ウォームアップ

[練習 1] 整数を読み込んで、その絶対値を求めるプログラムを作りなさい。

11. 関数

今までこの授業では、printf、scanf などの関数を使ってきた。これらの関数はみなさんの使っている C コンパイラにあらかじめ組み込まれたもの (標準ライブラリ関数という) だが、新しい関数を自分で作ることもできる (これをユーザー関数という)。
次の例は、$|a| + |b|$ (a と b は整数) を計算するプログラムである。

```
 1  #include <stdio.h>
 2
 3  int abs(int);                              /* 関数 abs のプロトタイプ宣言 */
 4
 5  main()
 6  {
 7    int a = 3, b = -4, kekka;                /* 整数変数 a, b, kekka の宣言 */
 8
 9    kekka = abs(a) + abs(b);                 /* |a|+|b| を計算して kekka に代入 */
10    printf("%d\n", kekka);                   /* 結果を出力する */
11  }
12
13  int abs(int x)                             /* 関数 abs の定義 */
14  {
15    int kekka;                               /* ローカルな変数の宣言 */
16
17    if(x>=0) {
18       kekka=x;
19    } else {
20       kekka=-x;
21    }
22    return(kekka);                        /* 関数 abs の結果として kekka の値を返す */
23  }
```

このプログラムの実行を辿ってみよう。まず main の中身 (4 行目から 8 行目まで) の頭から実行されるのは今までと同様。しかし、6 行目で abs(a) を計算しようとすると、abs という名前の関数を探して、実行はその関数のプログラムへ移る。移る時に、その関数に () 内の式の値 (この場合変数 a のこの時点での値 3) をデータとして渡す。

9 行目以降がユーザー関数 abs の定義となる。abs は、1 つの整数をもらってその絶対値を計算し返す。一般に、もらってくる値を「**実引数** (actual parameter)」、返す値のことを「**返り値** (return value)」と呼ぶ。もらってくるのは値だけなので、ユーザー関数を定義する時は、その値をしまうための仮の変数が必要となる。この仮の変数のことを、一般に「**仮引数** (formal parameter)」という[1]。9 行目の abs の定義では、x が仮引数となる。返り値にも、仮引数にも、型を指定しなくてはならない。9 行目の頭の int が返り値の型の指定で、int x の int が仮引数 x の型の指定になっている。関数の定義の仕方をまとめると次のようになる。

```
返り値の型　関数名 (型 仮引数, 型 仮引数, ....)
{
        .....
        return(式);
}
```

さて、6 行目で abs(a) の処理に移ると、その時の a の値 (3) が仮引数 x の箱の中へコピーされる。10 行目以降が main の時と同じように順々に実行される。今の場合 x の値が 3 なので、13 行目の代入が実行されることとなり、17 行目で kekka の値 (3) が返り値となって、「その関数が呼ばれたところに」戻る。こ

[1] 実引数や仮引数をひっくるめて単に引数ということもある。

の場合、6 行目に戻って、返り値 (3) が `abs(a)` の値として使われ、実行が続けられる。この場合は、次の `abs(b)` が実行され、関数 `abs` が再び呼び出される。今度は、仮引数 `x` へは `b` の値 (−4) が渡される。そして、今度は 15 行目の代入が実行され、17 行目で `kekka` の値 (4) が返り値となって、6 行目に戻り、`main` の実行が続けられる。

プロトタイプ宣言

2 行目は、プロトタイプ宣言と呼ばれるもので、ユーザー関数を使う時は (通常) 必要となる。大雑把に言うと、関数定義の最初の行だけを書いたもの。今の例では、

```
int abs(int);
```

と、引数の型だけが括弧の中に書かれているが、仮引数名も書いて、

```
int abs(int x);
```

とすることもある。単に、

```
int abs();
```

として、引数に関する情報を書かない省略記法もあるが、あまりよくない[2]。

関数は便利!

ユーザー関数の中で他のユーザー関数を呼ぶことも出来る。関数をうまく用いると、プログラム全体の見通しを良くしたり、同じことを何度も書かなくて済むようにできる。

注意

例のプログラムで、関数 main の中で変数 kekka を使い、関数 abs の中で 変数 kekka を使っている。この 2 つの kekka は別のもので、関数 abs 中の kekka は関数 abs に制御が移るたびに作られる。

[練習 2] 前ページのプログラムで、関数 abs が小数値 (float) を扱えるようにするにはどこを書き換えればよいか。

[練習 3] 2 つの整数を引数にとって、小さくない方を返す関数 `notsmaller` を作りなさい。

[2] 古い C コンパイラはこのような宣言しかできないので、互換性のために許されている。この授業で使っているような新しい (ANSI 規格を満たす)C コンパイラでは引数の情報も書いた方が良い。

[練習 4] 2 つの整数を引数にとって、その平均値を小数 (float) で計算して返す関数 heikin を作り、それを使って 2 つの整数を入力して平均を小数で出力するプログラムを書きなさい。

補足　前の説明で、実引数の値が仮引数の箱の中へコピーされると書いた。よって、ユーザー関数の中で仮引数の箱の中身を書き換えても、実引数の方には何の影響もない。すなわち、仮引数の名前はなんでもよく、次のような abs の定義でも全く同一の動作となる。

```
int abs(int p)
{
  int kekka;
  if(p>=0) { kekka=p; }
  else { kekka=-p; }
  return(kekka);
}
```

それでは、仮引数の x を a にしたらどうだろうか。6 行目で関数 abs を呼び出す時に abs(a) となっているので、何か問題が起こるのではと心配する人もいるかもしれない。しかしこの場合、

```
int power(int a)
{
  int kekka;
  if(a>=0) { kekka=a; }
  else { kekka=-a; }
  return(kekka);
}
```

としても main の中の a と abs の中の a は全く別のものとして区別されるので、何の問題もない。なお、関数 abs は次のように簡潔に定義できる。このようにすると、kekka といったローカルな変数が不要になる。

```
int abs(int x)                          /* 関数 abs のより簡潔な定義 */
{
  if(x>=0) {
      return(x);
  } else {
      return(-x);
  }
}
```