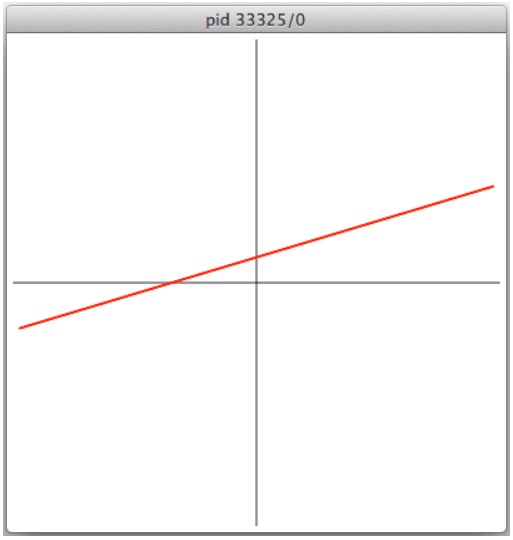
基礎プログラミング演習 II 教材 (#7)

■ 文法的落ち穂拾い：関数における引数の型

以下のプログラムは、ドット（小さな箱）を繰り返して描くことで斜め線を作るものです。このプログラム中で、y 軸座標位置を計算する際に整数変数 x と実数変数 a, b との「型が異なる」演算が行われています。




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

int main() {
  int x;
  double y, a, b; // y = ax + b

  HgOpen(400.0, 400.0);
  HgSetWidth(1.0);
  HgSetColor(HG_BLACK);

  HgSetFillColor(HG_WHITE);
  HgLine(5.0, 200.0, 395.0, 200.0);
  HgLine(200.0, 5.0, 200.0, 395.0);

  a=0.3;
  b=20.0;
  HgSetColor(HG_RED);
  for(x=-190; x<190; x++) {
    y=a * x + b;  <<<< この行！
    HgBox(x + 200.0, y + 200.0, 1.0, 1.0);
  }

  HgGetChar();
  HgClose();

  return 0;
}
```

また、関数の引数（カッコの中に与えるパラメタ、sin(x) であれば x の部分に相当）にも型があります。また、戻り値にも型があり、プログラマはこれらを意識して使用すべきです。つまり絶対値を求める関数のひとつ abs( )は引数、戻り値ともに整数型です。このため、実数の絶対値を求めることはできません。実数向けには double 型に対応する fabs( )が用意されています。なお float 型は fabsf( ) です。

サンプルでは暗黙の型変換が働くことを前提に、整数型のデータを特に double 型にキャストせずに書いています。

型の相違について、暗黙の型変換で済ませるか明記するかは、プログラマに任せられています。ただ、右の例ほど厳密に書いてある場面には余り遭遇しません。

```
y=a * (double)x + b;
HgBox((double)x + 200.0, y + 200.0, 1.0, 1.0);
```

つまり、余り厳密に型を合わせて書かなくても良いが、問題が生じたときに型違いの可能性を疑える程度には意識しておくべきだ、と考えて下さい。その一つの例が既に説明した、scanf() のケースです。

scanf( ) 関数で実数型に値を入力する場合、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);

この段階でもう一度「#3 データ型 整数と実数」の「データの型に合わせた変換文字列と桁数指定」の部分を見直すことを勧めます。

■ グラフィクス画面への文字の描画

右のサンプルプログラムはグラフィクス画面へ文字を描画するものです。

```
char string[100];
printf("input string >> ");
scanf("%s", string);
HgOpen(400.0, 400.0);

HgText(100.0, 200.0, string);

HgGetChar();
HgClose();
```

HgText( ) 関数の引数を以下に示します。

HgText(x, y, string)

・x, y ：座標位置（double 型）

・string：描画する文字列（char 型の配列）

文字サイズは HgSetFont( ) 関数で指定します。HgSetFont( ) では同時に字体も設定します。

HgSetFont(フォント名, size)

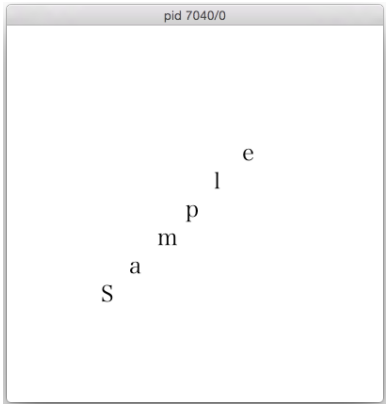
フォント名は右表のマクロ名を指定してください。サイズは double 型で与えます。

フォント	細字体	斜体	太字体	太字斜体
Times	HG_T	HG_TI	HG_TB	HG_TBI
Helvetica	HG_H	HG_HI	HG_HB	HG_HBI
Courier	HG_C	HG_CI	HG_CB	HG_CBI
明朝	HG_M		HG_MB	
ゴシック	HG_G		HG_GB	

詳しい仕様は HandyGraphic のマニュアルを参照してください。
http://www7a.biglobe.ne.jp/~ogihara/Hg/hg-jpn.html

□ 課題 1.

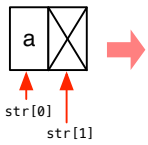
入力した文字列を一文字ずつに分解して、斜めに描画するプログラムを作ってください。



HgText 関数は文字型のデータを描画する機能がなく、文字列（文字配列）であることが必要です。文字列（文字配列）から一文字だけ取り出し、その文字だけの文字列を作って HgText に与えると良いでしょう。

参考：
「一文字だけの文字列」を格納するためには、終端文字を含めて 2 要素用意することになります。

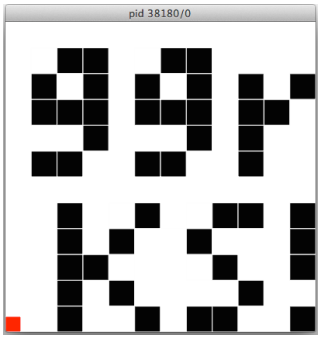
`char str[2];` と宣言すると2文字ぶん用意される



中身を `str[0]` が `'a'`
`str[1]` が `'\0'`
とすれば「一文字だけの文字列」ができる。

詳しい説明が補助資料にもあります。教科書の説明（p.203〜）で良く分からなかった人は見てみると良いかもしれません。

■ 課題 2. ドットエディタ（二次元配列）



右図のようなドットエディタを作ります。クリックした位置に応じた領域が白黒反転するものです。

□ クリックした位置を検出するサンプル

以下にクリックした位置を取得するコードを示します。
（基礎プロ I の「高度なテクニック集」で試した人も多いでしょう）

```
int main() {
    hgevent *event;
    double x, y;

    HgOpen(400.0, 400.0);
    HgSetWidth(1.0);
    HgSetColor(HG_BLACK);

    HgSetEventMask(HG_MOUSE_DOWN);
    for(;;) {
        event = HgEvent();
        if (event->type == HG_MOUSE_DOWN) {
            x=event->x;
            y=event->y;
            printf("x=%5.2f, y=%5.2f\n", x, y);
            HgBox(x, y, 5.0, 5.0);
        }
    }
    // このプログラムは無限ループして終了しない
}
```

これも

赤文字は定型処理

この位置にクリックに対応した処理を書く

赤文字の部分はマウスクリックの情報を取得する定型処理なので内容は気にしない。（技術的にもポインタや構造体を使っており、基礎プロ II では扱わない。）

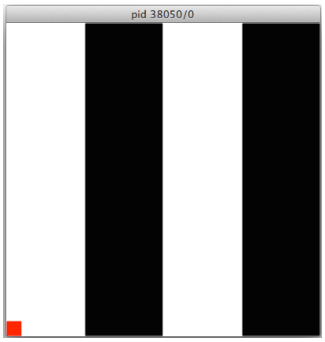
- 青文字の位置に自分がクリックに対応した処理を記述すれば良い。
- その時点では変数 x, y にクリックした座標位置が入っている。
- このコードではクリックした位置に小さな四角形を描画する。
- プログラム終了は Command キーを押しながら Q を押して操作。
（エラー表示が出るが気にしない）

このプログラムを動かしながら、確かにクリックした場所の情報が取れており、クリックした位置が起動したターミナル画面に表示されていることを確認してください。また、それが上に解説したとおりの挙動であることが納得できますか。

□ ステップ 1：終了機能を追加する

このページ右上の図のように、1) 画面の左下隅に小さな四角形を描画し、2) その範囲をクリックしたらループを抜けて終了する ようにプログラムを修正して機能追加してください。
領域範囲の判定は基礎プロ I の「条件分岐と繰り返し」の回の課題 1.「領域判定」でやりましたね？

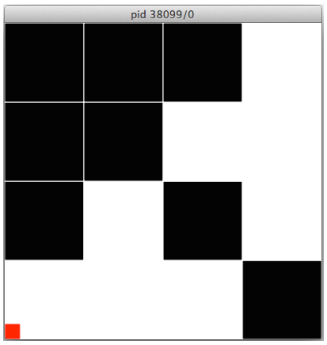
□ ステップ 2：画面を分割し、どの部分をクリックしたか求める



右図のように、画面を縦長のタイルに四分割して、それぞれのタイル範囲をクリックしたら、色が変わるように作って下さい。
1. タイルの色の情報（白か黒か）を保持する一次元配列を一つ用意し、
2. クリックした x 座標によって何枚目のタイルに触れたか判定し、
3. 対象となったタイルに相当する配列要素の色情報を違うものに変更する

ように修正すればよいでしょう。

□ 最終ステップ：タイルを縦横二次元に配置する



このページ右上の図のように、タイルを 4x4 の二次元に配置してください。
1. タイルの情報を二次元配列で持つ（教科書 p.199 末尾以降参照）
2. クリックした場所を x, y の両方で判定し、処理する

ように修正するだけで、それっぽいドットエディタになるでしょう。
分割数を 12x12 などにすると、かなりちゃんとした絵が描けます。（このページの右上肩のものがそれ）