プログラミング演習1

― 課題4の付録2 ―

コマンド行の引数

→ カーニハン、リッチー「プログラミング言語C」第5章 5.10 (pp.139～140) 参照

Windowsのコマンドプロンプト(DOSプロンプト)やUNIXのコマンドシェルでは プログラム実行時に直接パラメータを引き渡す方法が用意されている。例えばCで作られた大多数のDOSコマンドやUNIXコマンドは、実行時のパラメータにより 機能を切り替えられるようになっている。

プログラムに汎用性を持たせる段階になると、初心者は入力促進させてパラメータや数値を入力させる「対話的なプログラム」を作りがちだが、実はこれは人間が自らコンピュータにこき使われる状況を 作り出しているといえる。最初にパラメータを用いて内容を指示したあと、延々と実行させてほっておき、仕事が終わったら知らせるといった「バッチ処理」こそ、コンピュータにこき使われずに 人間が楽できる本来の使い方である。そのうまみを享受するためには、汎用性のあるプログラムにコマンド行の引数を渡す方法を 早めにマスターするのがよい。main関数の引数として以下のように書くと、コマンド行の引数が利用できるようになる。

```
int main(int argc, char *argv[])
```

整数変数 argc (argument countの意味) にはコマンド行の引数の個数が入る。文字列配列ポインタ argv (argument vectorの意味) にはコマンド行の引数の内容が格納される。最初の文字列 argv[0] には起動したプログラムのコマンド名が入り、これも argc の数に含まれる。2つめ以降の argv[1], argv[2],... にはコマンドに続く引数が、スペースによって区切られて、文字列として順に格納される。よって、以下のプログラム(mainarg.c)

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i;

    for (i = 1; i < argc; i++) printf("%s ", argv[i]);
    printf("¥n");

    return 0;
}
```

を次のように実行すれば、コマンド行の引数が 実際にどのように引き渡されたかを確認できる。

```
Z:¥proen1¥kadai4>mainarg 1.0 234 abc !"#
1.0 234 abc !#
```

ただし、数値を引数にしたつもりでも文字列として引き渡されるので、数値を変数に代入する際には数値変換関数(atoi(), atol(), atof())を用いて 変換する必要がある。例えば実数を3つ入力したい場合は、以下のようにすればよい(mainarg3.c)。

```
#include <stdio.h>  // printf
#include <stdlib.h> // atof

int main(int argc, char *argv[])
{
    double x1, x2, eps;

    if (argc < 4) {
        printf("usage: %s x1 x2 eps¥n", argv[0]);
        return 1;
    }
    x1 =atof(argv[1]);
    x2 =atof(argv[2]);
    eps=atof(argv[3]);

    printf("[% g, % g] % e¥n", x1, x2, eps);

    return 0;
}
```

実行例

```
Z:¥proen1¥kadai4>mainarg3 1 2.0 1e-6
[ 1,  2]  1.000000e-006
```

argc の数にはコマンド名も含まれるため、引数の個数よりも1つ多いことに注意。

2次元配列の引き渡し

→ 熊谷・玉城・白川「例題で学ぶC言語」第12章 12.2～12.3 (pp.110-114) 参照
→ 柴田望洋「新版 明解C言語 入門編」第6章 6-2 (p.138) 参照

配列を関数(サブルーチン)とやりとりする場合、受け取る側では主に2つのいくつかのやりかたがある。

(1)
　大域変数でやり取りする。
(2)
　関数(サブルーチン)の引数を介してやり取りする。

初心者のうちは(1)の方法しか知らなくてもやむを得ないが、プログラムが大きくなると大域変数の管理が大変になるので、(2)のやり方のほうがより一般的で、エレガントである。
引数を介するやりかたにおいては、いくつかの書き方が許される。

1. 配列の大きさ(サイズ)を明記する。

```
int func(double x[3], double f[3])
{
    ..................
}
```

大きさを事前に定数式として定義しておくほうが便利なので、実際には

```
#define ND 3
int func(double x[ND], double f[ND])
{
    ..................
}
```

2. (最上位の)配列の大きさを省略する。

```
int func(double x[], double f[])
{
    ..................
}
```

2次元以上の配列では2つめの方法は要注意である。

```
int Jcalc(double x[], double J[][])
{
    ..................
}
```

では、エラーが出てしまう。

```
int Jcalc(double x[], double J[][ND])
{
    ..................
}
```

としなければならない。これは、引き渡された2次元配列が どこで折り返されているかという最低限のサイズ情報がないと、配列の要素を特定できないからである。省略できるのは最上位(一番左の□)の大きさのみである。
2次元情報を1次元配列に並べ替えておくといった安易な方法もある。

```
int Jcalc(double x[], double J[])
{
    ..................
    J[1] = …;  // J11
    J[2] = …;  // J12
    J[3] = …;  // J21
    J[4] = …;  // J22
}
```
