プログラミング及び実習 I ・第 6 回

2025 年 7 月 10 日

配布資料の内容

6.1 文字列を使った出力と入力

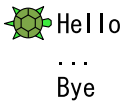
今回は、文字による出力と入力の方法について勉強します。今回紹介する `tPrintf` と `tScanf` という 2 つの関数は、この科目で使っているタートルグラフィックスの機能の一部で、一般の環境で利用できるものではありませんが、標準的な C 言語の環境には、同等の働きをする関数 `printf` と `scanf` が含まれていますので、ここで紹介する `tPrintf` と `tScanf` を、それぞれ `printf` と `scanf` に読み替えれば、普通の C 言語のプログラミング環境にも応用することができます。

文字列の出力　次のプログラム `hello.c` は、「Hello」、「...」、「Bye」という文字列を 3 行に分けてウィンドウに表示するものです。

hello.c

```
#include <turtle.h>

main()
{
    tPrintf("Hello\n");
    tPrintf("...\n");
    tPrintf("Bye\n");
}
```



一般に `tPrintf(" 文字列 ")` の形の関数呼び出し式を使うと 2 つの「"」(2 重引用符) の間に書いてある文字列を画面に表示することができます。ただし、この中に書いた `\n` は特別な意味を持っていて、ソースプログラム中では 2 つの文字として書かれていますが、この 2 つの文字の組合わせで「改行文字」と呼ばれる特殊な文字を表わしており、この文字が出力されることで、それ以降に出力される文字が表示される行が次の行に移ります。

メモ

ここで使われている「\」という記号は**バックスラッシュ**と呼ばれます。Windows 環境では、キーボードの「\」のキーを押しても入力できなかったり、「\」の代わりに半角の (幅の狭い)「¥」が表示されたりします。また、直接入力の状態で「¥」のキーを押すと「\」が表示されることもあります。日本向けの Windows の環境では、この 2 つの記号は同じ意味で使われることがあります。このプリントでは「\」と表記していますが、直接入力の状態でキーボードの「\」のキーを押して入力

してください。「\」のキーを押しても入力できない場合は、「¥」のキーを押してみましょう。画面には「\」ではなく半角の「¥」が表示されるかも知れませんが、Windows 環境では気にする必要はありません[1]。

一方、macOS では、「\」と半角の「¥」を区別して用いるのが普通です。「\」は通常「option」キーを押しながら「¥」のキーを押して入力します。

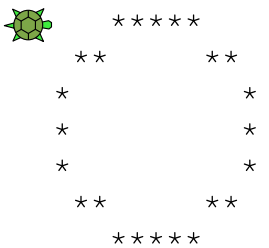
メモ

tPrintf を使った別の例を紹介しましょう。tPrintf の " と " との間に書かれたスペース (空白) がそのまま出力 (表示) されていることに注意してください。

starcirc.c

```
#include <turtle.h>

main()
{
    tPrintf("   *****\n");
    tPrintf(" **     **\n");
    tPrintf("*         *\n");
    tPrintf("*         *\n");
    tPrintf("*         *\n");
    tPrintf(" **     **\n");
    tPrintf("   *****\n");
}
```



整数値の出力　プログラム中にあらかじめ書いておいた文字列だけではなく、数値 (数式の計算結果) を出力することもできます。

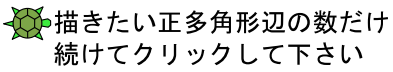
vpolygon.c

```
 1 #include <turtle.h>
 2
 3 main()
 4 {
 5     int i, n;
 6
 7     tPrintf("描きたい正多角形の辺の数だけ\n");
 8     tPrintf("続けてクリックして下さい\n");
 9     tGetClick();
10     n = tClickCount();
11     tPrintf("\n正%d角形を描きます\n", n);
12     tPenUp();
13     tBackward(100);
14     tPenDown();
15     i = 0;
16     while (i < n) {
17         tForward(50);
18         tTurn(360.0/n);
```

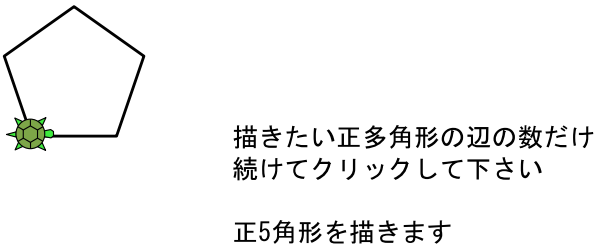
[1]全角の (幅の広い)「¥」は、これらの記号と区別されますので注意が必要です

```
19          i = i + 1;
20      }
21 }
```

このプログラムをコンパイルして、実行すると、まず、画面に



という文字列が表示されます。ここで、たとえば5回連続してマウスクリックすると、



のように「正5角形を描きます」という文字列が表示されて、カメは正5角形を描きます。

この「正5角形を描きます」の「5」の部分は、連続してクリックされた回数によって変わります。単に文字列を出力したいときには

```
tPrintf(" 出力したい文字列 ");
```

と書けばよかったわけですが、文字列の中に整数値 (変数の値や数式の計算結果など) を埋め込んで表示したい場合には、11 行目のように、整数値を埋め込みたい箇所に「%d」という暗号のようなものを書いておき、"出力したい文字列" の後に「,」で区切って (関数呼び出し式の第2引数として) 整数値を表す数式を書きます。tPrintf は %d 以外の部分はそのまま、%d の部分は「,」の後の数式を計算した結果 (整数値) を表示してくれます。もし % という文字自身を出力したい場合は %% と書いておきます。この2つの % に対して % が1つ出力されます。

メモ

tPrintf(のすぐ後に続けて書く文字列を**出力書式文字列**(あるいは**書式制御文字列**) と呼びます。C プログラム中に文字列を書く場合は「"」で囲むことになっていますので、出力書式文字列も「"」で囲んで書きます。もし「"」という文字自身を文字列の中に含めたい場合は「\"」の2文字を書いておきます。また、「\」という文字を文字列の中に含めたければ「\\」のように書きます。この2個の \ で、1文字分の \ を意味することになります。

tPrintf の出力書式文字列の中に %d があると、tPrintf は、そこに整数値を十進表記で埋め込んで出力します。どのような整数値が埋め込まれるかは、出力書式文字列の後に , で区切られて置

かれる数式の計算結果で決まります。出力書式文字列の中には複数の %d があっても構いません。その場合、埋め込まれる整数値は出力書式文字列の直後のものから順に使われます。

メモ

いくつか tPrintf を使った出力例を見てみましょう。

tPrintf の書き方	表示される文字列
例 1: tPrintf("計算結果は %d です。\n", -8+2);	計算結果は -6 です。
例 2: tPrintf("計算結果は -%d です。\n", 8+2);	計算結果は -10 です。
例 3: tPrintf("計算結果は -6 です。\n");	計算結果は -6 です。
例 4: tPrintf("%d足す%dは%d です。\n", 3, 4, 3+4);	3足す4は7です。
例 5: tPrintf("1+2+...+10=%d\n", 1+2+3+4+5+6+7+8+9+10);	1+2+3+...+10=55
例 6: tPrintf(" \"%d\"\n", 2+4+6-(1+3+5));	"3"
例 7: tPrintf("正答率は %d%% でした。\n", 63);	正答率は 63% でした。

tPrintf の出力書式文字列の中に並んでいる文字は、「\n」や「\"」、「\\」、「%d」、「%%」などを除いて[2]、そこに書かれている通りに出力されます。ですから、

```
tPrintf("5+8\n");
```

と書いた時に出力 (表示) されるのは「13」(と改行文字) ではなく、そこに書かれた通りの「5+8」(と改行文字) です。これに対して、

```
tPrintf("%d\n", 5+8);
```

と書けば、5+8 の計算結果である「13」(と改行文字) が出力されます。

メモ

実数値の出力　出力書式文字列の中に (%d の代わりに) %f と書くことで、(整数値ではなく) 実数値を出力することもできます。

real.c

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

[2]バックスラッシュ「\」で始まる「\n」や「\"」、「\\」と、パーセント記号「%」で始まる「%d」や「%%」とでは、文字がそのまま出力されない理由が異ります。先の 2 つは、C 言語のプログラム中で文字列を " で囲って表記するときの特例で、後の 2 つは tPrintf という関数の出力書式文字列での特例です。

```
main()
{
    double x, y;

    x = 20.0;
    y = 3.0;
    tPrintf("%f/%f = %f\n", x, y, x/y);
}
```

というプログラムをコンパイルして、実行すると


20.000000/3.000000 = 6.666667

のように、それぞれの実数値が小数点以下 6 桁まで表示されます。

出力書式文字列の中の %f に対応する引数は、それぞれ実数値を表す数式でなければなりません。同様に、出力書式文字列の中に書いた %d に対応する引数は、整数値でなければいけません。たとえば、つぎのような tPrintf の使い方は**間違い**です。

```
tPrintf("%f\n", 3);
tPrintf("%d\n", 3.0);
```

3 と書けば整数値を、3.0 と書けば実数値を意味することに注意してください。

%f の % と f の間に . と整数を書いておき、その整数で指定した桁数だけ小数点以下を表示することもできます。たとえば、tPrintf("%.3f\n", x/y); は、x/y の計算結果を小数点以下 3 桁まで表示します。%.0f と書けば、あたかも整数値のように見えるものが表示されますが、それでも、この %.0f に対応する引数は実数値を表す数式でなければならないことに注意してください。

メモ

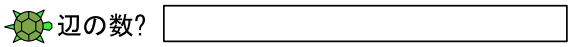
文字列による数値の入力　tPrintf を使って数値を文字列として出力できるのとは逆に、キーボードから入力された数値を入力 (変数に値を格納) することもできます。ここでは、この科目のタートルグラフィックスの機能の 1 つである tScanf を紹介します。

spolygon.c

```
 1 #include <turtle.h>
 2
 3 main ()
 4 {
 5         int i, n;
 6         double size;
 7
 8         tPrintf("辺の数? ");
 9         tScanf("%d", &n);
10         tPrintf("辺の長さ? ");
11         tScanf("%lf", &size);
```

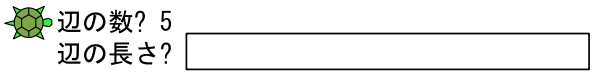
```
12          tPrintf("\n1辺%fの正%d角形を描きます\n", size, n);
13          tPenUp();
14          tBackward(100);
15          tPenDown();
16          i = 0;
17          while (i < n) {
18                  tForward(size);
19                  tTurn(360.0/n);
20                  i = i + 1;
21          }
22 }
```

このプログラムは、辺の数と辺の長さをキーボードから入力してもらい、指定されたような正多角形を描くものです。この spolygon.c をコンパイルして実行してみると、まず、

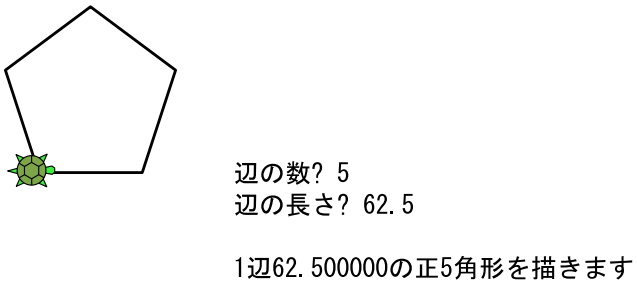


のように、8 行目の tPrintf の呼び出しによって表示された「辺の数? 」という文字列に続いて、キーボードからの入力を受け付けるための箱が表示されます。これは、9 行目の tScanf という関数呼び出しの働きです。

この箱にキーボードから整数 (たとえば 5) を入力して、Enter キーを押すと、箱は消えて「5」という数字が表示され、続く、10 〜 11 行目が実行されることで



のようになります。ここで、キーボードから辺の長さ (たとえば 62.5) を実数値で入力すると、



のように、1 辺の長さが 62.5 の正 5 角形を描きます。

メモ

整数値の入力　キーボードから入力された整数値を変数に格納するためには (このプログラムの 9 行目のように)、tScanf という関数を使って、

```
tScanf("%d", & 変数名 );
```

というような形の関数呼び出し式を書きます。この呼び出し方は、ちょっと tPrintf と似ています。tScanf の関数呼び出し式の最初の引数は**入力書式文字列**と呼ばれる文字列で、どのような書式の文字列がキーボードから入力されるのかを tScanf に教えるためのものです。spolygon.c のようにキーボードから整数値を 1 つ入力してもらう場合、入力書式文字列は "%d" になります。tScanf の入力書式文字列の後には、「,」で区切って「&」(アンパサンド) と変数名を書きます。この変数は整数値の変数 (int で宣言されたもの) でなければなりません。tPrintf を使って変数の値を出力する時には単にその変数名を書けばよかったわけですが、tScanf を使って入力された値を変数に格納する場合は変数名の前に必ず & を置かなければなりません。この & を書き忘れないようにしてください。

spolygon.c の 9 行目の

```
tScanf("%d", &n);
```

の部分が実行されると、画面にキーボードからの入力を受け付けるための箱が表示され、プログラムは入力待ちの状態になります。その状態でキーボードから整数値を入力して Enter キーを押すと、入力された整数値が変数 n に代入されます。これはちょうど

```
n = 入力された整数値 ;
```

が実行されたのと同じです。

tScanf による入力時に Enter キーが押されると、文字列の表示位置は新しい行に移動します。このため、10 行目の tPrintf("辺の長さ?　"); の出力は 1 つ下の行の左端から表示されています。

メモ

実数値の入力　キーボードからの実数値の入力も tScanf を使って行うことができます。ただし、入力された値を実数値の変数 (double で宣言されたもの) に格納する場合には、入力書式文字列には %lf と書かなければなりません。%lf の l は英小文字のエルです。整数値の場合は出力書式文字列でも入力書式文字列でも %d でしたが、実数値の場合は出力書式文字列では %f、入力書式文字列では %lf となることに注意してください。

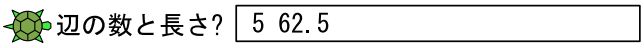
ソースプログラムの中では、3 と書けば整数値として、3.0 と書けば実数値として扱われますが、入力書式文字列に %lf と書いておけば、tScanf は、キーボードから入力された文字列が 3 であろうと 3.0 であろうと、実数値として指定した変数に格納してくれます。

メモ

複数の数値の入力　tScanf を使うと、入力された複数の数値をそれぞれ別の変数に格納することもできます。たとえば、spolygon.c の 8 ～ 11 行目を

```
tPrintf("辺の数と長さ? ");
tScanf("%d%lf", &n, &size);
```

と変えると、

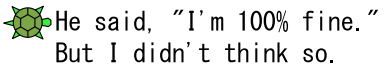


のように、箱の中に 2 つの数値をスペースで区切って入力することで、順に、それぞれ変数 n と size に格納することができます。tScanf の入力書式文字列の後に、複数の変数を指定していますが、それぞれ & が必要なことに注意してください。入力書式文字列の中に 3 個以上の %d や %lf を書いておけば、1 回の tScanf の呼び出しで、3 個以上の数値を入力することもできます。

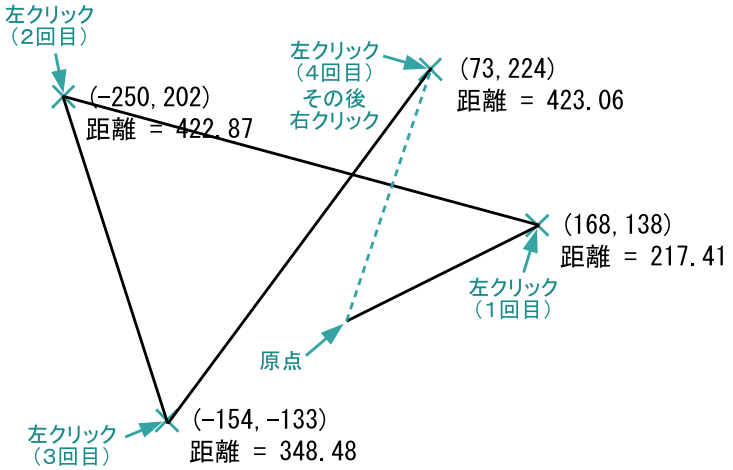
メモ

6.2　演習問題

1. 「He said, "I'm 100% fine."」と「But I didn't think so.」という 2 つの英文を、次のように (カメを動かさずに) 表示するプログラム fine.c を作成し、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。記号やスペースや含めて、全く同じ文字列が表示されるように注意してください。この 2 つの英文に使われている文字や記号はすべて半角 (幅の狭い) 文字です。

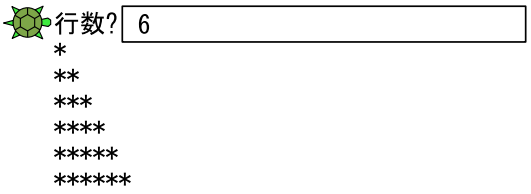


2.　ウィンドウ内をマウスの左ボタンでクリックする度に、カメがクリックされた位置に向かって線分を描き、クリックされた座標と移動距離 (描いた線分の長さ) を下図のように表示するプログラム vfollow.c を作成し、コンパイル、実行して、正しく動作することを確認しなさい。ただし、マウスの中ボタンや右ボタンが押されると、カメは仕事をやめて (ペンを上げて) 原点に戻るようにしなさい。また、移動距離は小数点以下 2 桁まで表示するようにしなさい。



ヒント: 第 4 回の例題プログラム `follow.c` や `follow1000.c` を改造するのが簡単です。関数 `tMoveTo` を呼び出して移動すれば、そのとき移動した距離を戻り値として知ることができます。

3. 下図のようにキーボードから非負の正数 n を入力すると、「*」を、1 行目には 1 個、2 行目には 2 個、3 行目には 3 個、$\cdots$、n 行目には n 個並べて表示するようなプログラム `asterisks.c` を作成し、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。このプログラムでは、カメはまったく移動しません。「*」はすべて半角 (幅の狭い) のアスタリスクです。図中の横長の長方形 (キーボードからの入力を受け取る枠) は、実際には Enter キーを押した時に消えてしまっているはずです。



ヒント: n の値を読み込んだ後、n 行に亘って「*」を表示する部分は、二重の繰り返しの処理で実現しましょう。`tPrintf("*");` を繰り返し実行することで「*」を横に並べることができます。また、`tPrintf("\n");` を実行すれば、文字を表示する位置を次の行の先頭に移動することができます。

6.3 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。`hello.c` と `spolygon.c` の 2 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、実行して、正しく動作することを確認してください。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。前回までと同様に「課題の提出と確認」の Web ページで「第 6 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。

プログラミング及び実習 I ・第 6 回・終わり

付録: 前回の演習問題のプログラム例

rightmost.c

```
#include <turtle.h>

main ()
{
    int n, x, y, xmax, yxmax;

    tPenUp();
    n = 0;
    while (tGetClick() == 1) {
        x = tClickX();
        y = tClickY();
        tMoveTo(x, y);
        tPenDown();
        if (n == 0 || x >= xmax) {
            xmax = x;
            yxmax = y;
        }
        n = n + 1;
    }
    if (n > 0) {
        tPenUp();
        tMoveTo(xmax, yxmax);
        tPenDown();
        tMoveTo(0, 0);
    }
}
```

mesh.c（その 1）

```
#include <turtle.h>

main()
{
    int i, j, k, n;
    double s;

    tGetClick();
    n = tClickCount();
    tPenUp();
    tMoveTo(tClickX() - 100, tClickY() - 100);
    tPenDown();
    s = 200.0 / n;
    i = 0;
    while (i < n) {
        tTurnTo(0);
        j = 0;
        while (j < n) {
            k = 0;
            while (k < 4) {
                tForward(s);
                tTurn(90);
                k = k + 1;
            }
            tForward(s);
            j = j + 1;
        }
        tBackward(200);
        tTurn(90);
```

```
        tForward(s);
        i = i + 1;
    }
}
```

mesh.c（その 2）

```
#include <turtle.h>

main()
{
    int x, y, i, n;

    tGetClick();
    n = tClickCount();
    x = tClickX() - 100;
    y = tClickY() - 100;
    i = 0;
    while (i <= n) {
        tPenUp();
        tMoveTo(x + 200.0 * i / n, y);
        tPenDown();
        tMoveTo(x + 200.0 * i / n, y + 200);
        i = i + 1;
    }
    i = 0;
    while (i <= n) {
        tPenUp();
        tMoveTo(x, y + 200.0 * i / n);
        tPenDown();
        tMoveTo(x + 200, y + 200.0 * i / n);
        i = i + 1;
    }
}
```

mesh.c（その 3）

```
#include <turtle.h>

main()
{
    int x, y, i, n;

    tGetClick();
    n = tClickCount();
    x = tClickX() - 100;
    y = tClickY() - 100;
    tPenUp();
    tMoveTo(x, y);
    tPenDown();
    tMoveTo(x + 200, y);
    i = 0;
    while (i < n) {
        if (i % 2 == 0)
            tTurn(90);
        else
            tTurn(-90);
        tForward(200.0 / n);
        if (i % 2 == 0)
            tTurn(90);
        else
```

```
            tTurn(-90);
        tForward(200);
        i = i + 1;
    }
    if (i % 2 == 0)
        tTurn(-90);
    else
        tTurn(90);
    tForward(200);
    while (i < 2 * n) {
        if (i % 2 == 0)
            tTurn(-90);
        else
            tTurn(90);
        tForward(200.0 / n);
        if (i % 2 == 0)
            tTurn(-90);
        else
            tTurn(90);
        tForward(200);
        i = i + 1;
    }
}
```