0.10000000149011612000 0.10000000000000001000

float 型と double 型では有効桁数が異なり、0.1 の近似値が異なっているからです。

【signed と unsigned の比較】

＜例文 1＞	出力結果

if (-10 > 10U) {
　printf("真");
}
else {
　printf("偽");
}
	真

「signed long int の-10」と「unsigned long int の 10」を比較すると「signed long int の-10」の方が大きいとなっています。
これは signed と unsigned を比較した場合、signed は unsigned に変換される仕様だからです。

単純に 1 バイトで説明すると「unsigned long int の 10」は 2 進数では「00001010」、「signed long int の-10」は、最初の 1 ビットが符号になるので、2 進数（2 の補数）では「11110110」になり、これを unsigned で読むと 2 進数「11110110」は 10 進数「246」になります。

よって「signed long int の-10」は「unsigned long int の 246」に変換され、「unsigned long int の 10」より大きいとなったのです。
（これは long が 1 バイトだった場合での話です。本来 long 型は 4 バイトです）

負数が含まれる場合、安直に signed と unsigned を比較してはいけません。