4　C 言語によるプログラミング 4: 関数

目 次

図 目 次

4.1 概説

プログラムのなかで、ある機能をもつ部分をひとまとまりにしたものを関数という [1, 2] 。関数の定義の部分は main program の後か前に記述する。（別のファイルに書くこともある。）主関数 main は、プログラム全体の流れを指定し、他の関数を呼んで必要な作業をおこなう。main 関数から呼び出された関数もまた、おなじように別の関数を呼び出して処理を実行する。

関数には、プログラマー自身のつくるユーザー関数、すなわち「お手製の関数」(※ これはここだけの呼び方である) とあらかじめコンパイラーによって用意されているライブラリー関数がある。前者をつかうばあいは、はじめにその関数がでてくる前に、あらかじめ入力の変数 (引数(ひきすう)、argument)、および出力の変数 (戻り値(もどりち), return value）のタイプを記述しておくことが必要である (関数プロトタイプ宣言)。ただし、関数の中味を主関数の前に定義した場合、基本的にプロトタイプ宣言を省略できる [4]。

プログラムは main 関数から始まり、該当する箇所にくると、関数のほうに引き数が渡される。関数のほうのプログラムが終わると、戻り値が呼び出した関数のほうに返されることになる。

4.2 ユーザー関数 (お手製の関数)

4.2.1 引数１個、戻り値１個の場合

たとえば、ループの例題『1 から n までの和』のプログラムを関数をつかってかくと、つぎのようになる。

例 1

```
/* sum3.c */

#include <stdio.h>

int sum(int a);  /* 関数プロトタイプ宣言 。たんに、int sum(int); としてもよい。*/

main()
{
    int n;

    printf("1 から n までの整数の和をもとめます。\n");
    printf("整数 n を入力してください。n: ");
    scanf("%d", &n);
    printf("\n1 から %d までの和は %d です。\n", n, sum(n));

}

int sum(int a)   /* 関数の定義  */
{
    int i, wa = 0;

    for (i = 1; i <= a; i++)
    {
```

```
        wa += i;   /* wa = wa + i */
    }

    return wa;
}
```

関数プロトタイプ宣言

入力と出力の変数の型を、その関数をつかう前に宣言する必要がある。

`int sum(int a);`

の部分がこれに相当している。数学の関数 $y = f(x)$ と、$f(x) \leftrightarrow$ `sum(int x)` のような対応がある。括弧「()」のなかにあるのが入力変数 (引数、argument) である。。sum が出力の変数 (戻り値, return value）である。数学では x, y を変数 (それぞれ独立変数、従属変数) と呼ぶが、プログラミングでは変数という言葉はもっと一般的につかっている。また、関数の入力にたいする argument という表現は、プログラミング以外でもつかわれることもある。プロトタイプ宣言では、引数、および出力の変数 (戻り値, return value）のタイプさえ示されていればよい。したがって、たんに、`int sum(int);` としてもよい。

プロトタイプ宣言の基本的な表記のしかたは、

```
戻り値の型　関数名 (引数の型と名前);
```

あるいは、

```
戻り値の型　関数名 (引数の型);
```

となる。

関数の定義

主関数の後の `int sum(int a)` の`{ }`の内が関数の定義である。関数の定義の基本的な表記のしかたは、

```
戻り値の型　関数名 (引数の型と名前)
{
    宣言
    文
}
```

となる。

プログラムの流れ

このプログラム (sum3.c) をコンパイルして、実行する。まず、プログラムは主関数から始まる。プログラムは `n` の入力を促してくる。いま、仮に、`n` に 3 をあたえたとする。すると、つぎに主関数は、出力のところで、関数 `sum(3)` を呼ぶ。このとき、関数内の変数 `a` に `n` の値 3 のコピーが渡される。これを渡して、関数工場に 1 から 3 までの和を発注するのである。これ以降、プログラムの進行は関数の方に移る。そして、関数内で然るべき結果を得るまで作業をおこなう。その結果 6 が関数内の変数の `wa` に納められる。最後に、「`return wa;` 」でこの値を発注元 (=関数を呼び出したところ。いまの場合、主関数内の printf の箇所) に戻す。ここから、プログラムはふたたび主関数を走る。しかし、この例は簡単なので、こでもうプ

ログラムは終了する。

パラメータと引数

ここで注意すべきは、主関数の内の変数 n は主関数内のみに棲息する変数であり、関数 sum 内の a は関数 sum 内でのみ生存する変数 (局所変数、自動変数) である、ということである。よしんば、おなじ変数名がついていたとしても、両者は別物である。

関数の定義の括弧のなかに挙げられた変数をパラメーター (仮引数)、関数呼出しにつかわれる値を引数 (実引数) と呼ぶこともある。

関数の定義を先に記述する場合

関数の定義は関数のプロトタイプ宣言を含んでいる。したがって、関数の定義を始めにしておけば、その関数を使用する前に宣言をしたことになる。この場合、宣言を別けて記述する必要はない。ただ、複数個の関数の間で呼び出しをしあう場合などで、プロトタイプ宣言の必要なことも生じる [4, 10]。§ 4.3 の例 2(pell1a.c) も参照のこと。

例 1a

```
/* sum3a.c */

#include <stdio.h>

int sum(int a)   /* 関数の定義  */
{
    int i, wa = 0;

    for (i = 1; i <= a; i++)
    {
       wa += i;   /* wa = wa + i */
    }

    return wa;
}

main()
{
    int n;

    printf("1 から n までの整数の和をもとめます。\n");
    printf("整数 n を入力してください。n: ");
    scanf("%d", &n);
    printf("\n1 から %d までの和は %d です。\n", n, sum(n));

}
```

以下、関数の定義を先に記述する方法をつかうことにする。

4.2.2 引数が２個、戻り値１個の場合

自然数 n, p をキーボードからあたえる。このとき、1 から n までの和をもとめる。ただし、p の倍数目の項の符号を反転させることにする。
すなわち $f(n,p)$:

$$f(n,p) \equiv 1 + \cdots - p + \cdots + n$$

をもとめることになる

例 2

$f(5,2) = 1 - 2 + 3 - 4 + 5 = 3$
$f(2,3) = 1 + 2 = 3$
$f(8,3) = 1 + 2 - 3 + 4 + 5 - 6 + 7 + 8 = 18$

これを関数をつかって解いてみよう。すぐわかるように関数への入力は、n,p に相当する 2 変数である。すなわち、引数 2 個、戻り値 1 個 (いずれも整数型) の関数をつくればよい。この関数を sum_pm1 とすると、ソースプログラムは、たとえば、つぎのようになる。

```
/*** sum_pm1.c   **/

#include <stdio.h>

/*****  関数 sum_pm1 の定義 (プロトタイプ宣言を兼ねる)  *****/
int sum_pm1(int n, int p)
/************************************************/
{
    int i, sign, sum = 0;
  /* i は添字、sign は i 項目の符号、sum は i 項目まで の部分和*/

    for (i = 1; i <= n; i++)
    {
       sign = 1;        /* 符号は基本的に正と採る。*/

       if ((i % p) == 0)
       {
          sign = - sign;  /* i が p の倍数のとき、符号を変える。*/
       }

       sum += sign * i;
    }

    return sum;
}

/******  主関数  *****/
```

```
main()
/******************/
{
   int n, p;

   printf("1 から n までの和をもとめます。\n ");
   printf("ただし、p 番目の項の符号を反転させます。\n ");
   printf("整数 n を入力してください。n: ");
   scanf("%d", &n);
   printf("整数 p を入力してください。p: ");
   scanf("%d", &p);

   printf("1 - p + ⋯ (-1)^%d・ %d = %d\n", n - 1, n, sum_pm1(n, p));
}
```

プロトタイプ宣言

上の例に示したように、

```
int sum_pm1(int n, int p);
```

あるいは、

```
int sum_pm1(int , int );
```

のように書けばよい。数学の 2 変数関数のときの表記法、たとえば $z = f(x, y)$、との類似性がわかるであろう。

プロトタイプ宣言の基本的な表記のしかた は、

```
戻り値の型　関数名 (引数の型と名前 1, 引数の型と名前 2);
```

あるいは、

```
戻り値の型　関数名 (引数の型 1, 引数の型 2);
```

となる。

つぎの例は、負でない整数 a, b にたいして、a^b, b^a のおおきさを評価している。

例 3

```
#include <stdio.h>

int beki(int p, int n)
{
   int i;
   int prod = 1;

   for (i = 1; i <= n; i ++)
```

```
   {
        prod *= p;
   }

   return prod;
}

main()
{
   int a, b;

   printf("a^b と b^a をおおきい順に出力します。\n");
   printf("整数 a, b を空白を置いて入力してください。\n");
   scanf("%d %d", &a, &b);

   if (beki(a, b) > beki(b, a))
   {
       printf("a^b = %d^%d = %d\n", a, b, beki(a, b));
       printf("> b^a = %d^%d = %d\n", b, a, beki(b, a));
   }
   else if (beki(a, b) == beki(b, a))
   {
       printf("a^b = %d^%d = %d\n", a, b, beki(a, b));
       printf("= b^a = %d^%d = %d\n", b, a, beki(b, a));
   }
   else
   {
       printf("b^a = %d^%d = %d\n", b, a, beki(b, a));
       printf("> a^b = %d^%d = %d\n", a, b, beki(a, b));
   }
}


]$ ./a.out
a^b と b^a をおおきい順に出力します。
整数 a, b を空白を置いて入力してください。
5 2
b^a = 2^5 = 32
> a^b = 5^2 = 25
[matsu@localhost 冪]$ ./a.out
a^b と b^a をおおきい順に出力します。
整数 a, b を空白を置いて入力してください。
2 4
a^b = 2^4 = 16
= b^a = 4^2 = 16
```

4.2.3 複数の戻り値の場合

まず、単純にかんがえて、戻り値ひとつの関数を複数個用意するという手がある。たとえば、複素数から複素数への関数の場合、実部と虚部がそれぞれ戻り値となるような関数を、合わせてふたつ用意すればよい。よく似た例だが、ベクトルからベクトルへの関数の場合、ベクトルの各成分がそれぞれ戻り値となるような関数を、ベクトル空間の次元だけ用意すればよい。

だが、ひとまとまりの意味のあるものは一括してとりあつかうほうが自然である。上記の場合も、複素数やベクトルを構造体 (structure) として宣言し、あらたな変数の型をつくればよい。そして、構造体から構造体への関数を定義するのがいちばん自然である。

あるいは、配列 (おなじ型の変数をメモリ上に連続的に配置したもの。変数の団体さん) を引数や戻り値にとる関数もある。とくに文字列の操作にかんする関数はそうである。C では、文字列を文字配列としてあつかうからである。

配列や構造体の場合、じっさいに関数に引き渡されるのは、メモリの位置 (アドレス) を指し示す変数 (ポインタ変数) である。詳しくは、たとえば、[11] を参照されたい。

(※ この授業では「構造体」まで未だ至らず ···。)

4.2.4 引数や戻り値のない場合

関数をつくるとき、とくに、引数や戻り値の必要のない場合がある。このときは、引数、あるいは戻り値のないことを示すために、その変数の型として void (「空」という意味) と表示する。

サイコロ賭博の例 (§ 3.3.1 例 2 参照) をつかおう。

例 4

```
/* saikoro2.c  */

#include <stdio.h>

/****** 偶奇をきめる関数 *******/
int even_odd(int a, int b)
{
    int c;

    c = (a + b) % 2;

    return c;
}

/****** 丁半を出力する関数  ******/
void cho_han(int a, int b)
```

```
{
   int c1;

   c1 = even_odd(a, b);

   if (c1 == 0)
   {
      printf("%d %d の丁\n", a, b);
   }
   else
   {
      printf("%d %d の半\n", a, b);
   }
}

/****** 主関数    ******/
main()
{
   int a, b;   /*  変数の宣言 */

   printf("サイコロの目をふたつ、空白を置いて、あたえてください。");
   scanf("%d %d", &a, &b);

   cho_han(a, b);
}
```

関数 cho_han は引数 2 個である。しかし、戻り値はないので、戻り値の変数の型のところは void になっている。

主関数もこのような書きかたにしたがうと、以下のようになる [4]。

```
int main(void)
{
     ...

     return 0;
}
```

ここで、0 を返しているのはプログラムの正常終了を表わす。これはプログラムの実行される環境にたいして返している [12]。ただし、main は特殊であるので、上記の表現にあまりこだわらず、以前のように、引数と戻り値を表記しないという特別な書きかたをしてもよいであろう [4]。その場合、コンパイラに依ってはエラーの出るものもあるかもしれない。そのときは、`return 0;` をつけ加えればエラーは解消するとおもう。

♣ 演習問題 9 ♣

演習問題 7 の各問を、お手製の関数をつくってこれを呼び出すことによって、解け。

4.3 関数の再帰呼び出し

C では、関数のなかで、その関数自体をよびだすことができる (再帰呼び出し) [1, 3, 7, 9]。このような関数の定義を再帰的定義 (帰納的定義) という。漸化式をおもいおこすとよいだろう [5, 6]。なんとも魅力的なやりかたである。だが、長所ばかりでもない。メモリをつかいすぎるということがある [3]。もちろん、これにたいする対策 (再帰呼び出しの解消 [13] も存在する。

繰返しのプログラムを再帰呼び出しで解いてみるのもよい練習かもしれない。たとえば、§ 4.2.1 の例では、`sum(int a)` をつぎのようにすればよい。

例 1

```
int sum(int a)
{
    if (a > 0)
    {
        return sum(a - 1) + a;
    }
    else
    {
        return 0;
    }
}
```

※ もちろん、つぎのようにしてもよい。

例 1a

```
int sum(int a)
{
    int wa;

    if (a > 0)
    {
        wa = sum(a - 1) + a;
    }
    else
    {
        wa = 0;
    }

    return wa;
}
```

つぎに、関数が複数個あるときの再帰呼び出しの例として、§3.5 の差分方程式を再訪する。以下のものであった。

$$\begin{cases} a_1 = b_1 = 1 \\ a_{i+1} = a_i + 2b_i \\ b_{i+1} = a_i + b_i \end{cases} \qquad (\heartsuit)$$

今回の再帰呼び出しの場合のほうが (♡) をそのままソースコードに直せばよいことがわかる。また、繰り返しをつかったときのように、変数の上書きを心配することもない。しかし、繰り返し版のほうがメモリの節約ができることはあきらかである。

例 2

```
/* pell1a.c */
/* 関数 */
/* 再帰呼び出し版*/
/*この gcc の版では、int b(int n); のプロトタイプが先に来なくても動く。*/

#include <stdio.h>

int b(int n);

/************************************/
int a(int n)
/************************************/
{
    if (n > 1)
    {
        return a(n - 1) + 2 * b(n - 1);
    }
    else
    {
        return 1;
    }
}
/************************************/
int b(int n)
/************************************/
{
    if (n > 1)
    {
        return a(n - 1) + b(n - 1);
    }
    else
   {
```

```
        return 1;
    }
}
/***********************************/
int u(int n)
/***********************************/
{
    return a(n) * a(n) - 2 * b(n) * b(n);
}

/***********************************/
main()
/***********************************/
{
    int  i, n;

    printf("差分方程式\n");
    printf("a(1) = b(1) = 1\n");
    printf("a(i + 1) = a(i) + 2 * b(i)\n");
    printf("b(i + 1) = a(i) + b(i)\n");
    printf("の i =1 から n までの解をもとめます。\n");
    printf("また、u(i) = a(i)^2 - 2 * b(i)^2 と表記します。\n");
    printf("n をあたえてください。");
    scanf("%d", &n);

    printf("\n  i\t\t  (a(i), b(i))  \t u(i)\n");
    printf("========================");
    printf("========================\n");

    for (i = 1; i <= n; i++)
    {
        printf("%3d\t (%10d, %10d)\t %2d\n", i, a(i), b(i),
u(i));
    }
}
```

gcc の版で試すと、int b(int n); のプロトタイプがなくとも動く。この辺はコンパイラーの反応みるとよい。あるいは、以下のように先にプロトタイプ宣言をすべてしておくのもわかりやすいかもしれない。

例 2a

```
#include <stdio.h>
int a(int n);
int b(int n);
main()
```

```
{
  ...
}
int a(int n)
{
  ...
}
int b(int n)
{
  ...
}
```

♣ 演習問題 10 ♣
演習問題 7 の各問を、再帰的な関数をつかって解け。

4.4 ライブラリ関数

ライブラリ関数をあつかうのは、図書館から文献を借りてくるようなものである。標準ライブラリにはいくつもの関数群がある。それらは各ヘッダファイルのなかですでに標準入出力関数ヘッダ<stdio.h>の場合は、すでにおなじみであろう。他のものもおなじようにすればよい。#include で指定するのである。プロトタイプ宣言をする必要はない。ただし、事前に引数と戻り値の型を調べておくことは肝要である。市販の本の巻末には、たいてい、ライブラリ関数の説明がまとめて載っている [1, 9]。必要とする機能をもった関数がどのライブラリにあるか、また、その引数と戻り値の型は何か、そこで見出すことができるであろう。以下に節を分けて例をあげてみよう。

4.4.1 数学関数のとりあつかい

#include <math.h>で、数学関数のライブラリとのリンクをつける。この場合、とくに、以下の二点に注意が必要である。
(i) 引数と戻り値がともに倍精度実数型 (double) になる。
(ii)gcc の場合、コンパイル時に、 -lm(エル・エム) が必要になる。Windows 用のコンパイラの場合、最初の設定で数学関数を読み込むようにしておけばよい。

ここでは、数学関数を 2 重ループと合わせた例で見てみよう。

例 1

実数 a をあたえたとき、x-軸, y-軸 $x = a$ および $y = \sqrt{x}$ でかこまれた領域のなかの格子点をみつけだす (図 1 参照)。

```
  /*   lattice.c */

#include <stdio.h>  /* 標準入出力関数の取り込み*/
#include <math.h> /*  数学関数の取り込み */

main()
{
```

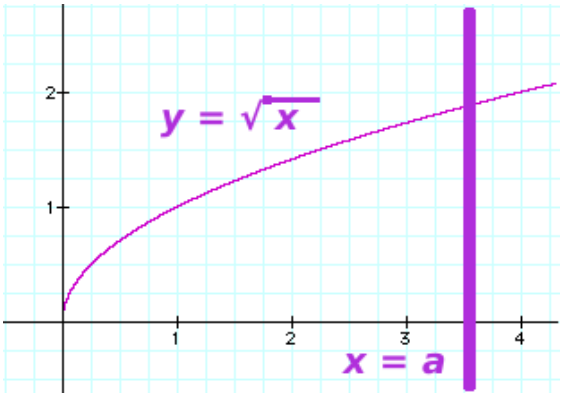

図 1: 数学関数と二重ループ

```
    double x, y, a;  /*  数学関数の引き数は倍精度実数。*/
    int n = 0;     /*n は格子点の数をあらわす変数  */

    printf("x-軸、y = √ x、x = a でかこまれた領域の格子点を列挙し、\n");
    printf("その個数をもとめます。\n");
    printf("実数 a をあたえてください。a: ");
    scanf("%lf", &a); /* キーボードから、倍精度実数を読み込む。(lf は エル・エフ) */

    for (x = 1; x < a; x++)      /*  x-軸方向に進む。*/
    {
        for (y = 1; y < sqrt(x); y++)    /* y-軸方向に進む。   sqrt …… 平方根 */
        {
            printf("(%.0f, %.0f) ", x, y); /* %.0f で整数部分だけを出力   */
            n++;                /*   格子点の数を勘定する。*/
        }
        printf("\n");
    }

    printf("格子点の数は %d です。\n", n);

}
```

♣ こんぱいる・おぷしょん ♣

```
gcc lattice.c -lm
```

lm を抜かした場合のエラー表示 (Vine Linux の例)

```
$ gcc lattice1.c
/tmp/cc3STpoq.o(.text+0x76): In function ‘main’:
: undefined reference to ‘sqrt’
collect2: ld はステータス 1 で終了しました
```

4.4.2 標準入力関数の戻り値の利用

scan 関数の戻り値は、書式指定どおり正常に読み込んだ変数の数である。これを利用して、書式指定に適合しない入力をあたえることによって 逐次入力を終了する ことができる。つぎの例 ([8]) を見てみよう。

例 2

```
/* sum4.c */
/* 整数列の和を計算する。*/

#include <stdio.h>

main()
{
   int i, s = 0;

   printf("整数を逐次入力してください。その最終的な和をもとめます。\n");
   printf("終了するときは整数以外の文字を入力してください: ");

   while (scanf("%d", &i) == 1)
   {
      s += i;
   }

   printf("和は %d です。\n", s);
}
```

実行例

```
$ ./a.out
整数をあたえてください。和をもとめます。
2
3
5
8
30
200
a
和は 248 です。
```

この例では、ちゃんと整数を読み込むと scanf は 1 を返す。すると、scanf("%d", &i) == 1 は「真」となるので、while ループはつづく。最後に整数ではなく、文字「a」をあたえることによって、入力を終了している。もし、ふたつの整数を読み込むときは、うまく読み込むと scanf の戻り値は 2 となる。

♣ 演習問題 11 ♣

キーボードからふたつの自然数 a, b をあたえる。このとき、$\sqrt{a \cdot b}$ の整数部をもとめよ。数学関数を駆使して解け。入力が適合しないときは、終了するようにせよ。

4.4.3 ほかのライブラリ関数の例 1: 乱数生成

ほかのライブラリ関数をつかうときも、おなじように、初めに然るべきヘッダファイルを取り込んでおけばよい。

ふたたび、サイコロ賭博の例 (§ 3.3.1 例 2、および、§ 4.2.4 参照) をつかおう。こんどは乱数生成プログラムが自動的にサイコロの目を用意してくれるという代物である。これでムードが出るだろうか。

乱数生成にかんしては、[9, 14] を参考にした。また、時間で種をセットすることは、[14] に依った。さらに、名誉 TA 密井さんの協力も得た。

例 3

```
/* saikoro3.c  */

#include <stdio.h>
#include <stdlib.h> /* srand, rand */
#include <time.h> /* time  */

/****** サイコロを振る関数 *******/
int dice(void)
{
   int number;

   number = rand() % 6 + 1;

   return number;
}
/****** 偶奇をきめる関数 *******/
int even_odd(int a, int b)
{
    int c;

    c = (a + b) % 2;

    return c;
}

/****** 丁半を出力する関数  ******/
void cho_han(int a, int b)
{
   int c1;

   c1 = even_odd(a, b);

   if (c1 == 0)
   {
      printf("%d %d の丁\n", a, b);
```

```
   }
   else
   {
      printf("%d %d の半\n", a, b);
   }
}

/****** 主関数   ******/
main()
{
   int a, b;   /* サイコロの目  */
   int seed;     /* 乱数の種 */
   int i, n;   /* i: カウンタ、n: ぞろ目の総数 */

   printf("サイコロ賭博をします。\n");
   printf("gcc が自動でサイコロを連続的に振り、丁か半かを出力します。\n");
   printf("ぞろ目が n 回出れば終了します。\n");
   printf("n をあたえてください。\n");
   scanf("%d", &n);
   printf("\n");

   seed = (unsigned int) time(NULL); /*時刻によって乱数の種を決める  */
   printf("seed: %d\n", seed);
   srand(seed); /* 乱数の種をセットして、乱数生成プログラムを初期化する。 */

   for (i = 1; i <= n; i++)
   {
      do
      {
         a = dice();
         b = dice();

         cho_han(a, b);
      }
      while (a != b);

      printf("    ↑                \n");
      printf("%d 回めのぞろ目です。\n\n", i);
   }
}
```

実行例

```
$ ./a.out
サイコロ賭博をします。
gcc が自動でサイコロを連続的に振り、丁か半かを出力します。
```

ぞろ目が n 回出れば終了します。
n をあたえてください。
3

seed: 1215597947
6 4 の丁
2 3 の半
1 6 の半
4 1 の半
1 1 の丁
    ↑
1 回めのぞろ目です。

4 1 の半
3 4 の半
1 1 の丁
    ↑
2 回めのぞろ目です。

5 2 の半
4 1 の半
5 4 の半
2 6 の丁
6 3 の半
4 1 の半
3 3 の丁
    ↑
3 回めのぞろ目です。

もちろん、何回でもいけるのであるが、紙面の節約もある。また、仁侠映画などでは、ぞろ目のときには、「1 1 の丁」とはいわずに、もっと粋な表現をしているようである。以上の二点、ここではこの程度で容赦してほしい。

4.4.4 ほかのライブラリ関数の例 2: 文字関数

つぎは、文字クラスの判別を調べよう。ここでは、§ 3.5.3 の連続的なアルファベット文字入力を、大文字と小文字の区別を込めて取りあげる。

例 4

```
/*  char_counter3.c */

#include <stdio.h>
#include <ctype.h>

main()
```

```
{
   char key;  /* キーボードから入力する文字 */
   int n;  /* 文字の総数。大文字と小文字の数の和。*/
   int n_upper = 0, n_lower = 0; /* 大文字と小文字の数。 */

   printf("キーボードから文字を連続入力してください。\n");
   printf("終了したいときは、'q'、または'Q' です。\n");

   do
   {
       scanf("%1s", &key); /* 空白キーを流すために、%1s をつかっている。*/

       if (isupper(key) != 0 && key != 'Q') /* 大文字のとき   */
       {
          n_upper++;
       }
       else if (islower(key) != 0 && key != 'q') /* 小文字のとき   */
       {
          n_lower++;
       }



   }
   while  (key != 'q' && key != 'Q');

   n = n_upper + n_lower; /* 大文字と小文字の数の和をとる。*/

   printf("ゲーム終了！\n");
   printf("あなたの入力した文字は 'Q(q)' をのぞいて%d 個です。\n", n);
   printf("そのなかで、大文字は%d 個、小文字は%d 個です。\n", n_upper, n_lower);
}
```

実行例 1

```
$ ./a.out
キーボードから文字を連続入力してください。
終了したいときは、'q'、または'Q' です。
A
B
c
d
e
F
g
q
ゲーム終了！
あなたの入力した文字は 'Q(q)' をのぞいて 7 個です。
```

そのなかで、大文字は 3 個、小文字は 4 個です。

文字を一気に入力してもおなじ結果を得る。

実行例 2

```
$ ./a.out
キーボードから文字を連続入力してください。
終了したいときは、'q'、または'Q' です。
ABcdeFgq
ゲーム終了！
あなたの入力した文字は 'Q(q)' をのぞいて 7 個です。
そのなかで、大文字は 3 個、小文字は 4 個です。
```

isupper, islower は、<ctype.h>に入っているので、最初にこのライブラリーを読み込んでいる。isupper は、入力文字が大文字なら非零を、そうでなければ 0 を返す。おなじように、islower は、入力文字が小文字なら非零を、そうでなければ 0 を返す [15]。最後の「Q(q)」を勘定しないように、それぞれ、「key != 'Q'」、および 「key != 'q'」を入れてあるのは言うまでもないことである。

参考文献

[1] B. R. Kernighan & D. M. Ritchie (石田 晴久　訳)『プログラミング言語 C・第 2 版 (The C programming Language)』（共立出版、1989 年)。関数の記述は、おもに、第 1 章 § 1.7、および、第 4 章。

[2] J. May & J. Whittle (武舎 広幸　訳)『Symantec C++ for the Macintosh トレーニングブック (Symantec C++ for the Macintosh)』（翔泳社、1994 年）第 6 章。

[3] L. Ammeraal (小山 裕徳　訳) 『C で学ぶデータ構造とプログラム (Programs and Data Structures in C)』（オーム社、1995 年)。

[4] [3] の付録 A。

[5] M. A. Arbib, A. J. Kfoury and R. N. Moll（甘利・金谷・嶋田　訳)『計算機科学入門 (A Basis for Theoretical Computer Science)』、（サイエンス社、1984 年） pp 37-41 and pp 101-105。

[6] L. Goldschlager and A. Lister（武市・小川・角田　訳)『計算機科学入門 —第 2 版 (Computer Science)』、（近代科学社、2000 年） pp 49-58。

[7] P. H. Winston (大鋳 史男、鬼頭 繁治 訳) 『ウィンストンの C (on to C)』（ピアソン・エデュケーション、1995 年)。

[8] [3] の § 1-4。

[9] D. マーク『Windows ではじめる C プログラミング』（星雲社、2001 年)。

[10] [7] の § 15。

[11] [7] の § 19-23。

[12] [1] の pp 30-33。

[13] 都倉 信樹『プログラミング入門』（放送大学教育振興会、2000 年)。第 8 章。pp177-78。

[14] 平林 雅英 『新 ANSI C 言語辞典』(技術評論社、1997 年)。

[15] [1] の pp 202-203。