プログラミング演習1

― 課題4の付録1 ―

プログラムで間違いやすい分数式

プログラムでは分数式のまま記述することができないので、割り算で代用することになる。

たとえば、 $\frac{a}{b}$ は a / b でよいが、

$\frac{a + b}{c}$ は a + b / c ではなく、(a + b) / c

$\frac{a}{b + c}$ は a / b + c ではなく、a / (b + c)

$\frac{a}{b \cdot c}$ は a / b * c ではなく、a / (b * c)

と分子や分母をカッコで括らなくてはいけない。

また、整数どうしの割り算は整数除算を行うので、実数除算にするためには 分子か分母を実数化する必要がある。
整数値を実数化するには小数点をつければよい。

```
1/2    →  0    // 整数除算
1/2.   →  0.5  // 分子を実数化
1./2   →  0.5  // 分母を実数化
1./2.  →  0.5  // 分子・分母を実数化
```

整数変数を実数化するにはキャスト演算子 (double) を用いて型変換を行う。

```
int a, b; double x;
x = (double)(a / b);        // 整数除算後に実数化（意味がない）
x = (double)a / b;          // 分子を実数化
x = a / (double)b;          // 分母を実数化
x = (double)a / (double)b;  // 分子・分母を実数化
```

破局のbreak文、未練のcontinue文

case文のときにbreak文が出てきたが、**break**文のもう一つの使い方に ループ(反復)からの脱出がある。break文に出会うと、プログラムは その文を含む最小のループ(for, while, do～while)から抜け出す。if文と組み合わせることで、ループ内の任意の場所から条件付きで抜け出すことができる。

```
do {
    n++;
    …(1)…
    if (n > N) {
        printf("*** 反復回数が%d回を超えました。 ***¥n", N);
        break;
    }
    …(2)…
} while (err >= eps);
```

break文とよく似たものに**continue**文がある。continue文はループの中でのみ用いられ、この文以降の文を省略してループの最後までジャンプする。ループの外には出ないことがbreak文との違いである。

continue文が出てくるシチュエーションでは、if文を使えば代用できる。また、break文が出てくるシチュエーションでは、continue文とループでの条件文の追加で 代用できないことはないが、同種の条件式を2か所で使うことになる。

```
do {
    n++;
    …(1)…
    if (n > N) {
        printf("*** 反復回数が%d回を超えました。 ***¥n", N);
    } else {
        …(2)…
    }
} while ((err >= eps) && (n <= N));
```

多重ループを一気に抜けるには**goto**文が便利である。また、関数(メイン関数の場合にはプログラム)を強制終了させるには **return** 文、サブルーチン内からプログラムを強制終了させるには**exit**関数を用いる。

scanf関数は、お薦めできない！？

scanf関数はいろいろと問題が多い(例題で学ぶC言語 p.121 参照)ので、代わりにfgets関数とsscanf関数の組み合わせをお薦めする。通常は

```
int main(void)
{
    double x1, x2, eps;

    …

    printf(" x1, x2, eps = ");
    scanf("%lf%lf%lf", &x1, &x2, &eps);

    …
}
```

と書くところをfgets関数とsscanf関数で置き換えると…

```
int main(void)
{
    double x1, x2, eps;
    char s[128];

    …

    printf(" x1, x2, eps = "); fgets(s, 128, stdin);
    sscanf(s, "%lf%lf%lf", &x1, &x2, &eps);

    …
}
```

デリミタ(数字の区切り)としてSpace, Tab以外にコンマも許すようにし、ついでにエラー処理したいときは、

```
int main(void)
{
    double x1, x2, eps;
    char s[128];

    …

    printf(" x1, x2, eps = "); fgets(s, 128, stdin);
    if (sscanf(s, "%lf%lf%lf", &x1, &x2, &eps) != 3)            // スペース、タブ
        if (sscanf(s, "%lf,%lf,%lf", &x1, &x2, &eps) != 3) {    // コンマ
            printf("*** 入力形式が異常です。 ***¥n"); return 1;
        }

    …
}
```

main関数の型と戻り値

main関数は、プログラム内では戻る先がないので、型は決めなくても プログラムの動作自体にはまったく影響がない。大昔は void (不完全型) としているソースが多かった。

```
1: #include <stdio.h>
2:
3: void main()
4: {
5:     printf("hello, world!¥n");
6: }
```

しかし、ANSI準拠のCコンパイラでは、デフォルトでint型となっており、厳しめのコンパイラチェックをかけると、警告として戻り値を要求される。実は、main関数の戻り値はOSに引き渡されるので、OS側で正常終了か異常終了かを 調べるのに利用されることが多い。したがって、積極的に戻り値を返す癖をつけるようにしてほしい。終了状態を返す整数型関数の戻り値としては、正常終了時に 0、異常終了時に 0 以外を返すのが慣例である。

```
1: #include <stdio.h>
2:
3: int main(void)
4: {
5:     printf("hello, world!¥n");
6:     return 0;
7: }
```

ちなみに、stdlib.h をインクルードしているときには、戻り値用の定数マクロとして EXIT_SUCCESS, EXIT_FAILURE が使用できる。

```
1: #include <stdio.h>
2: #include <stdlib.h>
3:
4: int main(void)
5: {
```

```
6:     printf("hello, world!¥n");
7:     return EXIT_SUCCESS;
8: }
```

ごく短いプログラムでは野暮な感じがするが、可搬性が良くなるので、プロのプログラマをめざすなら知っていて損はない。

サブルーチンからの強制終了

main関数から OSへの戻り値は return文で指定するが、サブルーチンからエラーなどで強制終了するときには exit関数を用い、そのときの戻り値は exit関数の引数に入れる。

デバッグに便利な定数マクロ

Cでは、標準で __FILE__, __LINE__ などの定数マクロが用意されている。これは、printf と共に使えば有用なデバッグツールになる。__LINE__ にはソース内の行番号が入る。__FILE__ にはソースのファイル名が入る。文字列定数なので、printf 内では他の文字列と連結できる。ソースのはじめに #define でマクロ定義しておくと便利である。

```
1: #include <stdio.h>
2: #define CLINE printf(__FILE__":%d¥n", __LINE__)
3:
4: int main(void)
5: {
6:     printf("hello, world!¥n"); CLINE;
7:     return 0;
8: }
```

2行目は、

```
2: #define CLINE printf("%s:%d¥n", __FILE__, __LINE__)
```

としてもよい。たとえば、上記のプログラム hello2.c の実行結果は

```
hello, world!
hello.c:6
```

となる。途中でフリーズするようなソースをデバッグする際に CLINE; をブレークポイント代わりにあちこち入れておくと、どこまで実行しているかがデバッガを使わなくても追跡できる。

【目次】|【1.】|【2.】|【3.】|【付録2】|【付録3】|