計算機基礎実習 I ・第 8 回

2017 年 6 月 9 日

今回の内容

8.1 繰り返しの処理 (続き)

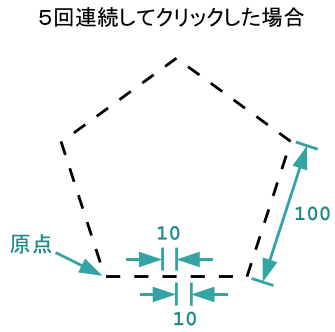
繰り返しの中で場合分けの処理を行うプログラム while 文によって繰り返し実行する文は、どのような形をしていても構いませんから、繰り返す仕事の一部として場合分けの処理を行うこともできます。次のプログラム dpolygon.c は、ウィンドウ内を n 回連続してマウスクリックした時に、1 辺が 100 の正 n 角形を破線で描くプログラムです。

dpolygon.c

```
#include <turtle.h>

main ()
{
    int n, i;

    tGetClick();
    n = tClickCount();
    i = 0;
    while (i < 10 * n) {
        if (i % 2 == 0)
            tPenDown();
        else
            tPenUp();
        tForward(10);
        i = i + 1;
        if (i % 10 == 0)
            tTurn(360.0 / n);
    }
}
```



このプログラムは、まず、連続したクリックの回数を取得して変数 n に格納しています。その後、変数 i を 0 で初期化し、i < 10 * n が成り立っている間、以下の処理を繰り返しています。

1. i が偶数のときはペンを下げ、そうでなければペンを上げる。
2. 前へ 10 進む。
3. 変数 i の値を 1 増やす。
4. i の値が 10 の倍数になったら $\frac{360^\circ}{n}$ だけ左に向きを変える。

このプログラムでは、正 n 角形の各辺を 10 等分し、10 等分されてできた線分の奇数番目[1]ではペンを下ろしてカメを進め、偶数番目はペンを上げて進めることで、長さ 100 の各辺が破線となるよ

[1]変数 i が 0 の時が 1 番目となりますから、i が偶数の時に奇数番目となります

うに描いています。正 n 角形の頂点にやってきた時には、i は 10 の倍数となりますから、その時カメの向きを変えています。

メモ

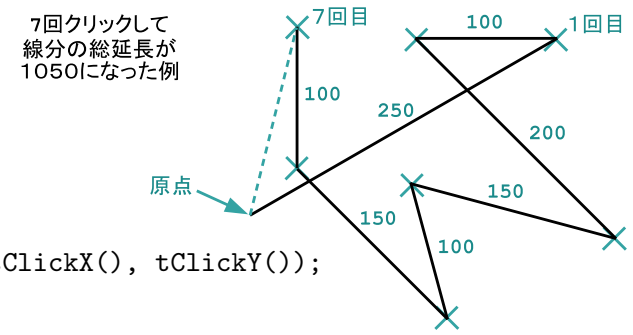
入力を繰り返すプログラム　繰り返される仕事の一部として、マウスクリックの入力処理を行いたい場合もあります。次のプログラム follow1000.c は、ウィンドウ内をマウスでクリックする度に、カメがクリックされた位置に向かって線分を描いていくプログラムです。ただし、描いた線の長さの合計が 1000 に達すると、この仕事を終え、ペンを上げて原点に戻ります。

follow1000.c

```
#include <turtle.h>

main ()
{
    double sum;

    sum = 0;
    while (sum < 1000.0) {
        tGetClick();
        sum = sum + tMoveTo(tClickX(), tClickY());
    }
    tPenUp();
    tMoveTo(0, 0);
}
```



ここで使われている tMoveTo(...) は、カメを指定した座標に移動させるための指示ですが、

```
tMoveTo(100, 200);
```

のように単独の文として使うことができる他に、この例のように数式の中で使うこともできます。数式の中に tMoveTo(x, y) と書くと、その部分が計算される時、カメが 点 (x, y) に移動するとともに、そのときカメが移動した距離が、この tMoveTo(x, y) という式の値となります。この例では、変数 sum への代入すべき値を表す数式

```
sum + tMoveTo(tClickX(), tClickY())
```

の一部として使われていますが、この式全体は足し算の式なので、+ の両側の式の値が決まらないと計算できません。+ の左式は変数 sum なので、この時 sum に記憶されている値がその値となります。一方、+ の右式は tMoveTo(tClickX(), tClickY()) です。コンピュータは、この式の値を計算するために tMoveTo(tClickX(), tClickY()) を実行します。これにより、カメは tClickX() と tClickY() の値で定まる座標の点に移動しますが、その際の移動距離が tMoveTo(tClickX(), tClickY()) という式の値となり、左式の sum の値と足し合わされて、その結果が変数 sum に代入されます。

メモ

8.2 関数

C 言語では、コンピュータに何らかの仕事や計算を行わせるための一連の指示をひとまとまりにしたものを**関数**と呼びます。これまで見てきたプログラムで、タートルグラフィックスを行うために使ってきた tForward、tTurn、tMoveTo、tPenUp、tPenDown、tGetClick、tClickX、tClickY などはすべて、この関数と呼ばれるものの名前[2]です。

関数呼び出し式と引数　関数に仕事をさせたい場合は、その関数の名前に続けて (を書き、その仕事の詳細を指定するための式を「,」で区切っていくつか書いて、最後に) を書きます。たとえば、tForward(100) や tPenUp()、tMoveTo(x, y) などの書き方がそうです。この形で関数に仕事や計算をさせる書き方を**関数呼び出し式**といい、( ) の間に書いた式を (この関数呼び出し式の) **引数** (ひきすう) と呼びます。関数を呼び出す際に、どのような種類 (整数値なのか実数値なのか) の引数をいくつ書いて呼び出したらよいのかは、関数ごとに決まっています。もちろん、関数を呼び出したとき、その関数がどのような仕事や計算をするのかについても、関数ごとに違います。

関数呼び出し式に続いて ; を書けば文となります。これまでに何度も使ってきた

```
tForward(100);
tTurn(90);
tMoveTo(100, 200);
```

などは、この形の文です。

メモ

関数の戻り値　一部の関数は、仕事や計算を終えた後に、何らかの値を返してくれるようになっており、関数呼び出し式を数式の中に書けば、その返された値を計算に使うことができます。関数が仕事を終えた後に返す値のことを、関数の**戻り値**あるいは**返り値**と呼びます。数式中に tMoveTo(x, y) と書くと、2 つの引数で指定した座標にカメが移動するとともに、その式としての値が移動距離となるのはこの働きのためです。整数値が戻り値となる関数もあれば、実数値が戻り値となる関数もあります。マウスクリックの座標を返す tClickX や tClickY は前者ですし、tMoveTo は後者

[2]例示した関数は、すべて、この科目のタートルグラフィックスの機能として用意されているもので、その名前が t で始まっていますが、一般の関数の名前は、変数の名前と同様に、英文字で始まり、その後に、英文字か数字、下線記号 (_) が続くものとなります

です。戻り値のある関数の関数呼び出し式は、その式が計算されるときに、その関数が呼び出されて、呼び出された関数の仕事や計算が行われ、それが終わった後に戻される値が、その式の値として続く計算に使われます。

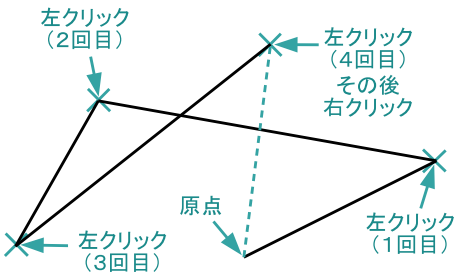
メモ

クリックされたボタンの識別　つぎのプログラム follow.c は、ウィンドウ内をマウスの左ボタンでクリックする度に、カメがクリックされた位置に向かって線分を描いていき、中ボタンや右ボタンが押されると、この仕事をやめ (ペンを上げて) 原点に戻るプログラムです。

follow.c

```
#include <turtle.h>

main ()
{
    while (tGetClick() == 1) {
        tMoveTo(tClickX(), tClickY());
    }
    tPenUp();
    tMoveTo(0, 0);
}
```



この科目のタートルグラフィックスの機能の一つである tGetClick という関数は、ウィンドウ内でのマウスクリックを待って、クリックされた位置などの情報を保存しておくという仕事をしますが、この仕事を終えると、マウスのどのボタンがクリックされたのかを整数値の戻り値として返します。この戻り値は、左ボタンなら 1、中ボタンなら 2、右ボタンなら 3 となります。

follow.c では、while 文の条件式に tGetClick() == 1 と書いてあります。次の繰り返しを行うかどうかを決めようとする度に、この条件式がチェックされ、== の両辺の値が計算されます。右辺には 1 と書かれているので、その値は整数値の 1 となりますが、左辺には tGetClick() という関数呼び出し式が書かれているので、関数 tGetClick が呼び出されます。関数 tGetClick が呼び出されると、コンピュータはマウスがクリックされるのを待ち、クリックされたら、その位置などの情報を保存しておくとともに、クリックされたボタンの番号を戻り値として返します。この値が == の左辺の値となり、先ほどの 1 と等しいかどうかが調べられて、この条件式が成り立つかどうかが決定されます。

メモ

8.3 条件式の組み合わせ

これまでは、if 文や while 文の条件式として 2 つの値の比較のみを扱いましたが、複数の条件式を組み合わせてもっと複雑な条件を指定することもできます。たとえば、変数 a、b、c の値について、a < b < c が成り立っている時に限って実行したい文があるとします。残念ながら C というプログラミング言語では、

```
if (a < b < c)
    実行したい文
```

のように書いたのでは、このような働きをさせることができません。a < b < c が成り立っている時に限って実行したい文があるときは、a < b と b < c という 2 つの条件式を組み合わせて、

```
if (a < b && b < c)
    実行したい文
```

のように書きます。「&&」は、日本語の「かつ」を表しており、両側の 2 つの条件が「共に成り立つ」ということを意味します。このプログラムは次のように書いたのと同じ意味になります。

```
if (a < b)
     if (b < c)
        実行したい文
```

この他にも「または」を表す「||」や、「... でない」を表す「!」があります。

条件式	調べられる条件
条件式$_1$ && 条件式$_2$	条件式$_1$ と 条件式$_2$ が両方成り立っている
条件式$_1$ || 条件式$_2$	条件式$_1$ と 条件式$_2$ の少なくとも一方が成り立っている
!（条件式$_1$）	条件式$_1$ が成り立っていない

条件を組み合わせる && や || は、それぞれ、掛け算の * と足し算の + の場合と同じように使うことができます。たとえば、

```
a < b && b < c && c == 0
```

のように、3 つ以上の条件を組み合わせることも可能です。複数の条件式が && で結ばれたときには、左から順に条件式がチェックされていき、途中で成り立っていない条件式を見つけた場合には、それより後の条件式はチェックされません。|| で複数の条件式が結ばれた場合も同様で、途中で成り立っている条件式を見つけると、それより後の条件式はチェックされません。

&& や || が混在した条件式の結び付き方を指定するためには

```
a < b || (b < c && c == 0)
```

あるいは

```
(a < b || b < c) && c == 0
```

のように適当に ( ) を用います。( ) を書かない場合は、ちょうど掛け算の * が足し算の + よりも先に計算されるのと同じように、&& の方が || よりも先に結び付きます。つまり

```
a < b || b < c && c == 0
```

と書くと

```
a < b || (b < c && c == 0)
```

の意味になります。

また、! は、比較のための ==、!=、<、<=、>、>= よりも強く結び付きますので、「a と b が等しくない」という条件を指定するためには !(a == b) のように、まず a == b という条件を () で囲む必要があります。同じように !(a > 0 && b < 0) の () も必要です。もちろん、それぞれ a != b や a <= 0 || b >= 0 のように ! を使わない等価な条件を書いておけばこのような配慮は不要になります。

メモ

条件式の組合わせを使ったプログラム例　次のプログラム follow5.c は follow.c とほぼ同じ動作をしますが、最大5本の線分しか描きません。5本の線分を描いたら、中ボタンや右ボタンが押されなくても、カメは仕事をやめて原点に戻ってしまうようになっています。

follow5.c

```
#include <turtle.h>

main ()
{
    int i;

    i = 0;
    while (i < 5 && tGetClick() == 1) {
        tMoveTo(tClickX(), tClickY());
        i = i + 1;
    }
    tPenUp();
    tMoveTo(0, 0);
}
```

while 文の条件式は i < 5 && tGetClick() == 1 のように、2つの条件式を && (かつ) で結んだ形になっていますので、

1. まだ5本の線分を描いていない
2. クリックされたのは左ボタンである

という2つの条件がともに成り立っている間だけ、続くブロックの中に書いてある

```
tMoveTo(tClickX(), tClickY());
i = i + 1;
```

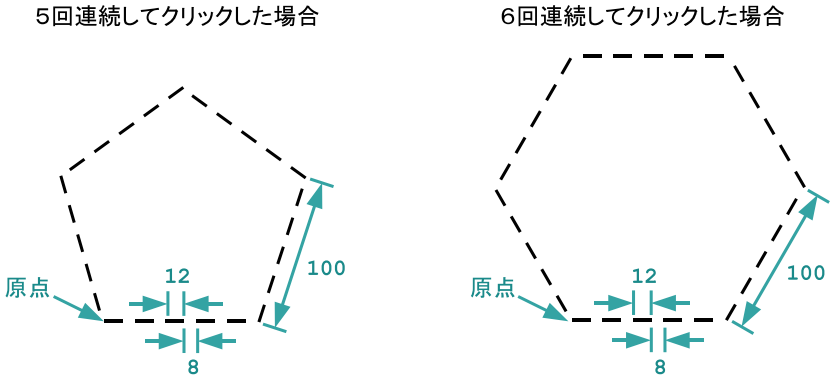
という2つの文が繰り返し実行されます。この2つの条件式の順番は重要です。&& で結ばれた条件式は、左から順にチェックされていきますから、もし、tGetClick() == 1 && i < 5 の順に書い

てしまうと、すでに i が5になっている (5本の線分を書き終えている) 場合でも、関数 tGetClick が呼び出され、コンピュータはマウスがクリックされるのを待ってしまいます。

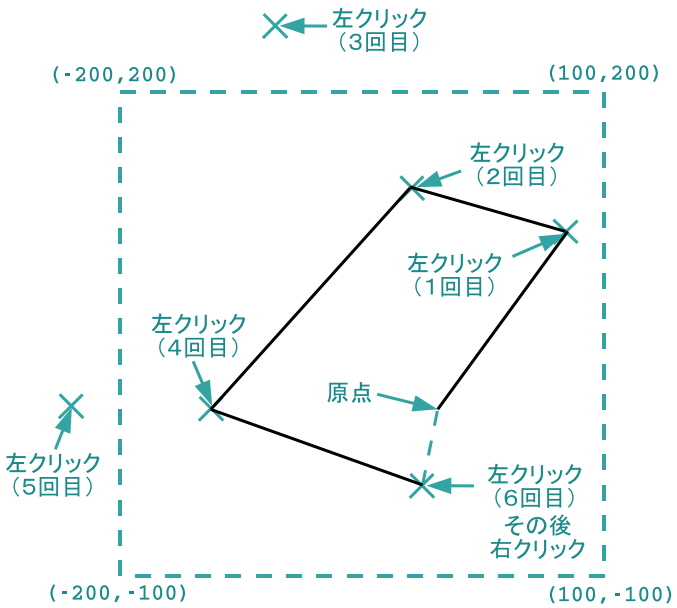
メモ

8.4 演習問題

1. 1ページの dpolygon.c を改造して、長さ12の線分を間隔8で繰り返すことで破線を描くようなプログラム dpolygon2.c を作りなさい。ただし、n 角形を破線で描くためにカメが移動する回数は、全体で $10n$ 回まで (dpolygon.c と同じ) にしてください。

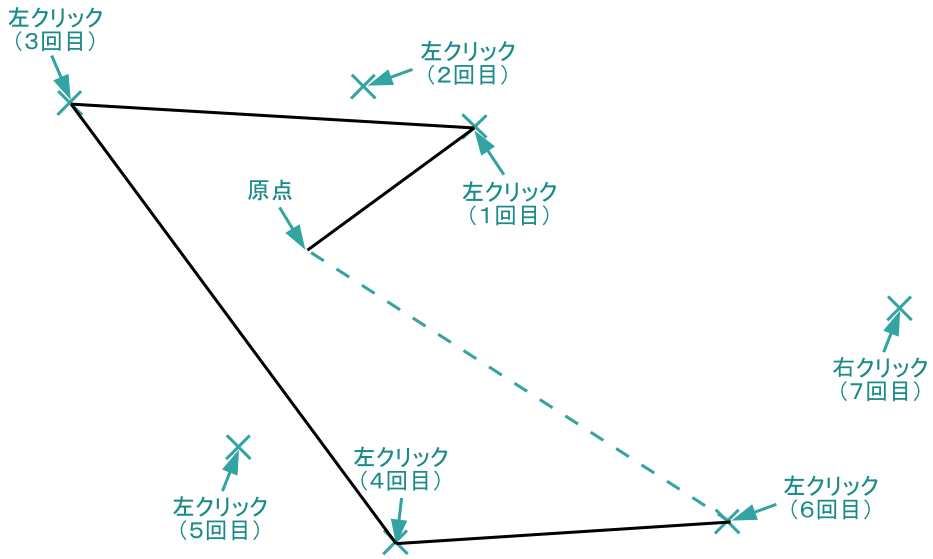


2. 4ページの follow.c を改造して、$(100,\ 200)$、$(-200,\ 200)$、$(-200,\ -100)$、$(100,\ -100)$ を頂点とする正方形の外部が左ボタンでクリックされても、そのクリックを無視するようなプログラム margin.c を作りなさい。



この正方形の辺上や内部がクリックされている限り、このプログラムは follow.c と同じ動作をします。中ボタンや右ボタンがクリックされると、クリックされた位置がどこであってもカメは仕事をやめて (ペンを上げ) 原点に戻るようにしてください。プログラムをテストする際には、コントロールキーを押してマウスポインタの座標を確認しましょう。

3. 4 ページの follow.c を改造して、カメの現在位置と比べて、原点により近い点が左クリックされた場合は、そのクリックを無視するようなプログラム farther.c を作りなさい。原点から等距離か、より遠い点が左クリックされている限り、このプログラムは follow.c と同じ動作をします。中ボタンや右ボタンがクリックされると、クリックされた位置がどこであってもカメは仕事をやめて (ペンを上げ) 原点に戻るようにしてください。



ヒント 整数値を記憶するための変数を用意しておき、カメの現在位置の座標 (あるいは原点からの距離の 2 乗) を記憶しておくようにしましょう。左ボタンでクリックされる度に、クリックされた点とどちらが原点から遠いかを判定し、等距離か、クリックされた点がより遠い場合にのみクリックされた点に移動するようにします。移動した場合は、カメの現在位置 (あるいは原点からの距離の 2 乗) を記憶している変数の値を更新します。

8.5 今回の実習内容

1. プリントをもう一度読み返しましょう。dpolygon.c と follow5.c の 2 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。
2. 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 8 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 8 回・終わり

付録：前回の演習問題のプログラム例

star.c

```
#include <turtle.h>

int main()
{
    int i;

    tTurn(108);
    i = 0;
    while (i < 5) {
        tForward(200);
        tTurn(144);
        i = i + 1;
    }
}
```

steps.c

```
#include<turtle.h>

main()
{
        int i, n;

        tGetClick();
        n = tClickCount();
        i = 0;
        while (i < n) {
                tTurn(90);
                tForward(20);
                tTurn(-90);
                tForward(20);
                i = i + 1;
        }
        tTurn(-90);
        tForward(n * 20);
        tTurn(-90);
        tForward(n * 20);
}
```

spiral2.c

```
#include <turtle.h>

main()
{
    int n, i;
    double a;

    tGetClick();
    n = tClickCount();
    a = 49.0;
    i = 0;
    while (i < n) {
        tForward(a);
        tTurn(120);
        a = a * 1.3 + 8.0;
        i = i + 1;
    }
}
```