第6章 「関数」のまとめ

6.1 「関数とは」のまとめ

関数とは (教 p.142)

繰返し使うプログラムの一部の命令列を部品として、再利用できるようにしたもの。（他の言語ではサブルーチン・手続き・副プログラムなどとも呼ばれる。）

関数定義 (教 p.143)

分類	一般形
関数定義	***型*** *関数名*（ ***型*** *変数名*$_1$，…，***型*** *変数名*$_n$ ）*複合文*

関数定義には型が必要である。

かっこの中の変数（変数名$_1$～変数名$_n$）は＿＿＿＿(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、関数定義の中に関数定義は書けない。他のブレース「`{`」～「`}`」の中に入らない。）また、後述のプロトタイプ宣言をしているときを除き、使うよりも先（上）に書く。

Q 6.1.1 次のような関数を定義せよ。

1. `int` 型の引数 `n` を受け取って、`2 * n + 1` を返す関数 `foo`

   ＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

2. `double` 型の引数 `x` を受け取って、`x / 2` を返す関数 `bar`

   ＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

関数呼出し (教 p.144)

分類	一般形
関数呼出し式	*関数名*（ *式*$_1$，…，*式*$_n$ ）

関数呼出しには型は不要である。

かっこの中の式（式$_1$～式$_n$）は＿＿＿＿(argument) と呼ばれる。

これで文法上、式 (expression) になる。

注: ここのコンマはコンマ演算子ではない。

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。

return 文 (教 p.145)

分類	一般形	補足説明
return 文	`return` 式 `;` `return ;`	値を返す場合 値を返さない場合

関数の呼出し元に値を返す。つまり、プログラムの実行が関数の呼出し元に戻り、 return 文の式の値が、関数呼出し式の値になる。

関数本体の最後の閉じブレース「}」にたどり着いたときも、プログラムの実行は関数の呼出し元に戻る。

値渡し（pass by value） (教 p.150)

値呼び（call by value）とも言う。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。仮引数の変数に値の代入を行なっても、呼出し元の実引数は＿＿＿＿＿＿。

Q 6.1.2 次のプログラム (cbv.c) の出力は?

```c
#include  <stdio.h>

void i_set(int v) {
    v = 0;
}

int main(void) {
    int a1 = 1, a2 = 3;

    i_set(a1);
    i_set(a2);

    printf("a1 = %d\n", a1);
    printf("a2 = %d\n", a2);

    return 0;
}
```

＿＿＿＿＿＿＿＿

6.2 「関数の設計」のまとめ

値を返さない関数 (教 p.152)

関数の定義の返却値型を書くところに ＿＿＿ と書く。

引数を受け取らない関数 (教 p.154)

関数の定義の仮引数のならびを書くところに＿＿＿と書く。関数の定義の仮引数のならびを空にすると、古い C 言語の規格で書かれていると見なされてしまい、意味が変ってしまう。関数を呼出すときは () のなかは空にする。

Q 6.2.1 引数を受け取らず `"Hey!"` と出力する（改行はしない）、値を返さない関数 `hey` を定義せよ。

ブロック有効範囲・ファイル有効範囲 (教 p.155)

変数には有効範囲(スコープ、scope)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（教 p.60）の中で宣言された変数（局所変数、ブロック有効範囲を持つ変数）は、宣言された場所から、＿＿＿＿＿＿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（大域変数・グローバル変数、ファイル有効範囲を持つ変数）は、宣言された場所から＿＿＿＿＿＿までが有効範囲となる。どうしても必要でない限り、使わないこと。

関数プロトタイプ宣言 (function prototype declaration) (教 p.157)

関数を定義より前に（あるいは定義されているのと別のファイルで）使用する場合は、関数プロトタイプ宣言が必要である。

以下を“宣言”に追加する。

分類	一般形
関数プロトタイプ宣言	***型*** *関数名*（ ***型*** *変数名*，…，***型*** *変数名* ）;

変数名は省略可能である。

関数定義がその呼出しよりも前にある場合は、定義が宣言を兼ねるのでプロトタイプ宣言は不要である。（いずれにしても、実行は常に `main`から開始される。）

ヘッダーとインクルード (教 p.158)

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

の stdio.h は、`printf`, `putchar` などの関数のプロトタイプ宣言が集められたもの（通常はファイル）である。このようにプロトタイプ宣言やマクロの定義が集められたものを＿＿＿＿と呼ぶ。

`#include` はヘッダーの内容を、そっくりそのままその場所に読み込む（インクルードする）指令である。

処理系により標準のヘッダーがおかれる場所は異なる。

ライブラリー関数（前もって用意された関数）を利用するときは、ほとんどの場合、適切なヘッダーをインクルードする必要がある。例えば、 `sin`, `cos`,

sqrt などの数学関数を利用するときは math.h というヘッダーをインクルードする。

関数の汎用性

できるだけ大域変数を使わないようにする。 (教 p.159)

配列の受渡し (教 p.160)

関数の引数として配列を渡すこともできる。仮引数の宣言は、 型名　引数名[] としておき、実引数としては　　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。**（重要）**
    - 一方、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。（プログラム例の cbv.c 参照）
    - 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、通常は要素数も引数として渡す必要がある。

配列の受渡しと const 型修飾子 (教 p.162)

関数の引数の配列が書換えられないことを保証するためには、　　という型修飾子を仮引数の宣言につける。 const をつけているのに、その変数を書き換えようとするとコンパイル時にエラーになる。

Q 6.2.2 次のプログラム (cbr.c) の出力は?

```c
#include  <stdio.h>

void a_set(int v[]) {
    v[0] = 0;
}

int main(void) {
    int a1[1] = { 1 }, a2[1] = { 3 };

    a_set(a1);
    a_set(a2);

    printf("a1[0] = %d\n", a1[0]);
    printf("a2[0] = %d\n", a2[0]);

    return 0;
}
```

番兵法 (sentinel) (教 p.166)

探索の対象となっているデータ（　　(sentinel)）をデータの最後に付け加えること。探索範囲の終わりのチェックをする必要がなくなるので、少し効率が良くなる。

6.3 「有効範囲と記憶域期間」のまとめ

有効範囲と識別子の可視性 (教 p.172)

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。

Q 6.3.1 次のプログラムの出力は？

```c
#include  <stdio.h>

int x = 9;

void foo(void) {
   printf("%d ", x);
}

int main(void) {
    int x = 5;
    printf("%d ", x);
    for (int i = 0; i < 2; i++) {
       int x = 2 * i;
       printf("%d ", x);
    }
    foo();
    printf("%d ", x);
    return 0;
}
```

記憶域期間 (教 p.174)

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) — 自動記憶域期間を持つ変数
    - 　　定義された変数で `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ときに、変数のための領域（箱）が確保され、初期化される。有効範囲を抜けるときに箱が回収される。
    - 初期化子が与えられていない場合、その値は　　となる。
- 静的変数 (static variable) — 静的記憶域期間を持つ変数

- ＿＿＿＿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
- ＿＿＿＿＿＿＿＿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、＿に初期化される。

静的変数は、過去の呼出しによって結果が変わるような関数の場合には必要となる。逆に言うと、必要なければ使ってはいけない。

Q 6.3.2 次のプログラムの出力は？

```c
#include <stdio.h>

void foo(void) {
    static int x = 0;
    printf("%d ", x++);
}

void bar(void) {
    int z = 9;
    printf("%d ", z--);
}

int main(void) {
    foo(); bar(); foo(); bar();
    return 0;
}
```

＿＿＿＿＿＿
