第B章　教科書 6–8章の復習

B.1　「関数」の復習

教 p.114

関数定義とは

分類	一般形	補足説明
関数定義	**型** 関数名 (**型** 変数名 , . . . , **型** 変数名) 複合文	定義には型が必要

かっこの中の変数は________（空欄 *B*.1.1）(parameter) と呼ばれる。

C 言語の関数定義は必ずプログラムのトップレベルに書く。つまり、関数定義は入れ子にできない（関数定義の中に関数定義は書けない。）

Q B.1.1 整数 (int) を 2 倍する関数 twice を定義せよ。
答: ____

____

____

Q B.1.2 次の関数 chome を定義できるのは、

```
double chome(double x) {
    return 1.41 * x;
}
```

次の /* い */ ～ /* に */ のどこか?

```
#include <stdio.h>
/* い */
int main(void) {
    /* ろ */
    printf("%f", chome(10.0));

    return 0;
    /* は */
}

/* に */
```

答: ________

教 p.114

関数呼出し式とは

分類	一般形	補足説明
関数呼出し式	関数名 (式 , ... , 式)	呼出しには型は不要

かっこの中の式は＿＿＿＿＿（空欄 *B*.1.2）(argument) と呼ばれる。

これで文法上、式 (expression) になる。

関数を呼出すと、プログラムの実行は呼び出された関数の定義の先頭に移り、実引数の値が仮引数の変数の初期値になる。

Q B.1.3 上記の関数 `twice` を用いて、100 の 2 倍を計算する式を答えよ。

答: ＿＿＿＿＿＿＿＿

教 p.120

値呼び（call by value） 引数は基本的に値がやりとりされる。関数呼出しのたびに仮引数のための新しいメモリ領域（“箱”）が用意される。つまり、仮引数の変数に値を行なっても、呼出し元には影響を与えない。

ただし、後述のように引数として配列が渡される場合は例外である。

Q B.1.4 以下のプログラムの出力を答えよ。

```
#include <stdio.h>

void hoge1(int x) {
    x = 1;
}

int main(void) {
    int a = 0;
    hoge1(a);
    printf("a=%d\n", a);
    return 0;
}
```

答: _

教 p.125

有効範囲 (スコープ、scope) 変数には有効範囲がある。同じ変数名でも有効範囲が異なれば別の変数になる。

- ブロックの中で宣言された変数は、そのブロック（宣言された場所から、ブロックの最後まで）が有効範囲となる。
- 関数の仮引数は、その関数本体が有効範囲となる。
- 関数の外で宣言された変数は、宣言された場所からファイルの終端までが有効範囲となる。

配列の受渡し　関数の引数として配列を渡すこともできる。実引数としては配列の名前だけを書く。

- 関数に配列を引数として渡す場合、コピーではなく、配列そのもの（正確にいうと、配列の先頭要素のアドレス）が渡される。
  - int, double 型などの配列でない普通の型の引数の場合は、値がコピーされて渡される。
  - 関数の中で、配列の要素の値を変更すると、呼出し側の配列に反映される。
    int, double 型などの普通の型の引数の場合は、呼出し側には反映されない。
- 引数として渡された配列の要素数を関数の中で知る方法はないので、要素数も引数として渡す必要がある。

Q B.1.5 次のように配列中の要素の和を求める関数を定義した。

```
#include <stdio.h>
#define N 5

int sumArray(int v[], int n) {
    int i, ret = 0;

    for (i=0; i<n; i++) {
        ret += v[i];
    }
    return ret;
}

int data[N] = { 2, 3, 5, 7, 11 };

int main(void) {
    printf("和は␣%dです。\n", [          ]);
    return 0;
}
```

空欄の中に、sumArray を用いて data という配列の中の要素の和を求める式を答えよ。
答: _____

Q B.1.6 以下のプログラムの出力を答えよ。

```
#include <stdio.h>

void hoge2(int y[]) {
    y[0] = 5;
}
```

```
int main(void) {
    int b[1] = { 0 };
    hoge2(b);
    printf("b[0]=%d\n", b[0]);
    return 0;
}
```

答: _

教 p.141

有効範囲と識別子の可視性　同名の変数の有効範囲が重なるとき、より内側のブロックで宣言されているものが優先する。

C 言語のスコープはプログラムのソーステキストのみで決まり、実行時の履歴には無関係な静的スコープである。

Q B.1.7 以下のプログラムでは x というが 3 箇所で宣言され、数箇所で参照され（使われ）ている。それぞれの参照が、どこで宣言された x を指すかを考え、このプログラムの出力を答えよ。

```
#include <stdio.h>

int x = 1;  /* x その 1 */

void foo(void) {
    printf("A:␣x=%d\n", x);  /* この x はどれを指す？ */
}

int main(void) {
    int i;
    int x = 2;  /* x その 2 */
    printf("B:␣x=%d\n", x);  /* この x はどれを指す？ */

    for (i=0; i<2; i++) {
        int x = 3*i;  /* x その 3 */
        foo();
        printf("C:␣x=%d\n", x);  /* この x はどれを指す？ */
    }

    printf("D:␣x=%d\n", x);  /* この x はどれを指す？ */
    return 0;
}
```

答:

記憶域期間　C 言語の変数の寿命（記憶クラス, storage class）には 2 種類のものがある。

- 自動変数 (automatic variable)
  - 関数の中で定義・宣言された変数で `static` という修飾子がついていないもの
  - プログラムの流れが宣言を通過する時に、変数のための領域（箱）が確保され、初期化される。有効範囲を抜ける時に箱が回収される。
  - 初期化子が与えられていない場合、その値は未定義である。
- 静的変数 (static variable)
  - 関数の外で定義・宣言された変数、または関数の中で宣言された変数で、`static` という修飾子がついているもの
  - プログラムの開始時に変数のための領域（箱）が生成され、初期化される。プログラムの終了時まで回収されない。
  - 初期化子が与えられていない場合、 0 に初期化される。

Q B.1.8 次のプログラムの出力を答えよ。

1.
```
#include <stdio.h>

int fuga1(void) {
    int n=0;
    printf("%d␣", n);
    n++;
}

int main(void) {
   int i;
   for (i=0; i<3; i++) {
       fuga1();
   }
   return 0;
}
```

   答: _____

2.
```
#include <stdio.h>

int n=0;
int fuga2(void) {
    printf("%d␣", n);
    n++;
}

int main(void) {
```

```
    int i;
    for (i=0; i<3; i++) {
        fuga2();
    }
    return 0;
}
```

答: _____

3.
```
#include <stdio.h>

int fuga3(void) {
    static int n=0;
    printf("%d␣", n);
    n++;
}

int main(void) {
    int i;
    for (i=0; i<3; i++) {
        fuga3();
    }
    return 0;
```

答: _____

Q B.1.9 上記 Q の 1., 2., 3. で `n=0` と初期化せずに変数を使用した場合、それぞれどうなるか?

1. 答: _____
2. 答: _____
3. 答: _____

B.2 「基本型」の復習

教 p.158

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

Q B.2.1 次のプログラム（の断片）の出力を答えよ。

1. `printf("%d", 0x31)` ··· 答: __
2. `printf("%d", 0100)` ··· 答: __

教 p.170

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、%o, %x（アルファベットを大文字にしたいときは%X）という書式指定を用いる。

Q B.2.2 それぞれ、··· の右のように出力するために、空欄の中に当てはまる書式指定を答えよ。

1. printf("　　", 49) ··· 61
2. printf("　　", 47) ··· 2F
3. printf("　　", 46) ··· 2e

教 p.176

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 7-4)

注意:　^ 演算子は、他のプログラミング言語では累乗の演算子を表すことがあるが、C 言語ではそうではない。
== 演算子と = 演算子、&& 演算子と & 演算子、|| 演算子と | 演算子があることに注意する。

B.3　「いろいろなプログラムを ... 」の復習

教 p.194

再帰 (recursion)　関数のなかで自分自身を呼出すこと。一般に x の定義に x 自身を使用すること。

```
int factorial(int n) {
    if (n==0) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}
```

```
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
```

- 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。ただし、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。自動変数（関数の仮引数も自動変数）は、プログラムの実行が宣言を通過するたびに新しいメモリ領域が生成されることに注意する。

Q B.3.1 次のプログラムの出力を答えよ。

1.
```c
#include <stdio.h>

int piyo(int n) {
    if (n<=0) {
        return 0;
    } else {
        return n + piyo(n-2);
    }
}

int main(void) {
    printf("%d\n", piyo(6));

    return 0;
}
```

答: ___

2.
```c
#include <stdio.h>

int hogera(int n) {
    if (n<=0) {
        return 1;
    } else {
        return n * hogera(n-1) * hogera(n-2);
    }
}

int main(void) {
    printf("%d\n", hogera(4));

    return 0;
}
```

答: ___