10 関数 (その3)

10–1 復習 参照呼出し

番地演算子 & と間接演算子 * (7.3節) :

&v ··· 変数 v へのポインタ (≈番地)。

*p ··· ポインタ p の指す記憶領域、
すなわち、p番地の記憶領域。

参照呼出しと同等のことを行なう方法 (7.3節) :

- 参照呼出しの仮引数は、ポインタとして宣言する。
- 関数の本体部では、参照呼出しの仮引数は間接演算子 * を付けて使う。
- 関数を呼ぶ時、参照呼出しの実引数として変数等のポインタ (i.e. 番地) を与える。

例10. 1 (参照呼出し)

```
[motoki@x205a]$  nl func-call-by-ref-Kelley.c
     1 #include <stdio.h>

     2 void swap(int *p, int *q);

     3 int main(void)
     4 {
     5   int i=3, j=5;

     6   swap(&i, &j);
     7   printf("i=%d  j=%d\n", i, j);
     8   return 0;
     9 }

    10 void swap(int *p, int *q)
    11 {
    12   int temp;

    13   temp = *p;
    14   *p   = *q;
    15   *q   = temp;
    16 }
[motoki@x205a]$  gcc func-call-by-ref-Kelley.c
[motoki@x205a]$  ./a.out
```

```
i=5  j=3
[motoki@x205a]$
```

10–2 復習 一次元配列を関数の引数として受渡し

一次元配列 a を関数の引数として受渡しする方法：

- 仮引数側では、次のいずれかの書き方をする。

  データ型 配列名 []

  データ型 *配列名

  データ型 配列名 [大きさ]

補足：
配列の大きさを明示する必要はない。
明示したとしても捨てられる。

- 関数本体の中では、仮引数の配列要素の参照はこれまでと全く同じ様な書き方をする。

- 実引数側では、次のいずれかの書き方をする。

  a

  &a[0]

補足：
配列名は、計算機内部では
先頭要素を指す定数ポインタ
として扱われている。

10–3 多次元配列を関数引数として受渡しする方

多次元配列を関数の引数として受渡しする方法：

- 仮引数側では、1次元目を除く全ての次元の大きさを指定して、

  データ型 配列名 [] [大きさ] ··· [大きさ]

  または データ型 配列名 [大きさ] ··· [大きさ]

  または データ型 (*配列名) [大きさ] ··· [大きさ]

  という書き方をする。

2次元目以降の次元の大きさを指定する理由：

そうしないと、コンパイラが配列の添字の値からその配列要素の番地を割り出せないからである。

また、1次元目の配列の大きさについては明示する必要はない。 明示したとしても捨てられる。

例10. 2 (多次元配列を関数の仮引数とする場合) 3次元配列

int a[7][9][2] を引数として受渡したい時には、仮引数部は次のように書く。

いずれか
- int a[][9][2]
- int a[7][9][2]
- int (*a)[9][2] ⟵ 明示的な書き方

- 関数本体の中では、仮引数の配列要素の参照はこれまでと全く同じ様な書き方をする。

- 実引数側では、
  a または &a[0]
  という書き方をする。

例えば
aが3次元配列の場合、配列名aは計算機内部では2次元配列 a[0] を指す定数ポインタとして扱われている。

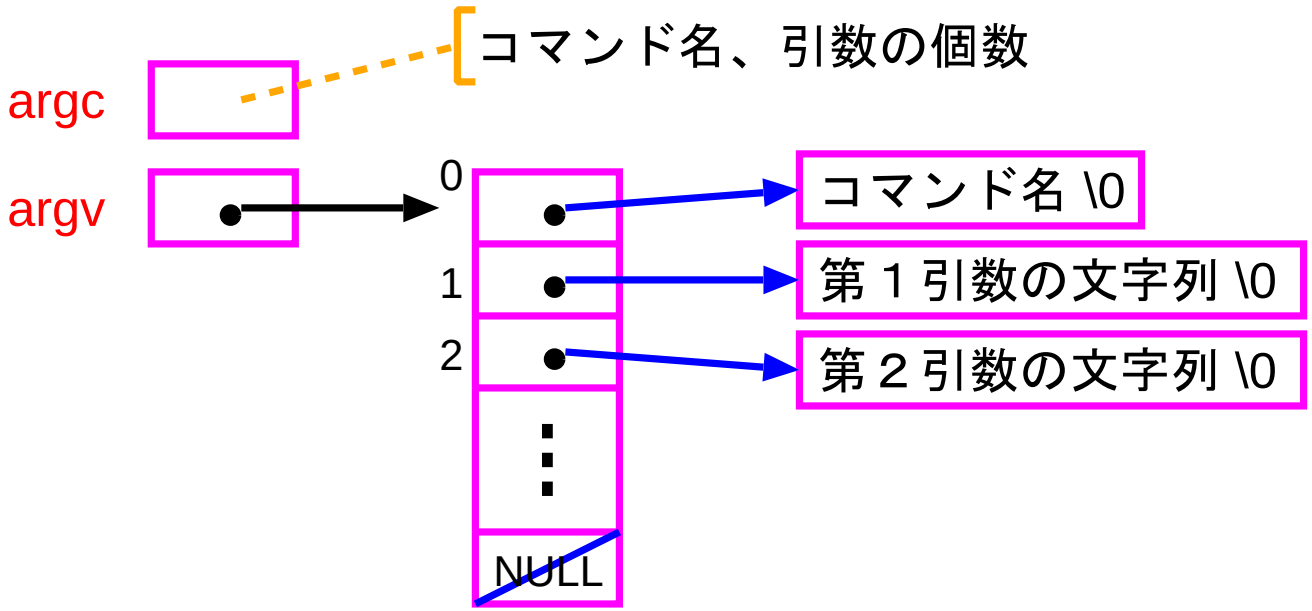
10–4 関数mainの引数—コマンドラインでパラメータを指定

コマンドラインでパラメータ指定する方法について：

- 主ルーチンの引数部を

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

と書けば、コマンドラインでパラメータを指定できる様になる。これによって、関数mainの起動直後には、argc と argv は次の図の様に設定される。



例題10. 3 (コマンドラインでのパラメータ指定)　整数乱数を次々と生成・出力するCプログラムを作成せよ。但し、このプログラムの実行に当たってはコマンドラインから次の形式のオプションを指定できるようにせよ。

- -seed 非負整数　　··· random　seed を非負整数に初期設定する、という指示。(デフォルトは1。)
- -max 正整数　··· [0, 正整数] の間の乱数を生成する、という指示。(デフォルトは RAND_MAX。また、指定した正整数が RAND_MAX よりも大きい場合も、RAND_MAX を用いる。)
- -num 正整数　··· 正整数個の乱数を生成・出力する、という指示。(デフォルトは10。)

(考え方)　この例題はコマンドラインでオプション指定する方法を例示するためのものなので、疑似乱数の生成にはライブラリ関数として用意されている

```
int srand(unsigned seed); と
int rand(void);
```

を用いることにする。

実行直後にコマンドラインオプションを解読することになるが、そのためには main( ) の引数 argc と argv を用いてコマンドに続く空白で区切られた文字列 (へのポインタ) argv[1], argv[2], ..., argv[argc-1] を順に調べて行けば良い。
そして、もしその文字列が "-seed", "-max" または "-num" というものであったなら、標準ライブラリ関数sscanf( )を用いてそれに続く文字列をint型のデータに変換して対応する変数に格納する。

文字列が例えば"-seed" と等しいかどうかの判定には、標準ライブラリ関数strcmp( )を用いれば良い。

(プログラミング)

```
[motoki@x205a]$  nl func-commandline-param-random.c
     1 #include <stdio.h>
     2 #include <stdlib.h>
     3 #include <string.h>

     4 #define  WIDTH  6

     5 int main(int argc, char *argv[])
     6 {
     7   int  i, count=1;
     8   int  seed=1, max=RAND_MAX, num=10;

     9   for (i=1; i<argc; i+=2) {  /* コマンドライン指定の解読
    10     if (*argv[i] != '-') {
    11       printf("Invalid description: %s\n", argv[i]);
    12       exit(EXIT_FAILURE);
```

```
13      }
14      switch (*(argv[i]+1)) {
15      case 's':
16        if (strcmp(argv[i],"-seed")!=0
17            || sscanf(argv[i+1],"%d",&seed)!=1
              || seed<0) {
18          printf("Invalid description: %s %s\n",
                    argv[i], argv[i+1]);
19          exit(EXIT_FAILURE);
20        }
21        break;
22      case 'm':
23        if (strcmp(argv[i],"-max")!=0
24            || sscanf(argv[i+1],"%d",&max)!=1
              || max<=0) {
25          printf("Invalid description: %s %s\n",
                    argv[i], argv[i+1]);
```

```
26          exit(EXIT_FAILURE);
27        }
28        break;
29      case 'n':
30        if (strcmp(argv[i],"-num")!=0
31            || sscanf(argv[i+1],"%d",&num)!=1
                || num<=0) {
32          printf("Invalid description: %s %s\n",
                      argv[i], argv[i+1]);
33          exit(EXIT_FAILURE);
34        }
35        break;
36      default:
37        printf("Invalid description: %s\n", argv[i]);
38        exit(EXIT_FAILURE);
39      }
40    }
```

```
41    srand(seed);
42    for (i=0; i<num; ++i, ++count) {  /* 乱数の生成･出力 *
43      if (max < RAND_MAX)
44        printf("%12d", rand() % (max+1));
45      else
46        printf("%12d", rand());
47      if (count >= WIDTH) {
48        printf("\n");
49        count = 0;
50      }
51    }
52    if (count > 1)
53      printf("\n");
54    return 0;
55 }
[motoki@x205a]$  gcc -o random func-commandline-param-random.c
```

```
[motoki@x205a]$  ./random
  1804289383   846930886  1681692777  1714636915  1957747793
   719885386  1649760492   596516649  1189641421
[motoki@x205a]$  ./random -seed 123 -max 99 -num 20
          93          13          73          30          79
          95          22          26           1          24
          21          91          16          81           9
          46          93
[motoki@x205a]$  ./random -aa 123
Invalid description: -aa
[motoki@x205a]$  ./random -seed aa
Invalid description: -seed aa
[motoki@x205a]$
```

10–5 自習 関数を関数の引数として受渡しする方

関数を引数として受渡しする方法： 例えば、

(引数) (値)

double ⟶ double の 関数を引数とする場合は、次のようにします。

- 仮引数部：

  double f(double) ⟵ (コンパイラがポインタと解釈してくれる。)

  または

  double (*f)(double) ⟵ (明示的な書き方。)

  と書く。

- 引数の参照：

  仮引数がfだと、 f(式) または (*f)(式) と書く。

- 実引数：

  関数名だけを書く。(&は付けない。)

具体的なプログラム例については、ケリー&ポール6.15~16節等を御覧下さい。