C言語によるプログラミング3:
プログラムの流れの制御

A Pedestrian Approach
to
the C Programming Language

目 次

図 目 次

表 目 次

3　プログラムの流れの制御

3.1　概論

プログラムは、主関数 (main) から始まり、とくべつな流れの制御の指示がなければ、上から順番に実行される (連続、逐次的実行)。プログラムの流れの制御の指示の種類としては、ある条件をみたすかどうかによって流れを分岐して選択させる (分岐、選択)、および、ある条件をみたすときに、プログラムのある部分を連続的に繰り返すもの (繰り返し、ループ) がある。

いかなるプログラムも、上記の (i) 連続 (ii) 分岐・選択 (iii) 繰り返し (ループ) の 3 種類をつかって構成することができる [1] (図 1 参照)。これは、現在のところ、プログラム言語の種類に依らない、アルゴリズムにかんする普遍的な結果である。

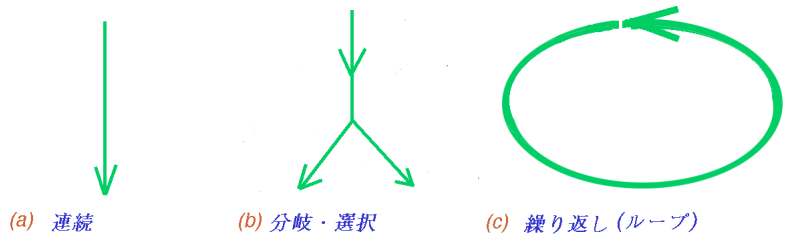

図 1: (a) 連続, (b) 分岐・選択, (c) 繰り返し (ループ)

※注意

ただし、ごく最近までの情報科学で普遍的と考えられていたことのなかで、これからは修正の必要なことも出てくる可能性がある。現在のコンピュータでは暗黙の内に情報を古典的にあつかう (0, 1 の 2 状態 = 1 bit を単位とする) ことを仮定している。しかし、量子力学にもとづいて情報をあつかう (0, 1 の 2 状態の重ね合わせ = 1 qubit[quantum bit の略記] を単位とする) コンピュータも最近、素子の開発や理論の両方で、考えられている [2, 3, 4]。これは、ひとつにはより速いコンピュータの必要性からである。しかし、量子コンピュータにおける重要な概念のひとつが、A. Einstein (アインシュタイン)、D. Bohm (ボーム)、J. S. Bell (ベル) 等による、本来まったく応用とは関係のない、量子力学の提示する自然観にたいする問いかけから発していることは注目に値する [5]。いずれにせよ、すぐに量子コンピュータが実用段階に入ることもない。また、たとえそうなっても、いま学んでいることも必要である。安心してこの授業に参加されよ。

実際のプログラムでは、(i) 連続 (ii) 分岐・選択 (iii) 繰り返し (ループ) の 3 種類を組み合わせて用いることになる。いくつかの例を模式的に挙げてみることにする (図 2 参照)。

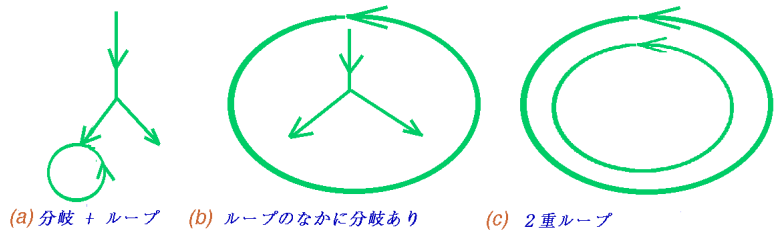

図 2: 基本制御の組み合わせ

3.2 プログラムの流れを制御する基本要素 (枠組のみ)

ここでは、(ii) 分岐・選択 (iii) 繰り返し (ループ) の各場合について、C 言語のソースコードによる記述の枠組みをまとめておく [7, 8, 9]。これは、いわば、絵のない額縁である。つぎのページから、それぞれの場合を具体的な例でみていくことにしよう。

1）二者択一

[い] if

```
if ( 式 )
{
    文;
}
```

* 式が真のとき、
文を実行する。

[ろ] if-else

```
if (式 1)
{
        文 1;
}
else
{
        文 2;
}
```

*else 以下がなければ [い] と同じ。

2）三つ以上のものからの選択

[い] else-if

```
if (式 1)
{
          文 1;
}
else if (式 2)
{
          文 2;
}
  ·····
else
{
          文 n;
}
```

[ろ] switch

```
switch (式)
{
    case 定数 1:
       文 1;
       break;
    case 定数 2:
       文 2;
       break;
    ·················
    default:
       文 n;
       break;
}
```

3）繰り返し（ループ）

[い] while

```
while (式)
{
       文;
}
```

* 式が真であるかぎり、
文を繰り返す。

[は] do-while

```
do
{
  文;
} while (式);
```

[ろ] for

```
for (式 1; 式 2; 式 3)
{
      文;
}
```

*これは以下とおなじ。

```
式 1;
while (式 2)
{
     文;
     式 3;
}
```

3.3 分岐

3.3.1 if-else

●基本的な枠組み

```
if（式１）
{
        文 1;
}
else
{
        文 2;
}
```

if 文では、括弧『(』、『)』のなかの「式 1」が真（true) のときに、「文 1」が実行される。「式 1」が偽（false) のときには、if ブロック (「{ }」) には入らず、else ブロックのほうに進み、「文 2」を実行する。ここで、真というのは、「式 1」の値が 0 以外のときで、偽は「式 1」の値が 0 になるときである。ここでの各ブロック中の「文」は、それぞれ、ひとつのこともある (単文)。また、複数の文から成る場合もある (複文) 。
※ 他のプログラミング言語で真偽の判定用に『論理型 (boolean)』の変数を設定しているものもある。この場合、上記のような値 (0 とそれ以外) で判定するのではないことに注意が必要である。(たとえば、[11, 13] を参照のこと。)

例 1　絶対値 (absolute value) をもとめる (図 3 参照) [8]。

```
/*   abs1.c   */
#include <stdio.h>

main()
{
      int a, b;

      printf("あたえられた整数 a の絶対値をもとめます。\n");
      printf("整数 a を入力してください。a: ");
      scanf("%d", &a);

      if (a <  0)   /*   a が負のとき   */
      {
              b = - a;
      }
      else         /*   a が正、または零のとき   */
      {
              b = a;
      }

      printf("| a | = %d\n", b);
}
```

図 3: if-else による分岐

実行例

```
$ g++ abs1.c
$ ./a.out
整数 a の絶対値をもとめます。
整数 a をあたえてください。a: 10
| a | = 10
$ ./a.out
整数 a の絶対値をもとめます。
整数 a をあたえてください。a: 0
| a | = 0
$ ./a.out
整数 a の絶対値をもとめます。
整数 a をあたえてください。a: -3
| a | = 3
```

****

else ブロックの省略

else ブロックは省略することができる。おなじく絶対値をもとめる、つぎの例を見よう [8]。(図 4 参照。)

例 1a

```
/*   abs2.c   */

#include <stdio.h>

main()
{
      int a;

      printf("あたえられた整数 a の絶対値をもとめます。\n");
      printf("整数 a を入力してください。a: ");
      scanf("%d", &a);

      if (a <  0)
      {
              a = - a;
```

```
    }

    printf("| a | = %d\n", a);
}
```

図 4: if による分岐

※ 例 1 ～ 例 2 のような機能をもった、整数の絶対値をもとめる関数 abs，labs が、標準ライブラリーに入っている。また、実数版の fabs もある [14]。関数にかんしては、本書ではこの後の § 4 に記述している。

元にもどって、もうひとつ if-else の例をあげよう。サイコロ賭博でよくある風景である。ほんらい乱数などでサイの目をきめるべきである。しかし、ここでは、いかさまの意図はないが、手動である。また、丁・半の分類だけで、「ぞろ目」の場合をとくに区別はしていない。(後述の演習問題 5 の 2 参照。)

例 2　丁か半か？

```
/*   saikoro1.c   */

#include <stdio.h>

main()
{
   int a, b, c;

   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);  /* ふたつのサイコロの目の数の標準入力 */

   c = a + b; /* サイコロの目の数の和 */

   if ((c % 2) == 0) /* 目の数の和が偶数のとき*/
   {
      printf("%d, %d の「丁」\n", a, b);
   }
   else /* 目の数の和が奇数のとき*/
   {
      printf("%d, %d の「半」\n", a, b);
   }
}
```

実行例

```
$ g++ saikoro1.c
$ ./a.out
```

```
サイコロの目をふたつ、空白を置いて、あたえてください。2 6
2, 6 の「丁」
$ ./a.out
サイコロの目をふたつ、空白を置いて、あたえてください。3 4
3, 4 の「半」
$ ./a.out
サイコロの目をふたつ、空白を置いて、あたえてください。1 1
1, 1 の「丁」
```

****

いろいろな演算子

よくつかう演算子にはつぎのようなものがある [7, 8, 9]。

算術演算子 (Arithmetic Operators)

`+, -, *, /`　　加減乗除
(※ ただし、整数どうしの演算のときには、割り算 (`/`) の結果は商を表わすことに注意されたい。)
`a % b`　　モジュロ演算子。整数 a を整数 b で割った余りを表わす。
具体的に試してみるとわかるが、ふつうの「余り」とはかならずしも一致しない。余りが負となることもある。いわゆる絶対値最小剰余 [15] となっているかもしれない。この点、演習問題 8 の 5 も参照のこと。
(※ 書式指定のときの % とは関係ない。)

関係・等値演算子 (Relational and Equality Operators)

`a > b`　　a is greater than b
`a >= b`　　a is greater than or equal to b.
`a < b`　　a is less than b.
`a <= b`　　a is less than or equal to b.
`a == b`　　等しい（イコール, a is equal to b.）
（※代入演算子『=』との混同に注意 !!! (Be careful not to use '=' in place of '==', and vice versa.）
`a != b`　　等しくない (a is not equal to b.)

論理演算子 (Logical Operators)

`!a`　　論理的否定（真、偽を反転させる。非零→零、零→非零。logical negation)
※以下のふたつは左から評価する.
`(a == b) && (c == d)`　　かつ (logical AND)
`(a == b) || (c == d)`　　または (logical OR)

if の入れ子 (nest)

if の連続する場合で、if-else の else ブロックの省略されたときは注意を要する。このとき、else は直前の if に対応することになる [7]。つぎの例で、整数として 10 や 20 をあたえると、「奇数です」という答えを得てしまう。

例 3

```
/* multiple.c  */

#include <stdio.h>

main()
{
    int number;

    printf("正の整数をあたえてください。(Enter a positive integer.)\n");
    scanf("%d", &number);

    if ((number % 2) == 0)

      if ((number % 3) == 0)
      printf("6 の倍数です。(The integer is divided by 6.)");
```

実行例

```
$ g++ multiple.c
$ ./a.out
正の整数をあたえてください。(Enter a positive integer.)
12
6 の倍数です。(The integer is divided by 6.)
$ ./a.out
正の整数をあたえてください。(Enter a positive integer.)
10
奇数です。(The integer is odd.)
$ ./a.out
正の整数をあたえてください。(Enter a positive integer.)
15
$ ./a.out
正の整数をあたえてください。(Enter a positive integer.)
20
奇数です。(The integer is odd.)
```

****

このような混乱をさけるためには、分岐の部分のコードを

```
    if ((number % 2) == 0)
```

```
    {
      if ((number % 3) == 0)
      {
         printf("6 の倍数です。(The integer is divided by 6.)");
      }
    }
    else
    {
      printf("奇数です。(The integer is odd.)\n");
    }
```

というように、『{』、『}』をつかってブロックをつくって、else がひとつめの if に対応することを明確にするとよい。また、ブロック内の文がひとつのときでも、『{』、『}』をつかっておけば、この種の混乱を未然にふせぐことができる [7, 8]。

3.4 分岐 2: 多重分岐

3.4.1 else-if

●基本的な枠組み

```
if (式 1)
{
          文 1;
}
else if (式 2)
{
          文 2;
}
   ·····
else
{
          文 n;
}
```

if-else での分岐が多重化したもの。カスケード式に上から条件をみたすブロックを実行していく。すなわち、「式 1」が真のときは「文 1」を実行する。偽ならつぎにすすむ。つぎに、「式 2」が真のときは「文 2」を実行する。偽ならつぎにすすむ。以下同様である。最後に、「式 1」～「式 (n - 1) 」 のどれにもあてはまらないときには、else ブロック内の「式 n」を実行する。

例 1　2 と 3 の剰余による整数の分類。

```
/* 三つ以上のものからの選択 (else-if)   multiple1.c   */

#include <stdio.h>

main()
```

```
{
    int a;

    printf("整数をあたえてください。\n");
    scanf("%d", &a);

    if ( (a % 2 == 0) && (a % 3 == 0) )
    {
        printf("%d は 6 の倍数です。\n", a);
    }
    else if ( (a % 2 == 0) && (a % 3 != 0) )
    {
        printf("%d は 2 の倍数ですが、3 の倍数ではありません。\n", a);
    }
    else if ( (a % 2 != 0) && (a % 3 == 0) )
    {
        printf("%d は 3 の倍数ですが、2 の倍数ではありません。\n", a);
    }
    else
    {
        printf("%d は 2 の倍数でも 3 の倍数でもありません。\n", a);
    }
}
```

実行例

```
$ g++ multiple1.c
$ ./a.out
整数をあたえてください。
12
12 は 6 の倍数です。
$ ./a.out
整数をあたえてください。
15
15 は 3 の倍数ですが、2 の倍数ではありません。
$ ./a.out
整数をあたえてください。
20
20 は 2 の倍数ですが、3 の倍数ではありません。
$ ./a.out
整数をあたえてください。
23
整数をあたえてください。
23
23 は 2 の倍数でも 3 の倍数でもありません。
```

```
*****************************
```

プログラムの流れの概略を図示するとつぎのようになる。ここで、 E はプログラムの分岐のしかたを決める変数である。また、$a_1, \cdots$ は E の分岐のしかたを決める条件式、あるいは式の値、$S_1, \cdots$ はそれぞれの場合におこなわれるつぎの処理を表わしている。([6] 参照。)

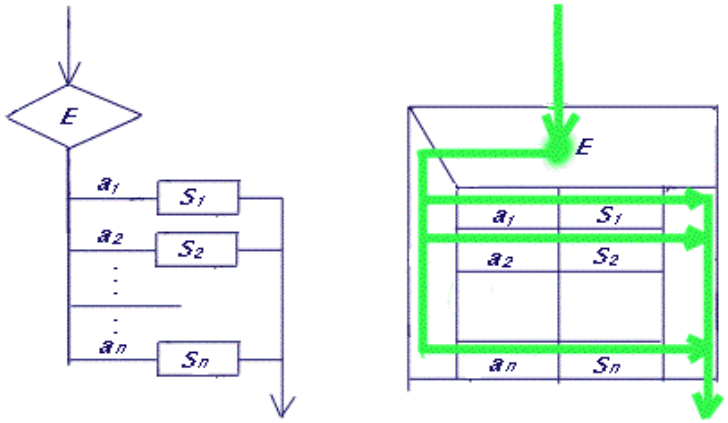

図 5: 多分岐

3.4.2 switch

●基本的な枠組み

```
switch (式)
{
    case 定数 1:
       文 1;
       break;

    case 定数 2:
       文 2;
       break;
    ..................
    default:
       文 n;
       break;
}
```

switch 文は、指定した式の値がいくつかの定数の整数値のひとつと一致しているかを判断し、その整数値をラベルにして多重分岐する。さっきとおなじ問題を switch をつかって試してみよう。

例 2

```
/*  三つ以上のものからの選択 (switch) */
/*   multiple2.c  */
```

```
#include <stdio.h>

main()
{
   int a, rem_sum;

   printf("自然数をひとつあたえてください。: ");
   scanf("%d", &a);

   rem_sum = (a % 2) + 2 * (a % 3);
 /* （あたえられた数を２でわったあまり）＋  ２ ×（あたえられた数を３でわったあまり） */

   switch (rem_sum)
   {
      case 0:
         printf("%d は 6 の倍数です。\n", a);
         break;
      case 2:
      case 4:
         printf("%d は２の倍数ですが、３の倍数ではありません。\n", a);
         break;
      case 1:
         printf("%d は３の倍数ですが、２の倍数ではありません。\n", a);
         break;
      case 3:
      case 5:
         printf("%d は２の倍数も３の倍数でもありません。\n", a);
         break;

      default:
         break;
   }
}
```

プログラムの流れの概略は、先に挙げた図 5 に準ずる。

ここでは、先の else-if のときとおなじ剰余による類別を実現するために、変数 rem_sum を上記のように、a % 2 と a % 3 の重率をつけた和として導入したことに注意。単純な和をとったのでは完全に場合を分けることはできないからである。

※ ある箇所の break をはずすとどうなるか。実験してみるとおもしろい。
※ できるかぎり switch をつかうほうがよいであろう。プログラムがみやすくなるとおもう。式と定数を比較するときは switch をつかい、式と式を比較するときは if-else をつかえばよい [8]。

♣ 演習問題 4 ♣

1) 三つの実数をあたえたとき、これらを３辺とする三角形が成立するか、を調べるプログラムをつくれ。
2) キーボードから整数をふたつあたえて、おおきいほうから出力せよ。

これは、具体的には、つぎのような意味である。すなわち、ふたつの整数 a, b をあたえる。このとき、$a < b$ のとき、メモリ空間のなかで、これらを交換せよ。(a, b の箱に入っているビットの列を交換する、ということ) しかる後に、a, b をこの順に印字せよということである。

※ 印字の順番を変える (`printf("%d %d", a, b)` を `printf("%d %d", b, a)` とする) のではない。もちろん、このやりかたでも、端末への出力はおなじである。しかし、ここでは、ビットの列自体を入れ換えること、また、変数の上書きを防ぐような解きかた、の演習を目的としている。(§ 3.5.2 の例 2a は本問と関連している。)

3) キーボードから正の整数を 3 個あたえる。三数を $a.\ b, p$ としておく。a, b をそれぞれ p で割ったあまり (remainder) をおおきいほうから出力せよ。

4) キーボードから 16 進表示の 1 桁の整数をあたえて、10 進表示で出力せよ [8]。ただし、`%x` は「禁じ手」とする。(ヒント: 入力を文字としてあつかえばよい。ASCII コードの表 [18] も参照のこと。)

♣ 演習問題 5 ♣

1) ★今は昔の 2000 年スペシャル★

西暦年をあたえて、その年が閏年かどうか、をしらべよ。これを、つぎの三通りのやりかたで解け。

い）論理演算子を組み合わせて、一度の if でおこなえ。

ろ）論理演算子をつかわずに、if, else の組み合わせのみでおこなえ。

は) 論理演算子をつかわずに、条件を整理して [19]、else-if をつかって解け。

2) ふたつのサイコロの目のプログラム (§ 3.3.1 の例 2 参照) について、「ぞろ目」の場合をとくべつに表示するように改造せよ。

♣ 演習問題 6 ♣

1) 何月かをあたえて、その月にふくまれる日数を出力せよ。平年としてよい。

2) 西暦年と月をあたえて、その月にふくまれる日数を出力せよ。

3.5 繰り返し (ループ)

1 から n までの整数の和をコンピュータをつかってもとめてみよう。

ある i $(i = 0, \cdots, n)$ にたいして、

$$S_i = \overbrace{\underbrace{1 + 2 + \cdots + (i-1)}_{S_{i-1}} + i}^{S_i} \qquad (*)$$

とおくと、S_n をもとめればよいことがわかる。

どのような変数をもちいればよいだろうか。S_i $(i = 0, \cdots, n)$ にたいしてひとつの変数 (以下のプログラム例では、これを sum としている) をあてがって（bit の列をいれる箱を用意するということ)、$i = 1$ から $i =$ n まで $(*)$ をくりかえしもちいればよい。もちろん、カウンタ用の変数 i も必要である。
すなわち、
$S_0 \equiv 0$ として、
$S_1 = S_0 + 1 = 1$
$S_2 = S_1 + 2 = 3$
$S_3 = S_2 + 3 = 6$
$\cdots$
のようになるので、S_i $(i = 0, \cdots, n)$ を収容する箱 sum の内容は、順次 $1, 3, 6, \cdots$ となる。
プログラムをかくとつぎのようになる。

3.5.1 while

●基本的な枠組み

```
while (式)
{
   文;
}
```

「式」が真であるかぎり、「文」を繰り返す。論理値が真であるかの判定は、「文」を実行するまえにおこなわれる。「式」はいつかは偽になるように設定しなければならない。そのためには、「文」が「式」にふくまれる変数の状態を変化させるようになっている必要がある。そうでないと「無限ループ (どうにもとまらない状態)」に陥ってしまう [8]。gcc の場合、そのときには、強制終了 (端末から C-c) することになる。

例 1

```
/* 1 から n までの整数の和 Ver. 1  while-loop */
/* sum1.c */

#include <stdio.h>

main()
{
  int i, n;           /*  変数の型宣言  */
  int sum = 0; /* 変数の型宣言および初期化   */

  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。");
  scanf("%d", &n);  /*   キーボードから入力（標準入力）   */

  i = 1;
  while (i <= n)   /*  ( ) のなかの条件のもとで、以下のループをくりかえす  */
  {
    sum += i;        /* sum = sum + i という意味  */
    printf("1 から %d までの和 = %d\n", i, sum);      /*  画面への出力（標準出力） */

    i++;   /* i = i + 1; という意味  */
   }
}
```

プログラムの流れの概略を図示すると次頁の図 6 のようになる。([6] 参照。)

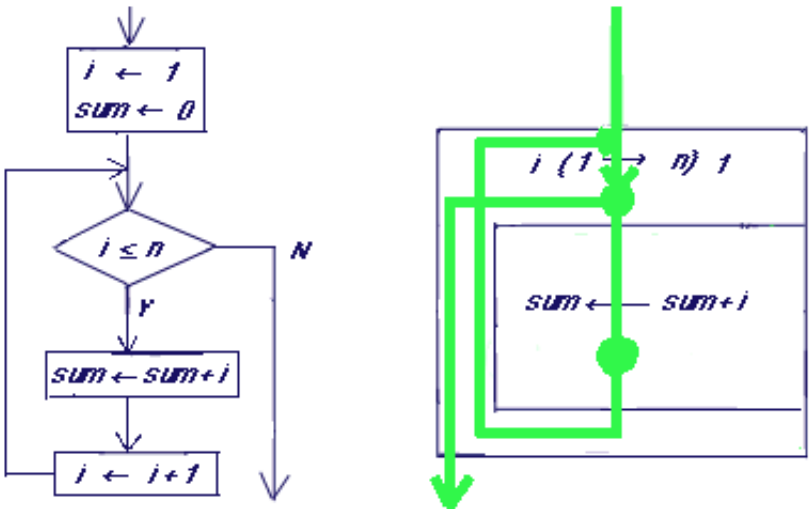

図 6: 繰返し (前判定ループ)

$n = 3$ のときの各ループでの変数の状態は、 次頁の図 7 のようになる。これは、つぎの for をつかったときもおなじである。

図 7: $n = 3$ のときの、ループの各周における変数 sum, i の状態

実行例

```
$ ./a.out
1 から n までの整数の和をもとめます。
整数 n を入力してください。5
1 から 1 までの和 = 1
1 から 2 までの和 = 3
1 から 3 までの和 = 6
1 から 4 までの和 = 10
1 から 5 までの和 = 15
```

3.5.2 for

●基本的な枠組み

```
for (式 1; 式 2; 式 3)
{
    文;
}
```

*これは以下とおなじ。

```
式 1;
while (式 2)
{
      文;
      式 3;
}
```

for ループは while ループと似ている。後者では初期化、条件の判定、ループカウンターの更新を一行に書くことができるので、プログラムがひじょうにわかりやすくなる [8]。

例 2

```
/* 1 から n までの整数の和 Ver. 2  for-loop */
/* sum2.c  */

#include <stdio.h>

main()
{
   int i, n;           /*  変数の型宣言  */
   int sum = 0; /* 変数の型宣言および初期化   */

   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。");
   scanf("%d", &n);  /*   キーボードから入力（標準入力）  */

   for (i = 1; i <= n; i++)   /*  ( ) のなかの条件のもとで、以下のループをくりかえす  */
   {
     sum += i;     /* sum = sum + i という意味  */

   }
   printf("1 から %d までの和 = %d\n", n, sum);     /*  画面への出力（標準出力） */
}
```

実行例

```
$ g++ sum2.c
$ ./a.out
1 から n までの整数の和をもとめます。
整数 n を入力してください。5
1 から 5 までの和 = 15
```

「例 1」の while ループのときとの出力のちがいに注意のこと。「例 2」の for ループのソースコード内の「printf」の位置が「例 1」のそれからすこしずれていることに気づくであろう。

※ 注意点 (for ループと while ループの使い分け)[8]
・カウンタなどを用いるときは、for ループを使うほうがわかりやすい。

・ループの終了条件が変数の比較だけのときは、while ループを使うとよい。

＊＊＊＊＊＊＊＊＊＊＊

さて、例 1、例 2 では、ループをまわるごとに、カウンタ変数 i がひとつすすんだ。だが、ループの周回に応じて、カウンタ変数がふたつすすんでもよいのはあきらかである。また、減少してもよいのも同様である。その辺は、状況にしたがって改造すればよい。

また、例 1、例 2 ではループをまわるごとに、変数 sum は上書きされた。このとき、新らしい sum に旧い sum と旧い i が関与した。だが、新規の i はまえの i だけできまった。

しかし、くりかえしのプログラムでも、ループの周回に応じて、ふたつ以上の変数が互いに上書きされるときは、すこし注意を要する。以下の例で調べてみよう。

例 2a

つぎの差分方程式の解 $\{(a_i, b_i)\}$ をもとめる [21]。

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

また、数列 $\{u_i\}$ $(u_i \equiv a_i^2 - 2b_i^2)$ も調べる。

```
/* pell1.c */

#include <stdio.h>

main()
{
   int i, n;  /* i: 数列の添字。n: 項数。*/
   int a = 1, b = 1; /* 数列 a(i), b(i) の実体。初項は a(1)= b(1) = 1。*/
   int temp;  /* ループをまわるとき、a の値を一時的に格納するための変数。*/
   int u ; /* u(i) ≡ a(i)^2 - 2 * b(i)^2 のための変数。*/

   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");
```

```
   printf("  1\t (         1,          1)  \t  -1\n"); /*  n = 1 のとき。  */
   for (i = 2; i <= n; i++)  /*  n >= 2 のとき。  */
    {
        temp = a; /* a の箱のなかのビットの列を、一時的に temp の箱に格納する。*/
        a = a + 2 * b;  /* a の値はあたらしくなった。  */
        b = temp + b; /* temp の箱に格納されている元の a の値をつかう。  */

        u  = a * a - 2 * b * b;

        printf("%3d\t (%10d, %10d)\t %3d\n", i, a, b, u);
   }
}
```

実行例

```
$ ./a.out
差分方程式
a(1) = b(1) = 1
a(i + 1) = a(i) + 2 * b(i)
b(i + 1) = a(i) + b(i)
の i =1 から n までの解をもとめます。
また、u(i) ≡ a(i)^2 - 2 * b(i)^2 と表記します。
n をあたえてください。13


  i         (a(i), b(i))       u(i)
================================================
  1  (         1,          1)   -1
  2  (         3,          2)    1
  3  (         7,          5)   -1
  4  (        17,         12)    1
  5  (        41,         29)   -1
  6  (        99,         70)    1
  7  (       239,        169)   -1
  8  (       577,        408)    1
  9  (      1393,        985)   -1
 10  (      3363,       2378)    1
 11  (      8119,       5741)   -1
 12  (     19601,      13860)    1
 13  (     47321,      33461)   -1
```

注意

（い）`b = a + b;` とすると a のあたらしい値 (a_i ではなく a_{i+1}) をつかうことになる。すなわち、$b_{i+1} = a_{i+1} + b_i$ を計算することになってしまう。直前に `a = a + 2 * b;` としたとき、すでに、a は上書きされてしまっているからである。

そこで、変数 temp に格納していた元の a の値 (a_i) を取り出してきて、b = temp + b; としている。この点にかんして、演習問題 4 の 2)((p 31)) を再訪するとよい。
（ろ）「\t」は「タブ (Tab)」キーをソースコード内で表現するもの。「\n(改行、Enter key)」などとおなじく制御用の特殊文字 (「エスケープ・シークエンス」[20]) のひとつである。%10d は、すくなくとも 10 文字幅で 10 進整数を表記する。

おまけ
・(♡) にしたがう $\{a_i\}, \{b_i\}$ にかんして、実行結果から予想もつくように、$a_i^2 - 2b_i^2 = \pm 1$ となる。すなわち、$(x, y) = (\{a_i\}, \{b_i\})$ は、$x^2 - 2y^2 = \pm 1$ の解となっている。しかし、おどろくべきことに、逆に $x^2 - 2y^2 = \pm 1$ の整数解は、上記の (♡) できまる $(x, y) = (a_1, b_1), (a_2, b_2), (a_3, b_3), \cdots$ で尽くされてしまうことが知られている [21, 22]。(さらにふかい文献には [15] がある。)
・実行例を表計算ソフト (OpenOffice 「Calc」や「Excel」をつかって確かめることもできる。

3.5.3 do-while

●基本的な枠組み

```
do
{
  文;
} while (式);
```

while とほとんどおなじだが、「式」が真かどうかの判定が最後におこなわれる。したがって、無条件に一度はループのブロックが実行される。

例 3

```
/* char_counter1.c */

#include <stdio.h>

main()
{
   char key;  /* キーボードから入力する文字 */
   int n = 0;  /*  キーボードから入力する文字の総数。 */
               /*  もちろん、はじめはゼロ個である。 */

   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' です。\n");

   do
   {
       printf("n = %d\n", n);   /*   現在の総文字数。    */
       scanf("%1s", &key); /* 空白キーを流すために、%1s をつかっている。*/
```

```
        n++;  /*    文字数をカウントする。     */
    }
    while  (key != 'q');

    printf("ゲーム終了！\n");
    printf("あなたの入力した文字は 'q' をのぞいて、%d 個です。\n", n - 1);
}
```

書式を%c とすると、scanf は空白文字 (Enter key など) も読み込んでしまうので、%1s(いちえす) をつかっていることに注意 [7]。

図 8: 繰り返し (後判定ループ)

※ 次頁の図 8 に、do-while でのプログラムの流れの図がある。図 8 で、緑の線 (グレースケールの印刷では灰色) は補助のためのもので、通常の NS チャートには記述しない。また、なかの表記もここでの目的に沿ったものにしていることに注意。(くわしくは、[6] 参照のこと。)

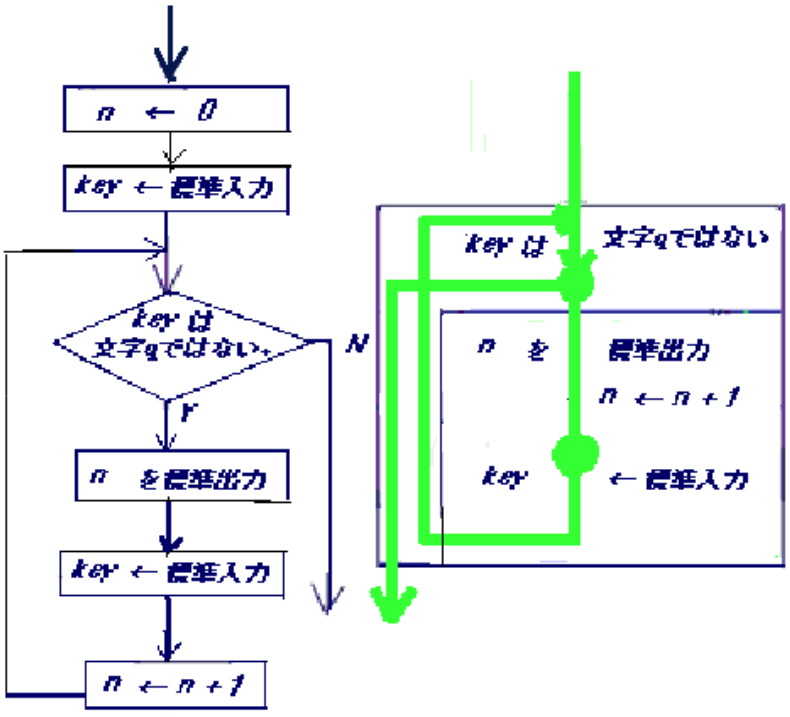

図 9: 繰り返し (前判定ループ)

以下の、while をつかった場合は、ループに入る条件として、ループに入る前に 1 文字分すでに文字を入力することに気をつけよう。また、前ページ図 9 に、while でのプログラムの流れの図がある。

例 3, 例 3a ともに、ループの終了条件 key != 'q ' は、カウンタ変数 n とは関係ないことに注意したい。ループの周回数ではなく、入力文字が「q」であるかどうか、によって、ループがつづくか終わるかがきまるのである。これは、 1 から n までの和 (例 1, 例 2)」や「差分方程式 (例 2a)」とは対照的である。ループの終了がいつもカウンタ変数に依るわけではないのである。

例 3a

```
/*  char_counter2.c */

#include <stdio.h>

main()
{
   char key;  /* キーボードから入力する文字 */
   int n;  /*  キーボードから入力する文字の総数。 */

   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' です。\n");

   scanf("%1s", &key);
   n = 1; /*  ループに入る前に 1 文字つかっている。*/
   while (key != 'q')
   {
       printf("n = %d\n", n);   /*   現在の総文字数。    */
       scanf("%1s", &key);
       n++;
   }

   printf("あなたの入力した文字は 'q' をのぞいて、%d 個です。\n", n);
   printf("ゲーム終了！\n");
}
```

♣ 演習問題 7 ♣
繰返しをつかって、つぎの問題を解け。ただし、以下の文字 n, r, p にあたる変数は、キーボードから (10 進整数として) 入力して初期化するものとする。

1)1 から n までの整数の 2 乗の和をもとめよ。
2)n の階乗をもとめよ。
3)n 個のものから r 個をとる順列の総数をもとめよ。
4)n 個のものから r 個をとる組み合わせの総数をもとめよ。
5)(i)3 の n 乗をもとめよ。

(ii)p の 5 乗をもとめよ。
(iii)p の n 乗をもとめよ。
6)n の最高位の桁の指数 h をもとめよ。すなわち、$10^{h+1} > n \geq 10^h$ となる h をもとめよ。
7)n を p 進数で表わしたとき、n の最高位の桁の指数 h をもとめよ。すなわち、$p^{h+1} > n \geq p^h$ となる h をもとめよ。
8)n の平方根の整数部をもとめよ。
$1 + 3 + \cdots + (2 \cdot p - 1) = p^2$ をつかってもよい。
9)Fibonacci(フィボナッチ)数列 $\{f_n\} = \{1, 1, 2, 3, 5, 8, 13, \cdots\}$ の n 項目をもとめよ。$\{f_n\}$ は、連続する 2 項を足し合わせたものがつぎの項になるように、つくられている。すなわち、$f_n = f_{n-1} + f_{n-2}$ が成り立つ。(Linux のプリントの § 5.3.3 の (ろ) も参照のこと。)
ヒント: ひとつのやりかたは、つぎの如きものである。まず、$\mathbf{u}_i = \binom{f_i}{f_{i-1}}$ のように、$\{f_n\}$ をふたつずつ組にする。すると、$\mathbf{u}_i$ と $\mathbf{u}_{i-1}$ の関係 (各成分がどうなるか) は、$\mathbf{u}_i = \mathsf{A}\mathbf{u}_{i-1}$ (ここで、A は、ある 2×2 行列) となる。これをつかうとよい。また、この方法で代数的に解くこともできる。ただし、プログラミングの場合、変数の上書きに注意する必要がある (この点、本節の例 2a のプログラムも参照のこと。)。
10) 負でない実数 x にたいして、その整数部分 $\lfloor x \rfloor$ をもとめよ。
例
・$x = 2.71828$ のとき
答え: $\lfloor x \rfloor = 2$。
・$n = 3.141592$ のとき
答え: $\lfloor x \rfloor = 3$。

※ 床記号 $\lfloor x \rfloor = x$ を越えない最大の整数、である。ここでは、かんたんのために、 $x \geq 0$ としている。もちろん、$x < 0$ のときも $\lfloor x \rfloor$ は存在する。たとえば、$\lfloor -3.141592 \rfloor = -4$ である。この床記号に対応する記号に、天井記号 $\lceil x \rceil (= x$ を越える最小の整数) がある。これらをあたえる関数 $\lfloor x \rfloor$、および $\lceil x \rceil$ はライブラリ関数に入っている (§ 4.4.1. 参照)。本問では、それとおなじような機能をもつプログラムを自前でつくろうとしているのである。

3.6 基本要素の複合例

いままでに制御の基本要素をみてきた。実際のプログラムでは、以上の基本要素を組み合わせてつかうことになる。もっとも、すでに、§ 3.3.1 の「if の入れ子」のところ【例 3 や演習問題 5 の 1) のろ)】で if、else を複合するような例はあった。

このような一般的な場合、プログラム全体の流れは以下のようになる。

一般的な場合のプログラム全体の流れ

(0) まず、§ 3.1 で述べたように、プログラムは、主関数から始まる。
その後の流れは、つぎの (i)〜(iv) に依る。
(i) 基本的には上から下へ進む。
(ii) 選択や繰り返しに当った場合、より外側のブロック（先に遭遇したブロック）の指示にしたがう。
(iii) つぎに、そのブロックの内でより外側のブロック（先に遭遇したブロック）の指示にしたがう。
(iv) 内側のブロックのコードが終われば、ひとつ外側のブロックに戻る。

具体的な例を見よう。

例 1

n をあたえて、1 から n までの整数を端末に対角的に印字しよう。ソースプログラムは、たとえば、つぎのようになる。

```
/*   diag1.c   */
/* 1 から n までの整数を端末に対角的に印字する。*/
/* 2 重ループをつかう。まず、下方向 (i-方向) の動きに外側のループを
もちいる。その i の各値にたいして、横方向 (j-方向) の変数の
ループを設定する。この 2 重ループのなかで、印字場所が対角線上にない場合は、
空白を印字して、横に一歩すすむ。対角線上 (i と j が等しい) にあるときは、
その i 値を印字する。これで、j-ループ 1 周分は終わり。
i-ループにもどる。ここで、改行して、i-ループ 1 周分は終わり。i をひとつふやして、
つぎの i-ループに入る。これを、n 回くりかえせばよい。*/
```

ソースプログラム

```
#include <stdio.h>

main()
{
   int i, j, n;

   printf("1 から n までの整数を対角的に印字します。\n");
```

```
   printf("n をあたえてください。:");
   scanf("%d", &n);

   for (i = 1; i <= n; i++)      /*   i-方向 (下方、「行」方向) に進む    */
   {
      for (j = 1; j <= i; j++)  /*   j-方向 (右、「列」方向) に進む    */
      {
         if (j == i)          /*  対角線上にあるとき    */
         {
            printf("%d", j);     /*  j = i を印字する    */
         }
         else                   /*  非対角なとき    */
         {
            printf(" ");      /*  「空白」を 1 文字印字する    */
         }
      }
      printf("\n");             /*  つぎの行に移る    */
   }
}
```

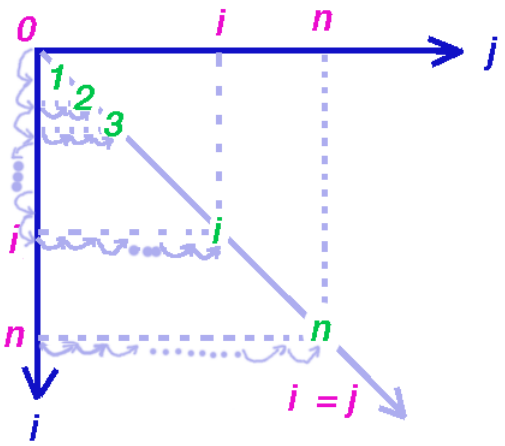

図 10: 対角線上に 1, ⋯, n を出力

実行例

```
$ g++ diag1.c
$ ./a.out
1 から n までの整数を対角的に印字します。
n をあたえてください。:5
1
 2
  3
   4
    5
```

このプログラムの流れは、 つぎの頁の図 11 ようになっているのがわかるであろう。

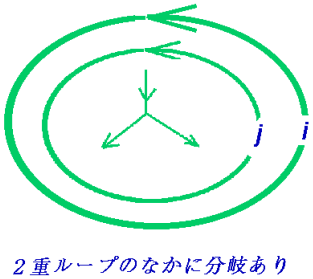

図 11: 「対角線上に 1, ⋯, n を出力」の全体の流れ

♣ 演習問題 8 ♣

1) 上の例で、$1, \cdots, n$ の順を逆にせよ。

2) 上の例で、$1, \cdots, n$ が右上から左下に並ぶようにせよ。

3) 中間レポートの温度変換の問題を連続試行できるように改造せよ。

4)x-y 平面上に、中心が原点 O、半径 r の円をあたえる。このとき、この円の内部 (周も込める) にある格子点をすべて書きだし、その総数 n を勘定せよ。また、n と円の面積との比をもとめよ。(π の値は適当にあたえればよい。) r がおおきくなったとき、この比はどうなるか?

5) 整数 a, b をあたえる。ただし、$b > 0$ とする。このとき、a を b で割ったときの商 q とあまり r をもとめよ。ただし、演算子『/』、および『%』は禁じ手とする。

例

・$(a, b) = (10, 3)$ のとき $\longrightarrow$ $(q, r) = (3, 1)$

・$(a, b) = (3, 8)$ のとき $\longrightarrow$ $(q, r) = (0, 3)$

・$(a, b) = (0, 2)$ のとき $\longrightarrow$ $(q, r) = (0, 0)$

・$(a, b) = (-13, 5)$ のとき $\longrightarrow$ $(q, r) = (-3, 2)$

0 は任意の整数 $b\,(b \neq 0)$ の倍数になっている ことに注意が必要である。

本問、および、つぎの 6), 7) にかんする基本的な文献として、[15, 23, 24, 25] などがある。もちろん、それらを参照しなくとも、問題を解くことができる。

6) 自然数 n をあたえる。このとき、1 から n までの数のなかで n と互いに素であるものをすべて書きだし、その総数φ(ファイ)(n)をもとめよ。

例

・$n = 3$ のとき

答え: 1, 2 $\varphi(3) = 2$。

・$n = 8$ のとき

答え: 1, 3, 5, 7 $\varphi(8) = 4$。

7) 自然数 n をあたえる。このとき、n の約数をすべて書きだし、その総和σ(シグマ)(n)をもとめよ。

例

・$n = 3$ のとき

答え: 1, 3 $\sigma(3) = 1 + 3 = 4$。

・$n = 8$ のとき

答え: 1, 2, 4, 8 $\sigma(8) = 1 + 2 + 4 + 8 = 15$。

8) 自然数 n をキーボードからあたえる。このとき、n を連続する自然数の和で表わせ。また、その表わしかたの総数は幾とおりあるかも記せ。さらに、このような表現のできない場合はその旨を記せ [26]。

例

・$n = 6$ のとき

答え: $n = 6 = 1 + 2 + 3$

の 1 とおり。

・$n = 7$ のとき

答え: $n = 7 = 3 + 4$

の 1 とおり。

・$n = 8$ のとき

答え: もとめる表現はない。

・$n = 15$ のとき

答え:
$$\begin{aligned} n = 15 &= 1 + 2 + 3 + 4 + 5 \\ &= 4 + 5 + 6 \\ &= 7 + 8 \end{aligned}$$

の 3 とおり。

・$n = 45$ のとき

答え:
$$\begin{aligned} n = 45 &= 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 \\ &= 5 + 6 + 7 + 8 + 9 + 10 \\ &= 7 + 8 + 9 + 10 + 11 \\ &= 14 + 15 + 16 \\ &= 22 + 23 \end{aligned}$$

の 5 とおり。

・$n = 105$ のとき

答え :
$$\begin{aligned} n = 105 &= 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12 + 13 + 14 \\ &= 6 + 7 + 8 + 9 + 10 + 11 + 12 + 13 + 14 + 15 \\ &= 12 + 13 + 14 + 15 + 16 + 17 + 18 \\ &= 15 + 16 + 17 + 18 + 19 + 20 \\ &= 19 + 20 + 21 + 22 + 23 \\ &= 34 + 35 + 36 \\ &= 52 + 53 \end{aligned}$$

の 7 とおり。

参考文献

[1] L. ゴールドシュレーガー & A. リスター（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）

[2] R. P. ファインマン (A. J. G. ヘイ & R. W. アレン 編、原 康夫, 中山 健, 松田和典 訳)
『ファインマン計算機科学 (*The Feynman Lectures on Computation*)』（岩波書店、1999 年)。第 6 章。

[3] 広田 修『量子情報科学の基礎—量子コンピュータへのアプローチ』、（森北出版、2002 年）

[4] M. Nielsen & I. Cuang 『Quantum Computation and Quantum Information』、（Cambridge Univ. Press, 2000）邦訳あり。

[5] J. S. Bell 『Speakable and Unspeakable in Quantum Mechanics (2nd edn)』、（Cambridge Univ. Press, 2004)

[6] 都倉 信樹『プログラミング入門』（放送大学教育振興会、2000 年)。

[7] B. W. カーニハン & D. M. リッチー (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)。

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

[9] P. A. Darnell & P. E. Margolis,『C — A Software Engineering Approach』3rd edn (Springer, New York, 1996)。

[10] 新井 潤『情報科学実習 II・テキスト』。

[11] [10] の No.2, p 4。

[12] P. H. ウィンストン & S. ナーラーシムハーン『ウィンストン の Java2(*On TO Java*)』(ピアソン・エデュケーション、2002 年)。

[13] [12] の § 21、および § 22。

[14] [7] の付録 B.4、および B.5。

[15] 高木 貞治『初等整数論 第 2 版』(共立出版、1971 年)。

[16] [7] 野付録 B4. と B.5。

[17] 授業資料 『C 言語によるプログラミング 1 & 2 (*A Pedestrian Approach to the C Programming Language*) 』

[18] [17] の p 17、§ 2.3.6 の表 2。

[19] 小川 貴英『Lisp プログラミング入門』(岩波書店、1989)。pp 16-17。

[20] [17] の § 1.3、p 4。および、§ 2.3.6、p 17。

[21] [23] の § 2.4。

[22] [25] の第 28, 29, 31 章。

[23] 一松 信『代数学入門第一課』(近代科学社、1992 年)。

[24] 片山 孝次『代数学入門』(新曜社、1981 年)。

[25] J. H. シルヴァーマン (鈴木 治郎　訳)『はじめての数論 (*A friendly Introduction to Number Theory*) 第 2 版』(ピアソン・エデュケーション、2001 年)。最新版 (第 3 版) も出版されているようである。

[26] [24] の p 30。