第8章 「いろいろなプログラムを作ってみよう」のまとめ

8.4 「再帰的な関数」のまとめ

関数と型 (教 p.240)

再帰 (recursion) とは、関数の定義の中で自分自身を呼び出すこと。一般に x の定義に x 自身を使用すること。

```
factorial(4)
  →  4 * factorial(3)
    →  4 * 3 * factorial(2)
      →  4 * 3 * 2 * factorial(1)
        →  4 * 3 * 2 * 1 * factorial(0)
          →  4 * 3 * 2 * 1 * 1
```

- "自分自身"と言っても、変数（仮引数や自動変数、 List 8-7 の `factorial` の場合 `n`）は別々に確保される。
- 繰返し（`for`, `while`）で簡単に実現できることを, 再帰で書くのは（C 言語の場合）良いこととはいえない。階乗の例題プログラムは、あくまでも再帰を説明するためのものと考えること。（もちろん、再帰を使わなければ簡単に書けないプログラムも今後たくさん出てくる。）
- 再帰関数には、特別な文法も特別な実行規則も必要ない。あくまでも C 言語の普通の関数で、普通の実行規則に基づいて計算される。

Q 8.4.1 次のように定義された関数 `foo` に対して、

```c
void foo(int n) {
    if (n > 0) {
        foo(n - 1);
        printf("%d ", n);
        foo(n / 2);
    }
}
```

`foo(1)`, `foo(2)`, `foo(3)`, `foo(4)` の出力はそれぞれどうなるか?

_____
_____
_____
_____

Q 8.4.2 次のプログラム (hanoi.c) は、ハノイの塔というパズルを解くためのプログラムである。

```c
#include <stdio.h>

void move(int n, int a, int b) {
```

```c
    printf("ディスク %d を棒 %d から棒 %d へ\n", n, a, b);
}

/* n 枚のディスクを a から b に移動する手順 */
void hanoi(int n, int a, int b, int c) {
    if (n > 0) {
        hanoi(n - 1, a, c, b);
        move(n, a, b);
        hanoi(n - 1, c, b, a);
    }
}

int main(void) {
    int n;
    printf("円盤は何枚ですか? "); scanf("%d", &n);
    hanoi(n, 1, 2, 3);
    return 0;
}
```

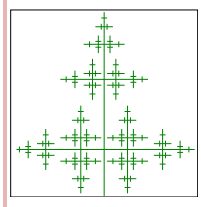
n が 3 のとき、このプログラムの出力はどうなるか?

Q 8.4.3 次のプログラム (tree.c)は、再帰を利用して、樹のような図形を描画する。

```c
#include <stdio.h>
#include <math.h>

void drawTree(int d, double x, double y,
                     double r, double t) {
    /* d       --- 再帰の深さ、
       (x, y)  --- 枝の根元の座標、
       r       --- 枝の長さ、
       t       --- 枝の伸びる向き（ラジアン）*/
    double r1;
    if (d == 0) return; /* 打切り */

    printf("%6.3f %6.3f %6.3f %6.3f\n",
            x,    y,
            x + r * cos(t), y + r * sin(t));
    drawTree(d - 1, x + r  * cos(t), y + r  * sin(t),
                    0.5 * r, t);
    r1 = 0.5 * r;
    drawTree(d - 1, x + r1 * cos(t), y + r1 * sin(t),
                    0.5 * r, t + 0.5 * 3.1416);
    drawTree(d - 1, x + r1 * cos(t), y + r1 * sin(t),
                    0.5 * r, t - 0.5 * 3.1416);
}

int main(void) {
    drawTree(6, 128, 255, 128, - 0.5 * 3.1416);
    return 0;
```

```
28 }
```

次の図はこのプログラムが出力する座標を結ぶ線分を描画したものである。



`drawTree` 関数の中の定数の値を少しずつ変えて、図形がどのように変化するか確かめよ。

8.5 「入出力と文字列」のまとめ

`getchar` 関数と `EOF` (教 p.244)

`getchar` は標準入力から＿＿＿＿を読み込んで返す関数。

`EOF` は `getchar` などが、入力の終わり（＿＿＿＿＿＿に由来）に達した場合に返す値をマクロで `EOF` と書く。（stdio.h に定義されている。）この値は、通常の文字とは区別される。

リダイレクト (教 p.247)

標準入出力をファイルへの入出力につなぎかえることで、C 言語ではなく OS (Unix, MS-DOS など)の機能になる。

コマンド `<` *ファイル名*	— ファイルの内容をコマンドの標準入力に渡す
コマンド `>` *ファイル名*	— コマンドの標準出力をファイルに書込む
コマンド `>>` *ファイル名*	— コマンドの標準出力をファイルの最後に追加する形で書込む

文字コードと数字 (教 p.248)

C 言語では文字は、単にその文字に与えられたコード（整数値）で表す。具体的な値は機種に依存する。

ASCII での文字コードの抜粋:

文字	10 進	16 進
`'0'`	48	0x30
`'A'`	65	0x41
`'a'`	97	0x61

ただし、数字 `'0'`, `'1'`, `'2'`, ..., `'9'`, については、文字コードもこの順に連続していることが保証されている。

Q 8.5.1 次の文の出力はそれぞれどうなるか? ただし、文字コードに依存するものには✗と書け。

```
putchar('1' + 3);
putchar(2 + '4');
putchar('2' + '3');
printf("%d", '9' - '7');
printf("%d", '5' - 2);
```

拡張表記 (教 p.250)

「\n」の他に、「\t」,「\a」,「\b」などいくつかの特殊文字を表す表記がある。特に、バックスラッシュ（円記号）そのものを表す時には「　」と書く。
