計算機基礎実習 I ・第 7 回

2017 年 6 月 2 日

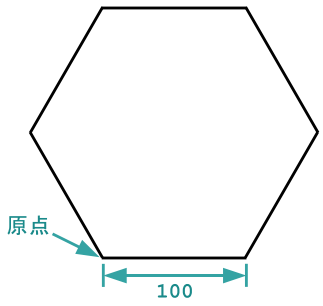
今回の内容

7.1 繰り返しの処理

これまで紹介したプログラムは、たとえ途中に場合分けの処理があったとしても、結局はプログラム中に書かれた一連の作業を書かれた順に実行することしかできませんでした。たとえば、1 辺の長さが 100 の正六角形を描くためには、次のプログラムのように `tForward(100);` や `tTurn(60);` という文を、何回もプログラム中に書いておくことが必要だった訳です。

```
#include <turtle.h>

main ()
{
    tForward(100);
    tTurn(60);
    tForward(100);
    tTurn(60);
    tForward(100);
    tTurn(60);
    tForward(100);
    tTurn(60);
    tForward(100);
    tTurn(60);
    tForward(100);
}
```



この例のように、コンピュータに実行してほしい作業を、実行してほしい回数だけプログラム中に繰り返し書いておくのは大変です。またこの方法ではプログラムの見かけの長さより複雑なことをプログラムにさせることができませんから、1 つのプログラムでできることは、そのプログラムの大きさで制限されてしまいます。

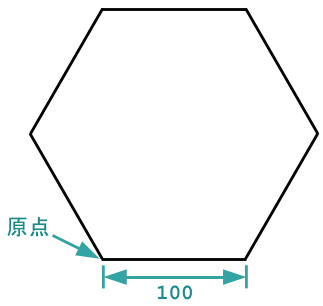
そこで、実用的なプログラミング言語には、作業の単純な繰り返しを簡潔に指示するために、また 1 つのプログラムにいろいろな大きさの仕事をさせるために、プログラム中のある場所に書かれて (指示されて) いる作業を何回か繰り返し実行できるような仕組みが用意されています。C では、これを `while` 文と呼ばれる書き方を使って利用することができます。たとえば次のようなプログラムで正六角形を描くことが可能です。

hexagon.c

```
#include <turtle.h>

main ()
{
    int i;

    i = 0;
    while (i < 6) {
```



```
        tForward(100);
        tTurn(60);
        i = i + 1;
    }
}
```

while 文の書き方　C の while 文は、一般に

```
while ( 条件式 )
    文
```

という形をしています。「条件式」が成り立っている限り「文」の部分が繰り返し実行されます。「条件式」の書き方は if 文の場合と全く同じです。条件式を囲っている (や) の両側には空白があってもなくてもかまいません。また、while 文を 1 行にまとめて

```
while ( 条件式 ) 文
```

のように書いても同じ意味になります。while という単語も if と同様に C の予約語となっていて、これを変数名などとして使用することはできません。

メモ

while 文の実行　while 文はつぎのような手順で実行されます。

(1)「条件式」が調べられて、もし指定された条件が成り立っていない場合は「文」の実行は行わずに while 文全体の実行を終える。

(2)「文」を実行する。

(3) (1) へ戻る。

繰り返される「文」の部分を

```
while ( 条件式 ) {
    文1
    文2
     ⋮
    文n
}
```

のようにブロックにすれば、複数の文からなる一連の作業を繰り返し実行させることもできます。先の hexagon.c というプログラムは、まさにこの形をしていました。この場合

(1)「条件式」が調べられて、もし指定された条件が成り立っていない場合は「$文_1$」...「$文_n$」の実行は行わずに while 文全体の実行を終える。

(2)「文$_1$」...「文$_n$」を順に実行する。

(3) (1) へ戻る。

のような手順で実行されます。hexagon.c の場合、整数値の変数 i は 0 で初期化されていますから、最初は while 文の条件式である i < 6 が成り立ち、

```
tForward(100);
tTurn(60);
i = i + 1;
```

の 3 つの文が順に実行されます。そこで、カメは前方に 100 進み、反時計回りに 60° 向きを変えるとともに、変数 i の値は 1 に変ります。その後、プログラムの実行は while 文の条件式のチェックに戻り、依然として i < 6 が成り立っていますので、

```
tForward(100);
tTurn(60);
i = i + 1;
```

の部分の 2 回目の実行が行われます。2 回目の実行が終わると、変数 i の値は 2 となり、while 文の条件式 i < 6 のチェックに戻ります。同様に、3 回目、4 回目、... の実行が行われますが、6 回目の実行の後には変数 i の値が 6 になってしまいますから while 文の条件式のチェックに戻った時に i < 6 が成立しません。そこで while 文全体の実行が終了し、プログラム全体の実行も終了してしまいます。

整数値の変数 i は、常に、それまでに繰り返した回数を記憶していて、その回数が 6 に満たない (i < 6 の) 場合にさらに繰り返すようにしています。変数 i は、繰り返した回数を記憶するためのものですから、

```
int i;
```

と、整数値の変数として宣言されていることは重要です。もし、変数 i が実数値の変数として、つまり double i; と宣言されていると、i の記憶する値には誤差が含まれる可能性がありますから、6 回繰り返した後の i の値は 6 の近似値かも知れません[1]。よって、依然として i < 6 が成り立つかも知れないことに注意してください。もし、i の値が 6 よりわずかに小さいと、7 回目の繰り返しが始まってしまいます。回数を数えるための変数は、整数値の変数として宣言しましょう。

メモ

[1]実際は、それほど絶対の大きくない (数桁の) 整数の計算だけを行っている限りは、実数値の計算や記憶に誤差が生じることはまずありません。しかし、整数値で扱えるものは整数値として扱い、実数値として扱うことは避けましょう。

while 文の応用 (その 1)

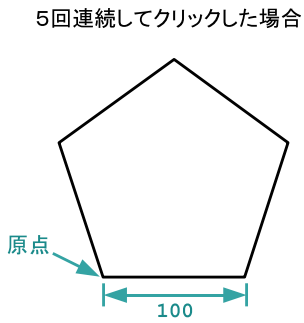
つぎのプログラムは、hexagon.c を発展させて、ウィンドウ内を n 回連続してマウスクリックした時に、正 n 角形を描くプログラムです。マウスクリックの位置は、描かれる図形に影響しません。

polygon.c

```
#include <turtle.h>

main ()
{
    int n, i;

    tGetClick();
    n = tClickCount();
    i = 0;
    while (i < n) {
        tForward(100);
        tTurn(360.0 / n);
        i = i + 1;
    }
}
```



このプログラムで使われている tClickCount() は、この科目のタートルグラフィックスの機能の一つで、クリックが連続した[2]回数を整数値で表すための数式です。polygon.c では、tClickCount() を使って取得したクリック数を整数値の変数 n に代入し、while 文を使って、繰り返しの回数 i が n に満たない間、次の 3 つの文を実行させています。

```
tForward(100);
tTurn(360.0 / n);
i = i + 1;
```

n が 360 の約数でなくてもカメが (できるだけ[3]) 正確に原点に戻って来られるように、カメが向きを変える角度は、360 / n ではなく、360.0 / n としていることに注意してください。

メモ

[2]クリックの間隔が 0.5 秒以内の場合に連続しているとみなされます

[3]実数値の計算には誤差が含まれますので、それでも正確に原点に戻れる訳ではありませんが、ディスプレイの画素の大きさに比べれば十分な精度が期待できます

while 文の応用 (その 2)

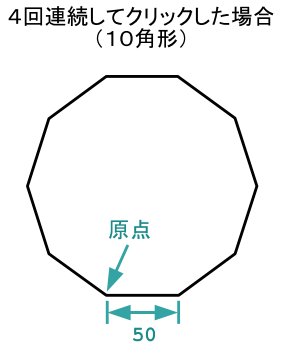
つぎの sum.c は、ウィンドウを n 回連続してクリックした時に、正 $(1+2+3+\ldots+n)$ 角形を描くプログラムです。たとえば、クリックの連続が 2 回なら 3 角形が、3 回なら 6 角形が描かれます。

sum.c

```
#include <turtle.h>

main ()
{
    int n, i, sum;

    tGetClick();
    n = tClickCount();
    sum = 0;
    i = 0;
    while (i < n) {
        i = i + 1;
        sum = sum + i;
    }
    i = 0;
    while (i < sum) {
        tForward(50);
        tTurn(360.0 / sum);
        i = i + 1;
    }
}
```



このプログラム内で、$1+2+3+\ldots+n$ の計算を行っているのは

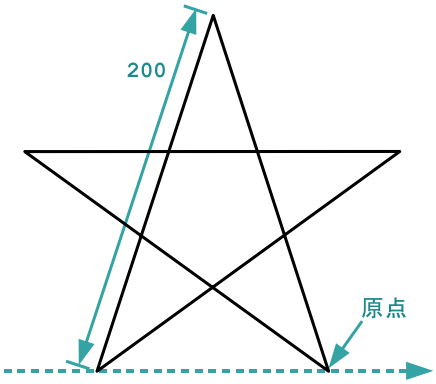
```
sum = 0;
i = 0;
while (i < n) {
    i = i + 1;
    sum = sum + i;
}
```

の部分です。まず整数値の変数 sum と i を 0 で初期化しておき、while 文を使って、i の値を 1 ずつ増やしながら、1 から順に、n まで sum に足し込むようにしています。sum = sum + i; という文が実行されると、変数 sum に記憶されている値は、i に記憶されている値だけ増えることに注意しましょう。

メモ

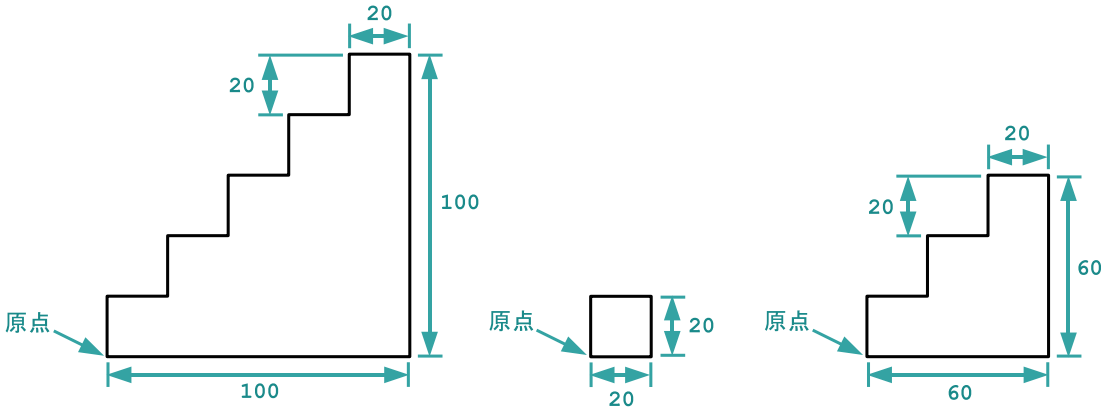
7.2　演習問題

1.　次のような図形をを描く C プログラム `star.c` を作り、コンパイル、実行して、正しく動作することを確認しなさい。ただし、プログラム中で `tForward` を使うのは 1 箇所だけにしてください。`tBackward` や `tMoveTo` は使ってはいけません。



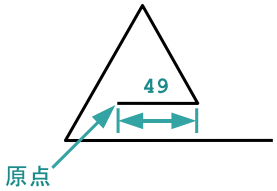
ヒント：　各頂点の内角は 36° です。

2.　ウィンドウ内を連続してマウスクリックすると、次のような階段状の図形を描く C プログラム `steps.c` を作り、コンパイル、実行して、正しく動作することを確認しなさい。連続したマウスクリックの回数が n 回の時に描かれる図形は、n 段の階段のような形になります。1 段の奥行きと段差は、すべて 20 です。

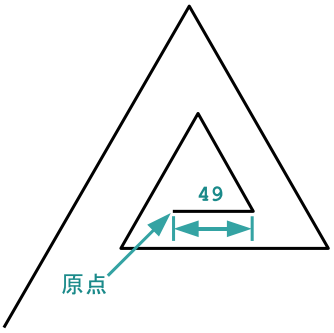


3.　初項 $a_1 = 49$ と漸化式 $a_{n+1} = 1.3\,a_n + 8 \quad (n = 1, 2, 3, \ldots)$ で定まる数列 $\{a_n\}$ を考えます。ウィンドウ内を n 回連続してマウスクリックすると、長さがそれぞれ a_1、a_2、a_3、…、a_n である n 本の辺で構成される下図のような図形を描く C プログラム `spiral2.c` を作り、コンパイル、実行して、正しく動作することを確認しなさい。各頂点で辺と辺のなす角はすべて 60° です。

4回連続してクリックした場合



6回連続してクリックした場合



ヒント： 第5回の演習問題で作成した `spiral.c` を参考にしましょう。

7.3 今回の実習内容

1. プリントをもう一度ゆっくりと読み直しましょう。`hexagon.c`、`polygon.c`、`sum.c` の3つの例題については、ソースプログラムを作成し、それをコンパイル、実行してみましょう。プログラムが完成したら「課題の提出と確認」の Web ページから提出してください。

   ソースプログラムは、必ず適当な「字下げ」を行って書いてください。プリントの例題プログラムの字下げを参考にして、`if` 文で条件付きで実行されたり、`while` 文で繰り返し実行される処理は字下げして書きます。今回の実習からは、字下げが適切になされていないプログラムは受理されません。何文字分の字下げを行うかについては、必ずしもプリントに合わせる必要はありませんが、プリントと同じように TAB キーを使って字下げを行うのがおすすめです。

2. 6ページの演習問題に取り組みましょう。それぞれのプログラムが完成したら、「課題の提出と確認」の Web ページからの提出を忘れないでください。

3. クイズに答えてください。前回と同様に「課題の提出と確認」の Web ページで「第7回 クイズ」を選択し、「送信」のボタンをクリックしてクイズに答えてください。

計算機基礎実習 I ・第7回・終わり

付録：前回の演習問題のプログラム例

diagonal.c

```
#include <turtle.h>

main()
{
    int x, y;

    tGetClick();
    x = tClickX();
    y = tClickY();
    if (y*y >= x*x)
        tMoveTo(x, y);
}
```

square.c（その 1）

```
#include <turtle.h>

main()
{
    int x, y;

    tGetClick();
    x = tClickX();
    y = tClickY();

    if (x < 0)
        x = -x;
    if (y < 0)
        y = -y;
    if (x < y)
        x = y;

    tPenUp();
    tMoveTo(x, x);
    tPenDown();
    tMoveTo(-x, x);
    tMoveTo(-x, -x);
    tMoveTo(x, -x);
    tMoveTo(x, x);
}
```

square.c（その 2）

```
#include <turtle.h>

main()
{
    int x, y, a;

    tGetClick();
    x = tClickX();
    y = tClickY();

    if (x < 0)
        x = -x;
    if (y < 0)
        y = -y;
```

```
    if (x < y)
        a = y;
    else
        a = x;

    tPenUp();
    tMoveTo(a, a);
    tPenDown();
    tMoveTo(-a, a);
    tMoveTo(-a, -a);
    tMoveTo(a, -a);
    tMoveTo(a, a);
}
```