基礎プログラミング演習 II 教材 (#4)

■ データ型（実数と整数）

（教科書 p.49「型」を参照）

□ 型変換、キャスト

まず定数の表記ですが、1, 2, 100 などは整数、1.5 など小数点が付けば実数として扱われます。つまり 2 は整数ですが 2.0 は実数となります。整数と整数の演算（ex. 1+100）結果は整数、実数どうし（10.0+20.5）は実数となります。問題はその「型」が混在するときです。

int 型変数は整数しか扱えません。そこで int 型変数に実数を代入すると、自動的に型変換が行われ、小数部が切り落とされて整数部だけが代入されます。
つまり `int i; i=123.456;` なら、i には少数以下が切り落とされた 123 が代入されます。

実数変数に整数を代入した場合は、単純に実数化された値が入ります。
`double a; a=123;` なら、123 が実数化されて 123.0 となって a に代入されます。

計算式に整数、実数の値や変数が混在していた場合は、精度の高い方に合わせて型変換が行われてから計算が行われます。`double a; a=1.5 * 3;` なら、1.5 * 3 は 1.5 * 3.0 として計算され、4.5 が a に代入されます。

注意が必要なのは /（割り算）で、整数と整数の割り算は整数となって余りは捨てられますが、実数と実数、または実数と整数の割り算では可能な限りの精度で小数点以下まで求められます。

つまり 10 / 4 は 2 ですが、10.0 / 4.0 は 2.5 です。変数を用いた計算でも同じことで、

```
int i, k; double a, b;
i=10; k=4; a=10.0; b=4.0;
```

の場合、i / k は 2 ですが、a / b は 2.5 です。i / b や a / k のように実数と整数を混在させた場合は、整数側が実数化されて計算され、結果はともに 2.5 になります。

これらは暗黙の型変換と呼ばれますが、明示的に指示して行うこともできます。
もし、i / k の計算を実数化して行い、2.5 という結果を実数変数に代入したい場合は、

```
double x; x = (double)i / (double)k;
```

と書きます。
この、値の直前に () で囲んで型名を明示指定する方法を「キャスト」と呼びます。

> キャストの有効範囲がどこまでか不安になるような場合があり得ます。例えば、
> `x = (int) a / k * 100;` のようなケースは、a だけキャストされるのか、全体の計算結果に最後に一度だけキャストされるのか、不安に思うかも知れません。そうしたときは、
> `x = (int)(a / k * 100);` のように、カッコをうまく使って記述すると良いでしょう。

□ 定数における型の明記

123.456 は実数でも double 型とみなされます。float 型と明記したい場合は 123.456f と最後に f をつけて表記します。（サンプルプログラムの f = 300.0f; を参照。）
実数はまた 1.2e-5; というように表記できます。（ 1.2×10^{-5} つまり 0.000012 ）
（興味のある受講生は教科書 p.38 も参照。整数定数の 8, 16 進表記法がある。）

□ データの型に合わせた変換文字

printf() を利用する際はデータの型に合わせて変換文字を指定しなければなりません。整数は %d, 実数は %f です。

実験：右のプログラムを入力して実行し、その結果を表示させて結果を確認してください。また、プログラムを修正して、i/30, i/30.0, i*d, i/30*d, i/d*30 がそれぞれどのような結果になるか試してください。納得できますか？
（他にも例えば整数を %f で表示させると？その逆では？なども試してください。）

```
short s;
int i;
float f;
double d;

s=100;       i=200;
f=300.0f;   d=400.0;

printf("short  %d\n",s);
printf("int    %d\n",i);
printf("float  %f\n",f);
printf("double %f\n",d);
```

□ 変数の型と精度

変数の型によって格納できる値の範囲・精度が異なります。

実験：精度（有効桁数）が型によって異なることを確認しましょう。上のプログラムに精度を超えると想像できる値を指定して結果を見てください。（教科書 p.49 表 3-1 参照）

□ データの型に合わせた変換文字列と桁数指定

整数は %d, 実数は %f あるいは %e が利用できます。また、%10d のように、% と変換文字の間に桁数の指定などができます。以下に代表的な変換文字列を示します。

	意味	使用例	その結果
%d	整数を表示	printf("[%d]\n",10);	[10] 桁数不定
		printf("[%5d]\n",10);	[10] 5 桁で表示。不足分は空白。
		printf("[%05d]\n",10);	[00010] 5 桁。不足はゼロで埋める。
%f	実数を表示	printf("[%f]\n",12.345);	[12.345000] 桁数不定
		printf("[%9.5f]\n",12.345);	[12.34500] 小数点含めて全体が 9 桁、小数以下が 5 桁。
%e	実数を表示	printf("[%e]\n",12.345);	[1.234500e+01] 浮動小数点で表示

また、scanf( ) 関数で実数型に値を入力する場合は、float 型変数であれば %f、double 型であれば %lf を変換文字として指定します。（つまり明確に型を合わせなければなりません）

```
float f; double d;
scanf(“%f %lf”, &f, &d);
```

□ 誤差：if 文による判定

右のプログラムは 100 回ループすると停止するように見えますが、実際には if 文の条件は成立しません。
しかし 0.1 ではなく 1.0 ずつ加算すれば停止します。
これは 0.1 が 2 進での浮動小数点表現では無限小数（割り切れない数）になるために生じる誤差が原因です。
制御文字を %50.45f などとして printf() すれば確認できます。

```
double d;
d=0.0;
while(1) {
  if(d == 10.0) break;
  printf("%f\n", d);
  d+=0.1;
}
```

このような場合に対処するため、実数では同一性判定は一定の範囲を定めて行うのが安全です。
`double e; e=0.1e-12;` などと非常に小さな値を用意して、
`if((d > 10.0-e)&&(d < 10.0+e)) break;` などとして判定します。

■ 数学関数

□ 数学関数の使用例

C 言語には sin( ) 関数や cos( ) 関数など、数学的な計算を行ってくれる関数がひと揃い用意されています。こういった関数を数学関数と呼んでいます。代表的な関数としては以下のようなものがあります。

sin　正弦を求める	sqrt　平方根を求める
cos　余弦を求める	pow　累乗を求める
tan　正接を求める	log　対数を求める
fabs　絶対値を求める	exp　指数を求める

他の関数、詳しい使い方などについては教科書 p.466 math.h を参照すると良いでしょう。

```
   y = sin(theta);
```

のようにして使えば y に角度 theta の時の正弦を計算してくれます。

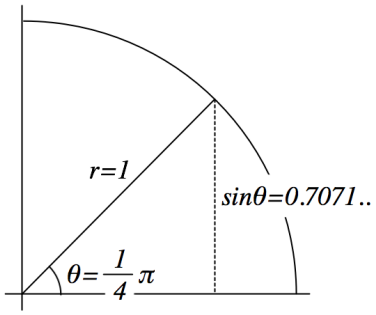
ただし sin 等の三角関数に与える引数は、
・ double 型の実数であること
・ 角度は、ラジアン単位（弧度）で指定する
ことに注意して下さい。
sin 関数の戻り値は double 型で返されます。

C 言語における三角関数はどれもラジアン単位（弧度法）です。（度数法で言う 360 度を 2π ラジアンとする）

右図は辺の長さ 1、角が 1/4π であるときの sin の値が約 0.7071 であることを示しています。(角 1/4π は度数法の 45 度に相当)
C プログラムでは以下のように記述して sin 1/4π を得られます。

```
   printf("%8.5f\n", sin(3.14/4.0));
```

結果は 0.70683 と表示されますが、結果の有効数字に注意が必要です。



これが√2 の逆数にあたることも、やはり数学関数 sqrt() を利用して確認することができるでしょう。

```
   printf("%8.5f\n", sqrt(2.0));
```

□ math.h ヘッダ・M_PI 定数

数学関数を使う場合は、先頭に以下の一行を書いて下さい。

```
#include <math.h>
```

これによって数学関数を定義したヘッダファイル math.h がプリプロセッサによって取り込まれます。Termnal で man 3 sin などとすると、どの関数がどのヘッダを必要とするか分かります。
また幾つかの数学関数で利用できる定数が定義されます。例えば円周率は下記のように定義されています。

```
#define M_PI  3.14159265358979323846264338327950288
```

これを利用して以下のように記述すると、精度はかなり高くなります。

```
   printf("%8.5f\n", sin(M_PI/4.0));
```

□ 参考：-lm オプション

MacOSX では不要ですが、幾つかのコンパイラでは -lm オプションが必要になります。

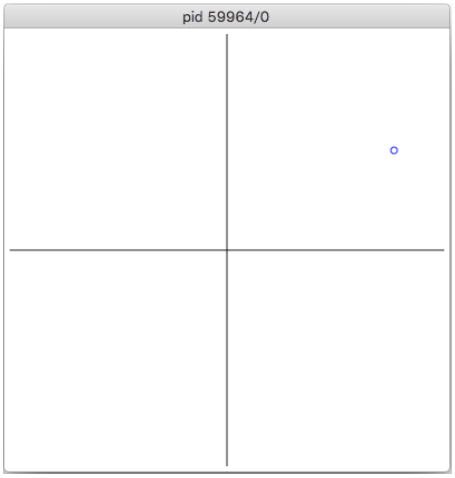
```
  $ cc sample.c -lm
```

これは数学関数のためのライブラリを使う指示で、これがないと下記のように「sin という名前は未定義 (undefined) である」といったエラーが出る場合があります。

```
$ cc –o sample sample.c
/tmp/ccHZhcFU.o: In function `main':
/tmp/ccHZhcFU.o(.text+0x3f): undefined reference to `sin'
collect2: ld returned 1 exit status
$
```

□ グラフの描画

右に(150, 90) の位置に小さな円を表示するプログラムと、その実行結果を示す。

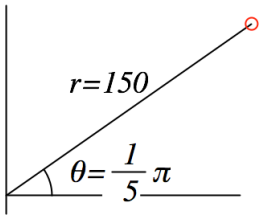



```
#include <stdio.h>
#include <math.h>
#include <handy.h>

int main() {
    double x, y;

    HgOpen(400.0, 400.0);
    HgSetWidth(1.0);
    HgSetColor(HG_BLACK);
    HgLine(5.0, 200.0, 395.0, 200.0);
    HgLine(200.0, 5.0, 200.0, 395.0);

    x = 150.0;
    y = 90.0;
    HgSetColor(HG_BLUE);
    HgCircle(x + 200.0, y + 200.0, 3.0);

    HgGetChar();
    HgClose();
    return 0;
}
```

押さえて欲しいポイント：
・ HgCircle( ) 関数で半径 3 の円を描くことで、計算した座標位置に目印を打っています。
・ 原点をグラフィクスウィンドウの真ん中（200, 200）に置いているので、HgCircle( ) 関数の x, y 座標位置指定にはそれぞれ 200.0 を加算しています。

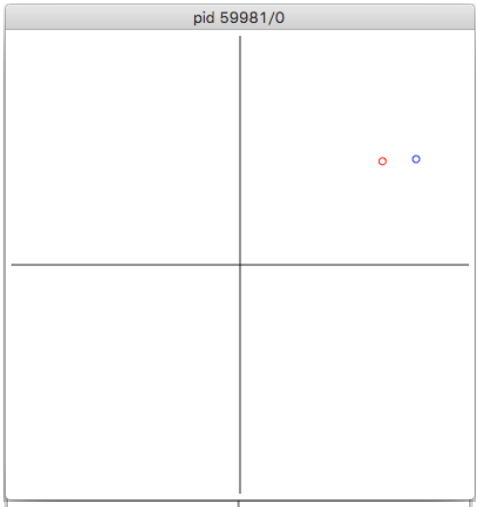
□ 課題 1. 三角関数を用いた描画

右図のように辺の長さが画素単位で 150、角が 1/5π の位置に赤い円を描画するように、上のプログラムに記述を追加せよ。

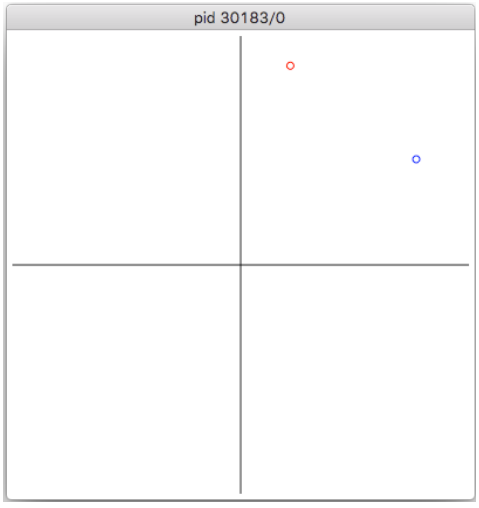


描画するべき x 座標、y 座標の値は三角関数を用いて求めることができることが分かるだろうか？

元の青い円（150, 90）の位置に対してどのあたりに表示されるか分かるように、実行結果も右につけておく。



□ 課題 2. 座標位置の回転



点 (x, y) について、原点を中心に θ だけ回転させたときの位置 (x', y') は以下のようにして求めることができます。

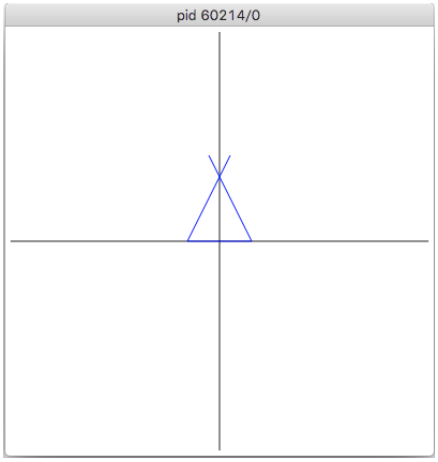
$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

この計算式を用いて (150, 90) 座標位置を 1/4π だけ回転させた位置に赤い円を描画せよ。

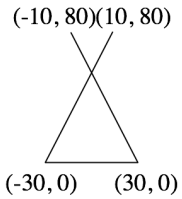
□ 課題 3. 連続した回転

課題 2. で試した回転操作を連続的に行って描画せよ。
右では半周にかけて、複数回、回転移動させた。回転角は 1/10π ずつとしている。

□ 複数の座標点を線でつなぐ



下図のような４つの頂点からなる図形を描画するプログラムを以下に示す。

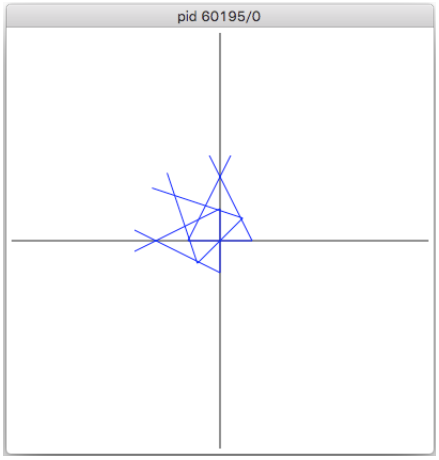



```
double x[4] = {-10.0, 30.0, -30.0, 10.0};
double y[4] = { 80.0,  0.0,   0.0, 80.0};
int i;

for(i=0; i<3; i++) {
    HgLine(x[i]   + 200.0, y[i]   + 200.0,
           x[i+1] + 200.0, y[i+1] + 200.0);
}
```

押さえて欲しいポイント：
・ x, y それぞれの配列にデータを格納
・ ループによって二つ連続する配列要素の間を HgLine( ) でつないで辺を描画
・ 頂点が３つしかないのに配列要素が４つあるのは、最終に最初の点の座標位置をもう一度入れているため
・ こうすることで単純な３回ループするだけのコードで３つの辺を描画できる

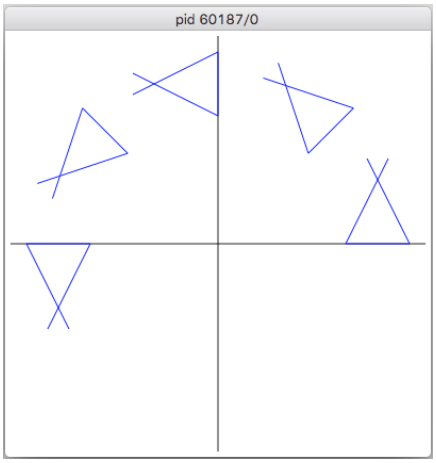
サンプルコードが教材 Web にあるので、ダウンロードして実行し、結果を確認せよ。

□ 課題 4. 複数座標点の回転



右図（左側）のように、この図形を何度か回転させて描画せよ。
座標原点が図形の(0, 0)位置と重なっていることを意識すると何が起きているか分かりやすいだろう。

□ 課題 5. 描画位置ごとの回転



右図（右側）のように図形の描画位置と図形の傾きの両方を変化させて描画せよ。
まず図形の(0, 0)の位置が全体座標系でのどこになるか求め、そこを基点に回転した図形の各頂点の位置を求めれば良い。

回転する径の大きさは任せます。うまく回転処理ができていると確認できれば良いです。

□ 参考：HgLines( ), HgPolygon( )

```
double x[4] = {-10.0, 30.0, -30.0, 10.0};
double y[4] = { 80.0,  0.0,   0.0, 80.0};
HgLines(4, x, y);
```

HgLines( ) 関数を用いても、複数の頂点からなる図形を描画することができます。

```
HgLines( 頂点数, X 座標用配列名, Y 座標用配列名 );
```

HgPolygons( ) 関数は配列中の最後の要素と最初の要素を結んだ（閉じた）多角形を描画します。

```
HgPolygon( 頂点数, X 座標用配列名, Y 座標用配列名 );
```