計算機基礎実習 I ・第 5 回

2017 年 5 月 19 日

今回の内容

5.1 変数 . . . . . 5−1
5.2 タートルグラフィックス機能の紹介 . . . . . 5−6
5.3 演習問題 . . . . . 5−8
5.4 今回の実習内容 . . . . . 5−9

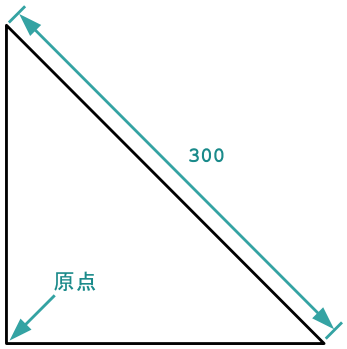
5.1 変数

一度計算した数式の値を、後でまた利用したい場合がよくあります。たとえば、斜辺が 300 の直角二等辺三角形を描きたい場合、他の 2 辺の長さはともに $\frac{300}{\sqrt{2}}$ となりますが、この計算を行う数式をプログラム中に 2 度書くのは面倒です。

C というプログラミング言語では、計算の途中結果などの数値を一時的に記憶しておくことができます。この数値を記憶する働きをもつものを **変数** と呼びます。変数は数値を書き留めておくメモ欄のようなものです。1 つの変数には 1 つの数値を記憶することができ、記憶された値はその後いつでも参照する (使う) ことができます。次のプログラム `var1.c` は、変数を利用して斜辺が 300 の直角二等辺三角形を描くものです。

var1.c

```
#include <turtle.h>

main()
{
    double a;

    a = 300 / 1.41421;
    tForward(a);
    tTurn(135);
    tForward(300);
    tTurn(135);
    tForward(a);
}
```



メモ

変数の宣言 `main()` に続く `{` のつぎに書かれている

```
double a;
```

は「これから実数値を記憶するために `a` という名前の変数を使いますよ」という宣言です。`double` は double precision (倍精度) という英語に由来していて、基本的な精度の倍の精度で実数値の近似値を計算したり記憶したりするという意味です。この宣言によって、実数値を記憶することのできる `a` という変数が 1 つ作られます。この `a` は `{` に対応する `}` までの間で、実数値を一時的に覚え

ておくための記憶場所として使用することができます。この科目で勉強している C 言語では、変数の宣言は { の直後に書かなければなりません。また、宣言されていない変数をプログラム中で突然使うことはできません。慣習的に、変数の宣言を (すべて) 書き終えたら、1 行空けてから具体的な作業を伴うプログラムを書き始めます。

メモ

変数への代入　1 行空けたあとの、

```
a = 300 / 1.41421;
```

という部分では、300 / 1.41421 という式を計算した結果の 212.132... という値を変数 a に記憶させています。このような操作を「変数への値の**代入**」と呼びます。変数に値を代入するときには、その変数名に続けて = を書き、その後に代入したい値を表す数式と「;」(セミコロン) を書きます。その数式の計算結果の値が = の前に書かれた変数に代入されます。変数に代入された値は後で別の計算に使うことができます。

実数値の変数に実数値が代入できるのはもちろんですが、

```
double a;

a = 100;
```

のように、実数値の変数への代入を行う = の右辺に、整数値を表す式 (この例では定数) を書くこともできます。この場合、整数値は自動的に実数値に変換されて、左辺の変数に代入されます。

メモ

変数の参照　これにつづく

```
tForward(a);
```

では、変数 a の値だけ、カメを前に進めています。変数は、このように 7.8 や 300.0 などの定数値と同じように使うことができます。数式の中で変数が使われた場合、その値は、その時点でその変数に格納されている値となります。

実数値の変数の値が参照されるときは、必ず実数値として扱われます。たとえば、

```
double a;

a = 100;
tForward(a / 3);
```

のような場合、a / 3 という数式の値は、整数値の 33 ではなく、実数値の 33.333333... となります。

メモ

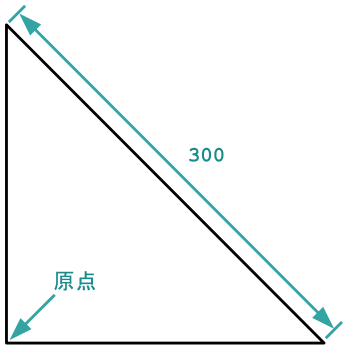
変数の初期化　変数を宣言してから、まだ 1 度もその変数への値の代入を行っていない状態での変数の値は予測がつきません。変数に初めて値を代入することを「変数の**初期化**」と呼びます。初期化されていない変数の値を参照すると (プログラムが実行される度に異なる可能性のある) 予測できない値を使うことになってしまい、プログラムの動作は予期できないものとなりますので注意が必要です。

メモ

何度でも代入できる　変数に記憶されている値は、代入によって何度でも置き換えることができます。代入を行うと元々その変数に記憶されていた値は失われてしまい、その時代入された新しい値が記憶されます。たとえばつぎの `var2.c` というプログラムは、先の `var1.c` と全く図形を描いてくれます。

var2.c

```
#include <turtle.h>

main()
{
    double a;

    a = 300 / 1.41421;
    tForward(a);
    tTurn(135);
    a = 300;
    tForward(a);
    tTurn(135);
    a = a / 1.41421;
    tForward(a);
}
```



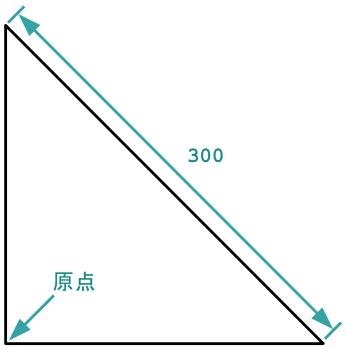
まず「`a = 300 / 1.41421;`」が実行されると、変数 `a` には 212.132... が代入され、次の行に書かれた「`tForward(a);`」によって、この値だけカメは前に進みます。「`a = 300;`」が実行されると、変数 `a` の値は (実数値の) 300 に変わります。この値は直角三角形の斜辺を描くために使われます。下から 3 行目の「`a = a / 1.41421;`」が実行されるときには、まず、`=` の右辺が計算されて 212.132... となり、この値が変数 `a` の新しい値となります。この時、`=` の右辺 `a/1.41421` の計算で参照される `a` の値は「`a = 300;`」で代入された (実数値の) 300 であることに注意してください。

メモ

複数の変数を使う　C プログラムでは、変数をいくつも使うことができます。それぞれに別の名前を付けて使います。次のプログラムもやはり var1.c と同じ図形を描くものです。

var3.c

```
#include <turtle.h>

main()
{
    double a, b, c;

    a = 300;
    b = a / 1.41421;
    c = 135;
    tForward(b);
    tTurn(c);
    tForward(a);
    tTurn(c);
    tForward(b);
}
```



このプログラムでは「double a, b, c;」のように、3 つの変数を一度に宣言していますが、

```
double a;
double b;
double c;
```

のようにいくつかに分けて宣言することもできます。こうした場合でも変数の宣言は { の直後にまとめられていなければなりません。

メモ

変数の名前　ここまでの例では、a や b など 1 文字の英文字からなる名前の変数を使ってきましたが、もっと長い名前の変数を使うこともできます。たとえば var3.c の変数名を次のプログラムのように変えてもプログラムの動作は変りません。

var4.c

```
#include <turtle.h>

main()
{
    double hypotenuse, leg, angle;

    hypotenuse = 300;
    leg = hypotenuse / 1.41421;
    angle = 135;
    tForward(leg);
    tTurn(angle);
    tForward(hypotenuse);
    tTurn(angle);
    tForward(leg);
}
```

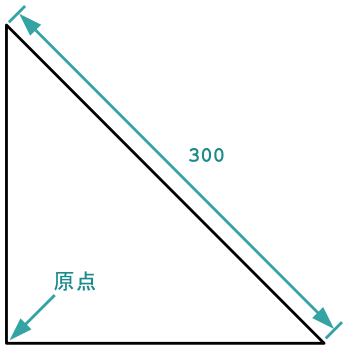
C プログラムの変数の名前としては、英字で始まり、そのあとに英字か数字あるいは _ (下線記号) がいくつか続くものを使います。ただし、いくつかの名前は C プログラムの**予約語** (特別な意味を持っている単語) となっていて、変数名としては使うことができませんので注意してください。たとえば、実数値の変数を宣言するために使った `double` という単語も C の予約語の 1 つです。`double` という名前の変数を使うことはできません。

メモ

整数値の変数　整数値を記憶する変数を使いたい場合は、`double` の代わりに `int` と書いて変数を宣言します。この `int` も C プログラムの予約語の一つです。`int` は integer (整数) という英単語に由来しています。整数値の変数への代入や、整数値の変数の値の参照は、実数値の変数と同じようにできます。次のプログラムも `var1.c` と同じ三角形を描きます。

var5.c

```
#include <turtle.h>

main()
{
    double leg;
    int hypotenuse, angle;

    hypotenuse = 300;
    leg = hypotenuse / 1.41421;
    angle = 135;
    tForward(leg);
    tTurn(angle);
    tForward(hypotenuse);
    tTurn(angle);
    tForward(leg);
}
```



`hypotenuse` や `angle` は整数値の変数にすることができますが、

```
leg = hypotenuse / 1.41421;
```

の右辺の計算結果は実数値となるため、`leg` を整数値の変数にできないことに注意してください。

メモ

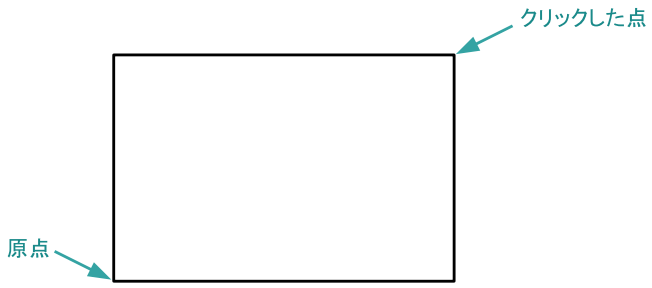
5.2　タートルグラフィックス機能の紹介

マウスクリック位置の取得　この科目で利用しているタートルグラフィックスのシステムには、カメが表示されるウィンドウ上でマウスがクリックされた場合に、その位置 (座標) をプログラム中で取得する機能が含まれています。

次の C プログラム `click.c` をコンパイルして、実行すると、はじめは何の作業も行いませんが、ウィンドウをマウスでクリックすると、クリックした点と原点を対角線とする矩形 (長方形) を描いてくれます。

click.c

```
#include <turtle.h>

main()
{
    int x, y;

    tGetClick();
    x = tClickX();
    y = tClickY();
    tForward(x);
    tTurn(90);
    tForward(y);
    tTurn(90);
    tForward(x);
    tTurn(90);
    tForward(y);
}
```



このプログラムに書かれている

```
tGetClick();
```

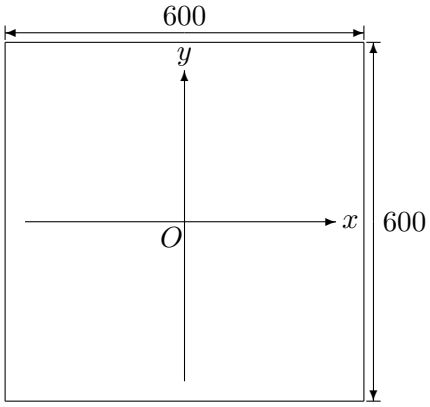
という指示が実行されると、プログラムの実行はそこで一旦停止し、カメが表示されているウィンドウがマウスでクリックされるのを待ちます。この状態で、ユーザがマウスでクリックすると、クリックした位置 (座標) が記録され、プログラムの実行は先に進むようになります。続く

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

で使われている `tClickX()` や `tClickY()` は、この x 座標と y 座標を**整数値として**それぞれ取り出すための書き方で、数式の一部としていつでも使うことができます。このプログラムでは、取り出した座標を整数値の変数 `x` と `y` にそれぞれ代入しています。この `tClickX()` や `tClickY()` の末尾の `()` は省略することはできませんので注意が必要です。

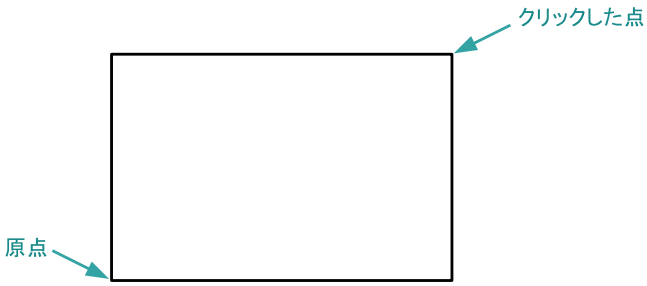
メモ

タートルグラフィックスの座標系　`tClickX()` や `tClickY()` の値は、下図のような座標系における x 座標と y 座標となります。



tClickX() や tClickY() の値は、次に tGetClick() が実行されるまでは変わりませんので、先のプログラム click.c は次のように書き換えても動作は変わりません。

```
#include <turtle.h>

main()
{
    tGetClick();
    tForward(tClickX());
    tTurn(90);
    tForward(tClickY());
    tTurn(90);
    tForward(tClickX());
    tTurn(90);
    tForward(tClickY());
}
```



メモ

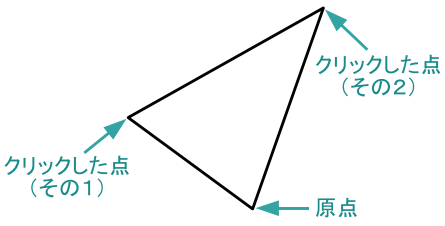
指定した位置への移動　この科目で利用しているタートルグラフィックスのシステムでは、カメの向きを変えたり、前進、後退させたりするだけでなく、

tMoveTo(*x*座標, *y*座標);

のような指示を書くことで、指定した座標へカメを移動させることもできます。次のプログラムは、クリックした 2 点と原点を頂点とする三角形を描くものです。

moveto.c

```
#include <turtle.h>

main()
{
    tGetClick();
    tMoveTo(tClickX(), tClickY());
    tGetClick();
    tMoveTo(tClickX(), tClickY());
    tMoveTo(0, 0);
}
```



メモ

ペンの上げ下げ　カメを移動させると、その軌跡が線となって描かれますが、この軌跡を描かせないようにすることもできます。

```
tPenUp();
```

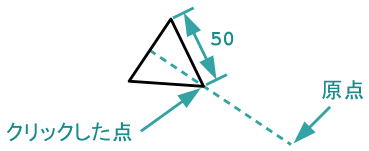
という指示を与えると、その後はカメが移動しても線が描かれないようになります。また、

```
tPenDown();
```

を実行すると、カメは再び線を描くようになります。次のプログラムは、マウスでクリックした点を頂点として、原点の反対側に一辺が 50 の正三角形を描きます。

pen.c

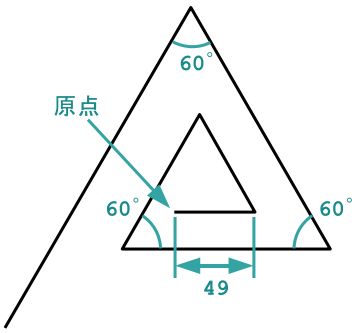
```
#include <turtle.h>

main()
{
    tGetClick();
    tPenUp();
    tMoveTo(tClickX(), tClickY());
    tPenDown();
    tTurn(-30);
    tForward(50);
    tTurn(120);
    tForward(50);
    tTurn(120);
    tForward(50);
}
```



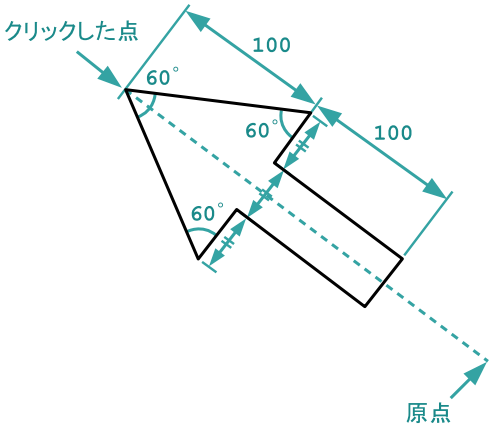
メモ

5.3　演習問題

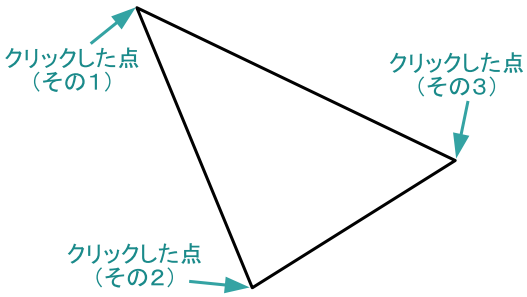
1.　初項 $a_1 = 49$ と漸化式 $a_{n+1} = 1.3\,a_n + 8 \quad (n = 1,\ 2,\ 3,\ \ldots)$ で定まる数列 $\{a_n\}$ を考えます。長さがそれぞれ a_1、a_2、a_3、a_4、a_5、a_6 の 6 つの辺で構成される下図のような図形を描く C プログラム `spiral.c` を作り、コンパイル、実行して、正しく動作することを確認しなさい。各頂点で辺と辺のなす角はすべて 60° です。ただし、定数としては 1.3、8、49、120 の 4 種類だけを使ってプログラムを書いてください。これら以外の定数を使ってはいけません。また、変数を 1 つだけ使用し、四則演算の演算子を使うのは、プログラム全体で 10 個所までにしてください。



2. ウィンドウをクリックすると、次のような図形を描く C プログラム `arrow.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。この図形は、クリックした点を頂点とする正三角形と長方形をつなげたような形をしています。長方形の短辺の長さは、正三角形の 1 辺の長さの $\frac{1}{3}$ です。クリックする位置によっては、原点がこの図形の内側となる場合もあります。$\sqrt{3}$ の近似値としては 1.73205 を使用してください。原点がクリックされた場合の矢印の向きは気にする必要はありません (どの向きに描かれても ok です)。



3. ウィンドウを 3 回クリックすることで、クリックした 3 点を頂点とする三角形を描く C プログラム `triangle.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。クリックする度に辺を描くようなプログラムでも、3 回クリックした後で一気に三角形を描くプログラムでも ok です。



5.4 今回の実習内容

今回からは、自分が作成したソースプログラムや、コンパイルしてできたオブジェクトプログラムは、「ホームドライブ (`Q:`)」や「デスクトップ」に置くのではなく、適当なフォルダを自分で作成し

て、その中に置くようにしてください。これまでに作ったプログラムも、そのフォルダに移動しておいてください。

1. このプリントをもう一度読み返しましょう。`var5.c`、`click.c`、`moveto.c`、`pen.c` の 4 つの例題プログラムについては、それぞれソースプログラムを作成し、コンパイル、実行してみて正しく動くかどうか確認しましょう。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 ページからソースプログラムを提出してください。
2. このプリント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、やはり「課題の提出と確認」の Web ページからソースプログラムを提出してください。
3. クイズに答えてください。「課題の提出と確認」の Web ページで「第 5 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。このクイズは何度でもやってみることができます。最高得点を評価の対象としますので、満点を狙ってください。

計算機基礎実習 I ・第 5 回・終わり

付録：前回の演習問題のプログラム例

paren.c

```
#include <turtle.h>

main()
{
    tForward((11 - (22 * 33 - 44 * 55) - 66) * 77 / (88 * 9.9));
    tTurn(120);
    tForward((11 - (22 * 33 - 44 * 55) - 66) * 77 / (88 * 9.9));
    tTurn(120);
    tForward((11 - (22 * 33 - 44 * 55) - 66) * 77 / (88 * 9.9));
}
```

halfarrow.c

```
#include <turtle.h>

main()
{
    tForward(235);
    tTurn(135);
    tForward((235 - 88) * 1.41421);
    tTurn(135);
    tForward(235 - 2 * 88);
    tTurn(-90);
    tForward(88);
    tTurn(90);
    tForward(88);
}
```

remainder.c

```
#include <turtle.h>

main()
{
    tForward(235);
    tTurn(90);
    tForward(235);
    tTurn(90);
    tForward(235 % 66);
    tTurn(90);
    tForward(235 % 66);
    tTurn(270);
    tForward(66);
    tTurn(90);
    tForward(66);
    tTurn(270);
    tForward(66);
    tTurn(90);
    tForward(66);
    tTurn(270);
    tForward(66);
    tTurn(90);
    tForward(66);
}
```