第7章　「基本型」のまとめ

7.1　用語のまとめ

教 p.174

文字型と整数型　signedは＿＿＿＿＿＿（負の数も扱える）、unsignedは＿＿＿＿＿＿（正の数と 0 のみ扱える）の整数を宣言する際の型指定子である。

教 p.176

limits.h ヘッダー　各数値型で表現できる値の最小・最大値をマクロとして集めたヘッダーである。

Q 7.1.1 教科書 List 7-1 を実行し、次の空欄を埋めよ。
「Microsoft Visual Studio の場合、INT_MIN は ＿＿＿＿＿＿＿＿、INT_MAX は ＿＿＿＿＿＿＿＿、UINT_MAX は ＿＿＿＿＿＿＿＿である。」

教 p.179

sizeof 演算子

```
sizeof (型名)
```

という形で指定した型のサイズ（単位: バイト）を返す。

```
sizeof 式    /* 通常は上の形にあわせて式を ( 〜 ) で囲む */
```

という形で、式（通常は変数）のサイズ（単位: バイト）を返す。特に、式が配列の場合は配列全体のサイズ（単位: バイト）を返す。
ただし、関数の引数として渡された配列では、別の値（ポインター型のサイズ）を返すので注意する（プログラム例 sizeof.c 参照）。

教 p.181

typedef 宣言　typedef 宣言は型の別名をつける。構造体・ポインターを学習したあとは頻繁に使う。

分類	一般形	補足説明
typedef 宣言	typedef 型 新しい型名 ;	例えば typedef unsigned size_t;

教 p.188

ビット演算　ビット単位の論理演算・シフト演算などは組込み用途では多用される。必要に応じて調べられるようにしておく。

教 p.194

整数定数　8 進定数は先頭に ___を、16 進定数は先頭に ___をつけて表記する。

10 進	8 進	16 進
48	060	0x30
65	0101	0x41
97	0141	0x61

教 p.196

整数の表示　printf 関数で整数を 8 進数または 16 進数で表示するためには、それぞれ、___, ___（A〜F を大文字にしたいときは ___）という書式指定を用いる。

教 p.201

math.h ヘッダー　sin, cos, tan, sqrt （square root — 平方根）, exp, log などの数学関数のプロトタイプ宣言が集められているヘッダーである。

gcc の場合、math.h ヘッダーの関数を使ったプログラムをコンパイルするときは、-lm オプションが必要である。

教 p.202

繰返しの制御　繰返しを制御する変数に、できるだけ浮動小数点数型（float, double）は使わない。

教 p.205

演算子の一覧　優先順位や結合性をすべてを覚える必要はないが、必要に応じて表を調べられるように、どのような演算子があるかくらいは覚えておきたい。(Table 7-11)

7.2　プログラム例

sizeof 演算子の確認 (sizeof.c)

```
1   #include <stdio.h>
2
3   void foo(int x[]) {
4       printf("sizeof(x) = %u\n", (unsigned)sizeof(x));
5   }
6
7   int main(void) {
8       int a[] = { 1, 2, 3 };
9
10      printf("sizeof(a) = %u\n", (unsigned)sizeof(a));
11      foo(a);
12      return 0;
13  }
```

関数の引数として渡される配列に対する sizeof はポインター型のサイズを返す。

第8章　「いろいろなプログラムを作ってみよう」のまとめ

8.1　用語のまとめ

教 p.224

再帰的 (recursive)　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（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多い。）
- 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。

教 p.228

`getchar` 関数　標準入力から＿＿＿＿を読み込んで返す関数。

教 p.228

`EOF`　`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。）

教 p.231

リダイレクト　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。

コマンド名 < 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
コマンド名 > ファイル名	—	コマンドの標準出力をファイルに書込む
コマンド名 >> 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

教 p.232

文字　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。

ASCII コード表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

教 p.234

拡張表記　\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。

8.2　プログラム例

ハノイの塔（再帰）(hanoi.c)

```
1   #include <stdio.h>
2
3   void move(int n, int a, int b) {
4       printf("ディスク%dを棒%dから棒%dへ\n", n, a, b);
5   }
6
7   /* n枚のディスクをaからbに移動する手順 */
8   void hanoi(int n, int a, int b, int c) {
9       if (n > 0) {
10          hanoi(n - 1, a, c, b);
11          move(n, a, b);
12          hanoi(n - 1, c, b, a);
13      }
14  }
15
16  int main(void) {
17      int n;
18      printf("円盤は何枚ですか?␣"); scanf("%d", &n);
19      hanoi(n, 1, 2, 3);
20      return 0;
21  }
```

樹の描画（再帰）(tree.c)

```
1   #include <stdio.h>
2   #include <math.h>
3
4   void drawTree(int d, double x, double y, double r, double t) {
5       /* d       --- 再帰の深さ、
6          (x, y)  --- 枝の根元の座標、
7          r       --- 枝の長さ、
8          t       --- 枝の伸びる向き（ラジアン）*/
9       double r1;
```

```
10      if (d == 0) return; /* 打切り */
11
12      printf("%6.3f␣%6.3f␣%6.3f␣%6.3f\n",
13              x,    y,    x + r * cos(t), y + r * sin(t));
14      drawTree(d - 1, x + r  * cos(t),
15                      y + r  * sin(t), 0.5 * r, t);
16      r1 = 0.5 * r;
17      drawTree(d - 1, x + r1 * cos(t),
18                      y + r1 * sin(t), 0.5 * r, t + 3.1416 / 2);
19      drawTree(d - 1, x + r1 * cos(t),
20                      y + r1 * sin(t), 0.5 * r, t - 3.1416 / 2);
21  }
22
23  int main(void) {
24      drawTree(6, 128, 255, 128,  -3.1416 / 2);
25      return 0;
26  }
```