プログラミング及び実習Ｉ・第１回

2020年6月11日

今回の内容

1.1 C プログラムの基本

もっとも単純な C のプログラムは次のようなものです[1]。

empty.c

```
main()
{
}
```

このプログラムは特に何の仕事もしません。このプログラムをコンパイルして実行しても何も起こりません。このプログラムは何の役にも立ちませんが、C のプログラムには必ず `main()` と、それに続く `{` と `}` が書かれていなければならないことに注意してください。

メモ

1.2 仕事のさせかた

何らかの作業をコンピュータにさせたいときには、この `{` と `}` の間に実行させたい作業を実行させたい順に書きます。「情報基礎」の最終回で紹介した `myfirst.c` の例では、

```
tForward(30);
tBackward(15);
tTurn(90);
tForward(150);
tTurn(120);
tForward(28);
```

のように書かれていました。このソースプログラムをコンパイルすることで得られた `myfirst.exe` というオブジェクトプログラムは、これらの指示を上から順に実行し、(画面に表示されたウィンドウ上のカメを操作して) 数字の 1 の形を描いてくれたのです。

ソースプログラム `myfirst.c` に書かれている `tForward(30);` などの指示は、それぞれ次のような意味を持っています。

[1] 最新の C 言語の標準規格では `main` の前に `int` を書く決まりになっていますが、ほとんどのコンパイラは、この `int` を省略しても、そこに `int` があるものとして扱ってくれます。この科目では、`main` の前の `int` を省略した書き方でプログラムを紹介していくことにします。

```
tForward(30);   — カメを前方に 30 進める
tBackward(15);  — カメを後方に 15 戻す
tTurn(90);      — カメを反時計回りに 90° 回転する
tForward(150);  — カメを前方に 150 進める
tTurn(120);     — カメを反時計回りに 120° 回転する
tForward(28);   — カメを前方に 28 進める
```

ディスプレイのスクリーンは、**画素**[2]と呼ばれる無数の点が格子状に並ぶことで構成されていて、この画素がそれぞれいろいろな色になることで、スクリーン全体に画像が浮かび上がる仕組みになっています。プログラムを起動した時に現れるウィンドウはこの画素を単位として、およそ 600×600 の大きさになっています。カメの移動距離も、この画素を単位として計られます。最初、カメはこのウィンドウの中央 (原点) に位置し、右を向いています。カメが移動すると、その軌跡が黒い線としてウィンドウ上に描画されます。

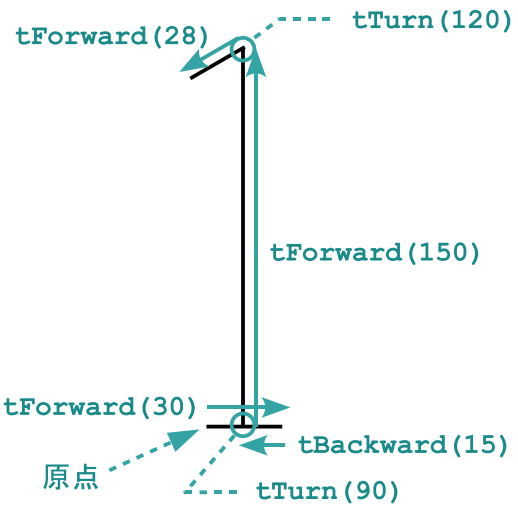
このように、画面上のカメの向きを変えたり、前進させたりして、図形を描く方法を、一般に**タートルグラフィックス**と呼びます。この「プログラミング及び実習 I」では、主にタートルグラフィックスを行うプログラミングを通して、プログラミングの入門を行っていく予定です[3]。

この科目で、タートルグラフィックス機能を使ったプログラムを作成する際には、ソースプログラムの冒頭に

```
#include <turtle.h>
```

の一行が必要になります。この行を書き忘れたり、書き間違えたりすると、ソースプログラムを正常にコンパイルすることができませんので注意してください。

`myfirst.c` に書かれた指示と、それが実行されることで描画される線との関係は、次の図のようになります。



[2] ピクセルと呼ぶこともあります。

[3] この科目で用いるタートルグラフィックスの機能は、この科目のために準備されたものであって、一般の C プログラミングの環境では利用できませんので注意してください。

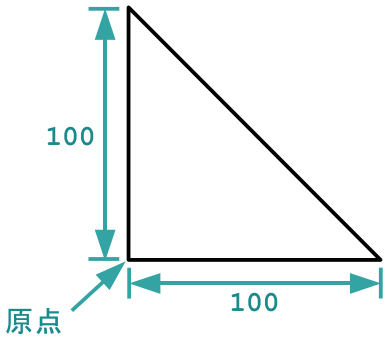
tForward(...) や tBackward(...)、tTurn(...) の ( ) の中の数値を変えたり、これらの指示を並べる順番を変えることで、いろいろな図形を描画することができます。

mysecond.c

```
#include <turtle.h>

main()
{
        tForward(100);
        tTurn(135);
        tForward(141.421);
        tTurn(135);
        tForward(100);
}
```

たとえば、この mysecond.c というプログラムをコンパイルして実行してみると、次の図のように、直角を挟む 2 辺の長さがそれぞれ 100 であるような直角二等辺三角形が描かれます。



C プログラムの main() に続く { と } との間には、実行させたい仕事をいくつも書くことができます。そこに書かれた順に書かれた仕事が実行されます。先の mysecond.c では、三角形を反時計回りに描きましたが、次の mythird.c というプログラムは、時計回りに描きます。tTurn(135); が実行されると、カメは**反時計回り**に 135° 向きを変えますが、tTurn(-135); の場合は**時計回り**に 135° 向きを変えることに注意してください。

mythird.c

```
#include <turtle.h>

main()
{
        tTurn(90);
        tForward(100);
        tTurn(-135);
        tForward(141.421);
        tTurn(-135);
        tForward(100);
}
```

メモ

1.3 数式の書き方

あらかじめプログラム中に書いておいた定数値だけではなく、数式の値を計算して、その計算結果を使った指示を書くこともできます。

mult.c

```
#include <turtle.h>

main()
{
        tForward(222);
        tTurn(135);
        tForward(222 * 1.41421);
        tTurn(135);
        tForward(222);
}
```

この mult.c をコンパイルして実行すると、直角を挟む 2 辺の長さがそれぞれ 222 であるような直角二等辺三角形が描かれます。プログラム中に現れている 222 * 1.41421 は、直角三角形の斜辺の長さを計算するために 222×1.41421 という数式を C 言語の書き方で表現したものです。

定数　C 言語では、-135 や 222 のような整数を表すデータと、1.41421 や 3.14159 のような実数値[4]を表すデータを区別して扱います。整数値の定数は、0 や 123 のように C プログラム中に書き、実数値の定数は 1.23 とか 3.45678 とか 1.23e4 のように書きます。最後の 1.23e4 は 1.23×10^4 (つまり 12300.0) のことです。同様に 1.23e-4 で 1.23×10^{-4} (つまり 0.000123) を表すことができます[5]。

3 と書くと整数値の定数を意味しますが、3.0 と書くと実数値の定数となります。数学的にはこの 2 つに違いがありませんが、プログラム中では、整数値の 3 と実数値の 3.0 は 別のもの[6]として扱われますので注意が必要です。

メモ

四則演算　C プログラムでは、数値の四則演算 (加減乗除) を表す数式を、それぞれ

$$a \texttt{ + } b \qquad a \texttt{ - } b \qquad a \texttt{ * } b \qquad a \texttt{ / } b$$

のように書きます。-a のように、数式の前に $-$ (マイナス) をつけることも可能です。2 つの整数値の間で四則演算を行うと、その計算結果も整数値となります。整数値を整数値で割る際には、余りは切り捨てられます。また、

$$a \texttt{ \% } b$$

[4]実際にはその近似値。

[5]1.23E4 や 1.23E-4 のように、大文字の E を使って書くこともできます。

[6]この 2 つは、コンピュータの内部では違った方法で表現されています。

と書くことで、a を b で割った余り (整数値) を計算することもできます。

2つの実数値の間で四則演算を行うと、その計算結果も実数値となります。また、整数値と実数値の間で四則演算を行う時には、まず整数値の方が実数値に変換されてから計算が行われ、計算結果も実数値となります。具体的な計算結果の例をつぎに示しましょう。

数式	計算結果
`7.6 + 3.2`	10.8
`7.6 - 3.2`	4.4
`7.6 * 3.2`	24.32
`7.6/3.2`	2.375

数式	計算結果
`1.0 / 3.0`	0.333333...
`1/3.0`	0.333333...
`1.0/3`	0.333333...
`1 / 3`	0

数式	計算結果
`1 / (1.0 + 2)`	0.333333...
`1 / (1 + 2)`	0
`(1 + 1.0) / 3`	0.666666...
`(1 + 1) / 3`	0

ただし、実数値の計算はあくまで近似値の計算ですから、たとえば `1.0 / 3.0 * 3.0` の計算結果と `1.0` は必ずしも等しいとは限らないことに注意してください。

メモ

演算子　`+` や `*` など、いくつかの数式を組み合わせて、また新たな数式を作る働きをするものを**演算子**と呼びます。演算子の両側の空白はあってもなくても数式としては同じ意味になります。つまり「`2+3*4`」と書いても「`2+3 * 4`」と書いても計算結果は全く同じです。日常での慣習と同様に `*`、`/`、`%` は `+` や `-` よりも先に計算されます。この順番を変えたいときには適当に `()` で式の一部を囲みます。C のプログラムでは、数式の中では中かっこや大かっこは使わずに、すべてこの `()` を使います。`()` は何重に使っても構いません。いくつか例を見てみましょう。

数式	計算結果
`- 3 - 7`	−10
`- (3 - 7)`	4
`2 + 5 * 3`	17
`(2 + 5) * 3`	21
`2 + 5 % 3 * 2`	6

数式	計算結果
`- (1 - (2 - (3 - 4)))`	2
`10 / 4.0 * 4`	10.0
`10 / 4 * 4.0`	8.0
`10.0 / 5.0 * 2.0`	4.0
`10.0 / (5.0 * 2.0)`	1.0

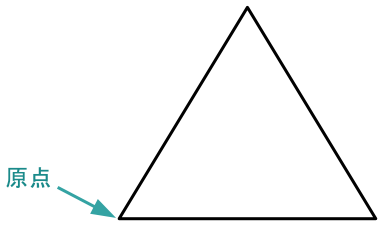
メモ

1.4 演習問題

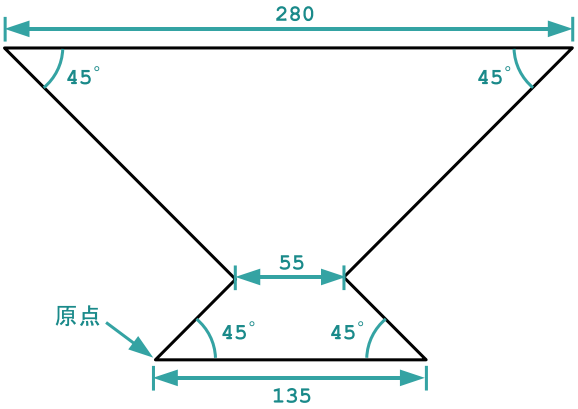
1. 1辺の長さが

$$\frac{1.1 - \{22 \times 33 + 44 \times (5.5 - 66)\} \times 77}{8.8 \times 99}$$

であるような正三角形を、次の図のように描く C プログラム `paren.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、この数式の計算はプログラムの中で行うようにしてください。



2. 次のような図形を描く C プログラム `cup.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、定数としてプログラム中に書けるのは 1.41421、2、55、90、135、280 の 6 種類だけとします。これら以外の定数は使わないでプログラムを書くように工夫してください[7]。

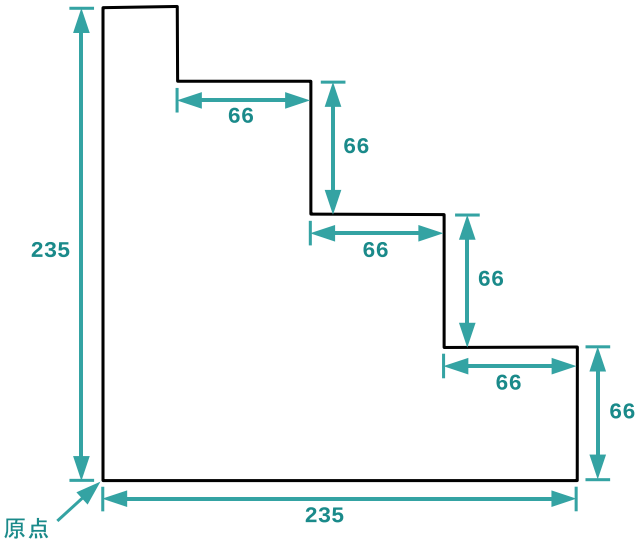


3. 次のような図形を描く C プログラム `remainder.c` を作り、コンパイル、実行して、正しく動作することを確認しなさい。ただし、定数としてプログラム中に書けるのは 66、90、235、270 の 4 種類だけとします。これ以外の定数を使うことはできません。また、使える四則演算子は *、/、% のみとします。加算 (+) と減算 (-) の演算子を使わないでプログラムを書くように工夫してください[8]。整数の除算の余りを計算する演算子を使いましょう。

---

[7] 90 が使えますので、-90 という数式を書くことはできます。

[8] この場合、- 演算子が使えませんので、-90 という数式を使うこともできません。



1.5 今回の実習内容

1. このプリントをもう一度ゆっくりと読み返しましょう。`empty.c`、`mysecond.c`、`mythrid.c`、`mult.c` の 4 つの例題プログラムについては、それぞれテキストエディタを使ってソースファイルを作成し、コンパイルして、実行してみましょう。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 ページからソースプログラムを提出してください。これを忘れると、課題を終えたことが記録されませんので注意してください。
2. 6 ページの演習問題の 1 〜 3 に取り組みましょう。プログラムが完成したら、それぞれ「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 1 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。

プログラミング及び実習 I ・第 1 回・終わり