Media Programing 1 - openFrameworks

- frame.01 - 2010.04.20 hrr

目的

openFrameworksを使って、プログラミングの概念と要素を学ぶ。
プログラミングの基本がわかれば、他の言語（Max, C, C++, perl,python)でもきちんとしたプログラムが作れるようになる。
openFrameworksらしいプログラムではなく、オブジェクト指向プログラミングやメディアの扱いの基礎的な学習を中心とします。本当の基礎はやりません。

この授業では、まずは手を動かしてください。
少しでもコードを書いて、書く事に慣れてください。
授業内でも、どんどん書いてもらいます。

内容

openFrameworks 全６回

- ・プログラムの基礎の復習
- ・基本的なアルゴリズム、オブジェクト指向
- ・マウス、キーボード、衝突判定
- ・ネットワーク
- ・画像処理
- ・グラフィック処理、音声処理

最終回のテスト
最終課題：対戦型ゲーム作品？　ネットワーク型作品？

afp://fs.iamas.ac.jp/share/MediaProgramming_1_20100930/openFrameworks/

開発環境

XCode 3.x or VisualStudio2008 + openFrameworks0061

openFrameworks:
http://www.openframeworks.cc/

日本語サイト
http://openframeworks.jp

参考文献
「Beyond Interaction ―メディアアートのためのopenFrameworksプログラミング入門 」

田所　淳・比嘉　了・久保田晃弘、BNN、3,570円

今日のテーマ

開発環境の整備
プログラミングの基礎を復習。

開発環境の準備

fs.iamas.ac.jpまたは、openframeworksのWebページから最新のopenFrameworksを
ダウンロードし、任意の場所に展開する。（現バージョンは0061）

MacOSの場合

fs.iamas.ac.jpから、LeopardまたはSnow Leopard用のテンプレートファイルを
コピーし、展開する。

展開したフォルダを
/ユーザー名/ライブラリ/Application Support/Developer/Shared/Xcode/Project Tempaltes
\openFrameworks\
にコピーする。
Project Templates フォルダが無かったら作成し、さらにopenFrameworksフォルダも作る。

（あるいは、共用マシン等で複数人でXCodeを使う場合には、
/Developer/Library/Xcode/Project Templates/　　以下にコピーしてもよい。）
openframeworks.jpのテンプレートを使っても良いでしょう。

テンプレートの作り方をTemplate作成.rtfに置いておきます。

Windows(VisualStudio2008)の場合

テンプレートは特にありません。

プロジェクトフォルダの作成

openFrameworksのフォルダ内に MyApps というフォルダを作成する。
MyApps フォルダ内に MediaPrograming1 というフォルダを作成する。
このフォルダ内に新しいプロジェクトを作成してください。
openFrameworksで使うライブラリファイルとの相対位置の関係で、
この階層にないとプログラムが作成できません。
これは、libsフォルダとapps/examplesの階層関係と同様にするようにするためです。

プロジェクトの作成（MacOS)

新しいプロジェクトを作成するには、XCodeから新規プロジェクトを選び
openFrameworks を選択して、新しいプロジェクトファイルを生成する。

この際、上で作成したMediaPrograming1フォルダを選択し、その中に作るようにしてください。

プロジェクトの作成(Visual Studio2008)

新しいプロジェクトを作成するには、readme.txtにあるように
emptyProjectフォルダを複製し、名前を付ける（ここではnewProjectとする）。
emptyProject.vcproj とemptyProject.sln を　newProject.vcproj ,newProject.slnに変更する。
newProject.vcproj をテキストエディタで開く。
Name と RootNamespace にあるemptyProject を newProject に変更する。
newProject.sln をテキストエディタで開く。
emptyProject.vcproj と emptyProject を newProject.vcproj と emptyProjectに変更する。

SatE-Oさんが、VC用のツールを公開されていますので、これを使うのも良いでしょう。
http://zampoh.cocolog-nifty.com/blog/2010/04/of---openframew.html

openFrameworksを使ってみる

その前に：openFrameworksで用意されているライブラリについては、
http://www.openframeworks.cc/documentation
に書いてあります。
ここを適宜参照しながら、プログラムを進めてください。

新規プロジェクトの作成して適当に名前を付ける。
XCodeのグループとファイルのsrcフォルダの中にあるtestApp.cpp を開く。
このファイルを編集していきます。

まずは、四角いエリアを作成し、真ん中に円を描く。
セミコロンは、一つの命令の区切り（一つの命令文の終わり）として重要。
忘れないように！

```
void testApp::setup(){

}
```

の中に書いていきます。

```
ofSetWindowShape(400,400);    // ウインドウの大きさを決める
ofBackground(0,0,0);          // 背景色を決めるには、
ofSetColor(255,0,0);            // 描画色を決める
```

次に円を描きます。

```
void testApp::draw(){
```

}
の中に

```
ofCircle(50.0, 50.0, 50.0);        // (50,50)に半径50の大きさの円を描く。
```

ここまでで、testApp.cppを保存し、ビルとと実行を押してみる。
プログラムが実行されるのを確認します。

確認できたら、ついでに、四角も書いてみる。

```
ofRect(100.0,100.0, 50,50);    // (100,100) に幅50,高さ50の四角を描く。
```

実行して確認する。

次に色を変えて線を書いてみる。

```
ofSetColor(0,255,0);            // 色を緑にする。
ofLine(200,200,300,300);        // (200,200)から(300,300)に線を引く。
```

実行して確認してみる。

全部が緑で描画されます。
なぜか？
setup()はプログラムの最初の一回だけ実行されます。
draw()は常に繰り返して実行されています。
つまり、ここの毎回違う物を描画するとアニメーションになります。
draw()内で色を変えてしまうと、次に描画するときは緑になりますので、
すべてが緑になってしまいます。
そこで、setup()にある ofSetColor(255,0,0); を draw()　内に移動します。

これで、赤を緑が描画されます。
他にも図形を書いたり、色を変えて試してください。
oFのドキュメントを参照。

```
ofNoFill();                    //塗りつぶししない
ofFill();                      //塗りつぶしする
ofTriangle(x1, y1, x2, y2, x3, y3);   // 三角形を描く
ofDrawBitmapString("Hello World!", 250, 200);   // ビットマップ文字を描画
```

実行した後は、赤いタスクボタンを押すか、アプリケーションの終了を選んで、
実行したアプリケーションを必ず終了してください。

プログラムの基本要素

変数

値を入れるための”もの”

箱があってそれに名前がついてる

変数の使い方

変数は使う前に宣言をする必要があります.
宣言には、変数名と変数の型を指定します.

```
int i;        // 整数型のiという変数
```

変数の型には,

int	整数型
float	浮動小数点型
char	文字型
byte	1byteの整数
boolean	論理型(true/false)

整数型には整数を浮動小数点型には実数を入れることができます.

基本的な型ではありませんが、openFrameworksで定義されている型（クラス）としては

ofColor	色
ofPoint	点
ofStyle	スタイル

などがあります.

draw() 内に以下を書いてみます。

```
--
   int i;                          //  変数の宣言　一回すればよい
   string str;                     // 文字列型の変数の宣言

   i = 10;                         //  変数へ値を代入する
   str = ofToString(i, 3);              // 数値を文字列に変換
   ofDrawBitmapString(str, 100, 50); // 文字列を(100,50)に描画

   i = 11.5;
   str = ofToString(i, 3);              // 数値を文字列に変換
   ofDrawBitmapString(str, 100, 80); // 文字列を(100,80)に描画
--
   整数に実数を代入しても、整数になってしまう。
--
   float f;

   f = 11.5;
   str = ofToString(f, 3);              // 数値を文字列に変換
```

```
   ofDrawBitmapString(str, 100, 110);// 文字列を(100,110)に描画
--
```
浮動小数点で宣言するとそのままの値が変数に入る。

```
--
```
演算の実行。

```
   i = i * 5;
   str = ofToString(i, 3);
   ofDrawBitmapString(str, 100, 130);

   i++;
   str = ofToString(i, 3);
   ofDrawBitmapString(str, 100, 150);
--
```
一度宣言した変数をまた宣言しようとするとエラーになる。

```
   int i;
--
```

命令文(statement)

命令語を組み合わせて、命令文を作る。

```
--
   ofSetColor(128, 0, 0);
   ofLine(0, 0, i, i);
   ofSetColor(192, 0, 0);
   ofLine(i, i, i+50, i+50);
   i = i + 50;
   ofSetColor(255, 0, 0);
   ofLine(i, i, i+50, i+50);
--
```
文の区切りはセミコロン。

複数の文を{ } で囲むと一つの文となる。

＊後から実例が出てきます！

変数の有効範囲（スコープ）

変数には、有効範囲がある。
一番外側で宣言すると、プログラムの全体で有効。
{ } で括られたブロック（文をまとめた文）内で宣言すると、
そのブロックの範囲内だけで有効。
このような変数の範囲のことを変数のスコープと呼びます.

```
--
```

```
   {
      int j;
      j = 20;
   }
   j = 10; //エラーになる
--
```

変数をスコープを意識しながら，自分で宣言し値を代入し出力してみよう.
同じ変数名で変数の宣言をした場合、スコープによって同じ名前だけど違うものとなっていることに注意する。

関数

ある機能を実現する手続きを名前を付けて定義したもの。

クラス

オブジェクト指向のときにでてきます。

参考：コメントの書き方

`//`：`//`以降の一行をコメントにする

`/*  */`：`/*` と `*/` の間をコメントにする。複数行も可。

制御構造

条件分岐

if　（もし、〜なら〜〜する）

```
if (条件式) 命令文;
```

```
--
   if (i > 10) i = 0;

   if (i > 10) {
      i = 0;
      j++;  // j = j + 1;
   }                          //{ } で囲むと一つの文とみなされる
--
```

if else　（もし、〜なら〜〜する、そうじゃなければ〜〜〜する）

```
   if （条件式）文1 else 文2
```

```
--
   if ( x > 200) {
      x = 0;             // xが200より大きいとき
      y = y + 10;
```

```
   } else {
      x = x + 10;        // そうじゃない(xが200以下）のとき
   }
--
```

switch case　（式の値に応じて実行する文を決める）

```
   switch(式) {
      case 値: 文１;
               break;
      case 値: 文２;
               break;
      default: 文３;
   }

--
   int num;

   switch (num) {
      case 1: ofDrawBitmapString("1desu", 100,100);
               break;
      case 2: ofDrawBitmapString("2desu", 100,100);;
              break;
      default: ofDrawBitmapString("default", 100,100);
   }
--
```

条件式

```
   i < 0                  // i より小さいとき
   i <= 0                 // i 以下のとき
   i > 0                  // i より大きいとき
   i >= 0                 // i 以上のとき
   i == j                 //  i と j が等しいとき
   i != j                 // i と j が等しくないとき
   (i < 0) && (j > 0)     // 且つ  AND
   (i < 0) || (i > 10)    // または OR
   !i                     // 否定  not
```

授業内課題１：

openFrameworks では、マウスの座標値は、mouseX, mouseY という変数に
自動的にセットされている。
マウスの座標値に応じて、色の違う点や四角を描いてみる。
まずは if を使って、右半分に描くときは赤、左半分に描くときは緑で描いてみる。
新しいプロジェクトを作成してtestApp.cppを書き換えてみましょう。

```
--
   void testApp::setup(){     // 最初の一回だけ実行される
      ofSetWindowShape(400, 400);
      ofBackground(0,0,0);
      ofHideCursor();       // カーソルが邪魔なので消す
   }

   void testApp::draw(){   // この部分はずっと繰り返し実行されている
      int x,y;             // 必要に応じて変数を宣言
      if (mouseX < ……      // この部分にif を使ってプログラム書いていく

      ofRect(mouseX, mouseY, 10.0, 10.0);
   }
--
```

注意：
プログラムを書くときは、{ があったら次からはタブでインデントするようにして
プログラムの制御構造、スコープが見ただけでわかるように書きましょう。

setup()内に
`ofSetBackgroundAuto(false);`
を加えると、画面の書き換え時に消去しなくなるので描画したものが残るようになります。

繰り返し（ループ）
for　初期値と、終了条件、繰り返しごとの処理を指定して繰り返す
for (初期化; 式がtrueの間繰り返す条件式; 繰り返し毎の処理){
　繰り返しをする文;
}

```
--
   int i;

   ofSetColor(0,0,128);
   for ( i=0; i < 10; i++){     // 最初はiを0にする、iが10より小さい間繰り返す
                                // 一回繰り返すごとにi++を実行
      ofRect(100, 35 * i, 30.0, 30.0);       // この文を10回繰り返す
   }
--
```

while　条件式を満たす間繰り返す
while (式がtrueの間繰り返す){
　繰り返しをする文;

```
  }

--
  int j = 0;
  while( j < 100){          // jの値が100より小さいときは実行する
    ofCircle(j*4, j*4, 3.0);  // 円を描く
    j++;                    // j = j+1;
  }
--
```

練習１：

（１）1から1000までを足した値を出力するプログラムをfor文を使って書く。

（２）1から100までの整数のうち、３で割り切れる値と７で割り切れる値を出力するプログラムをfor文、if文を使って書く。

ヒント： 余りを求める演算子 % 。3で割り切れ且つ７でも割り切れる値を二重に出力しないこと。文字列は文字列型+" " などで連結できます。改行文字は "\n" です。

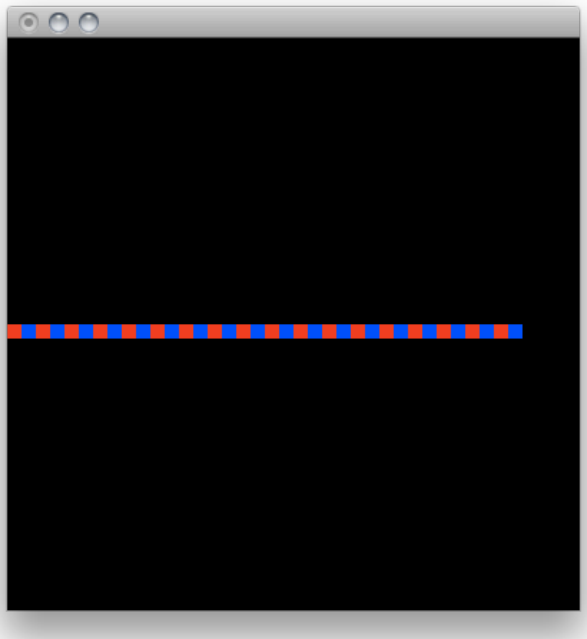
3 6 7 9 12 14 15 18 21 24 27 28 30 33 35 36 39 42 45 48 49 51 54 56 57 60 63 66 69 70 72 75 77 78 81 84 87 90 91 93 96 98 99

参考：

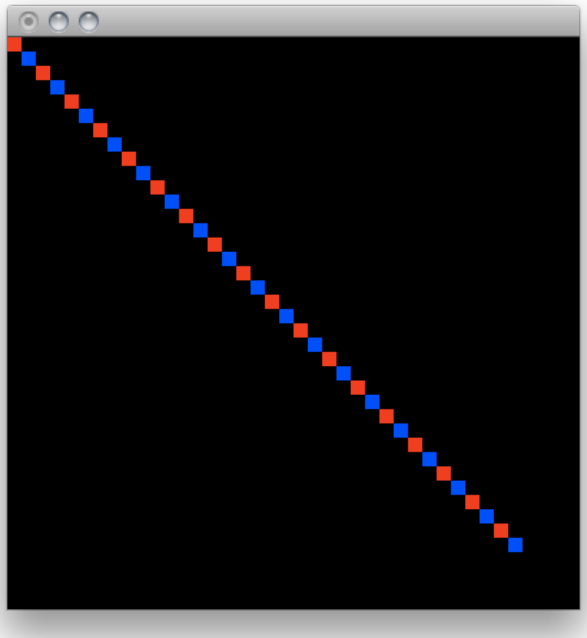
% は割った余りの値を求める演算子ですので、
10%2 は10割る２なので、余り0 となり、0を返します。
11%2 なら余りは1なので、1となります。

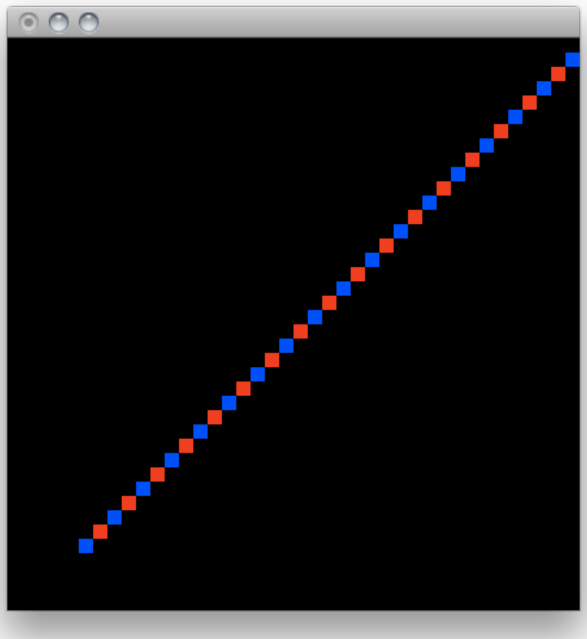
授業内課題２：

400x400のウインドウに10x10の赤と青の四角を並べて計36個描く。
for文を使って３６回のループを作り、その中で偶数/奇数をif文で判定し、
赤い四角か青い四角を描く。



授業内課題３：
さらに四角を横に並べるのではなく、斜めに並べてみる。
各方向（左上→右下、右上→左下）で試してみる。





次回までに、if 文やfor文などを使って、いろいろと描画をして、
使い方に慣れておいてください。

乱数を使って、色や座標を決めるとバリエーションがでます。
float f;
f = ofRandom(0, 100);　　// 0 から 100までの実数を返す。

関数

一連の手続き（命令文の固まり）をまとめて、名前をつけたもの。
名前は自由に決める事ができます。ただし、すでに予約されている
setup(), draw() など使うべきではありません。
関数は、値を返すことができ、返す型を宣言して定義する。
関数は、引数（ひきすう、パラメータ）を取ることができる。

関数の定義

関数の型　関数名(引数の型と変数名，引数の型と変数名)

```
int myfunction(int x, int y){
    //関数の中身
```

}
引数の数は自分で決める。上の例では２つ。
つまり、この関数では、整数型の引数を２つ与えて呼び出すと、
整数の値を返してくることになる。

関数の呼び出し

引数を与えて、呼び出すだけ。
ofBackground()やofRect()も関数なので、既に使っていることになる。
void 型は何も値を返さないので、変数を呼び出すだけで良い。

それ以外の型では、値を使う必要があるので、別の変数に代入したり、
別の関数のパラメータとして使ったりする。

```
--
   void testApp::setup(){

      ofSetWindowShape(400, 400);
      ofBackground(0,0,0);
    }

   void testApp::draw(){
      int i,j;

      myfunc();          // 関数myfuncを呼んで色を設定
      i = j = 5;
      ofRect(i, j, 10, 10);

      myfunc();
      i = myfunc2(i);    // 関数myfunc2を呼んで、計算
      ofRect(i, j, 10, 10);
   }

   void myfunc(){        // 値を返さない myfunc という名前の関数
      ofSetColor(ofRandom(0,255), 0, 0);
   }

   int myfunc2(int x){ // 整数の値を返すmyfunc2という名前の関数
      x = x * 10;      // 引数として受け取った値(xに入っている)
                       // にかけ算を行う、xはこの関数内だけで有効
      return x;        // 結果を返す
   }
--
```

練習2：

```
--
```

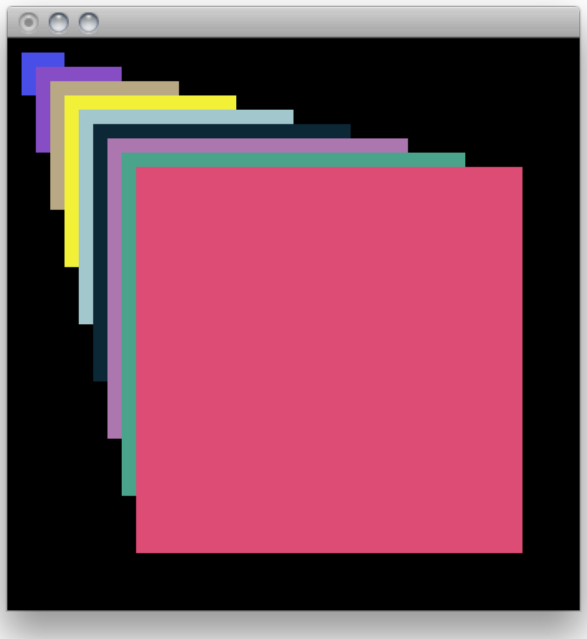
```
int sumOdd(int num){
   int i;
   完成させる
}

int sumDiv7(int num){
   完成させる
}

void testApp::draw(){

   int i;
   int result1, result2;
   string str;

   i = ofGetFrameNum();
   result1 = sumOdd(i);
   result2 = sumDiv7(i);
   str = ofToString(result1, 0) + " " + ofToString(result2, 0);
   ofDrawBitmapString(str, 100, 100);
   ofRect(result1 % 400, result2 % 400, 10.0, 10.0);
}
```

--

（1）引数として与えられた値までの、奇数の値の合計を計算する関数sumOddを
　　完成させる。
（2）引数として与えられた値までの、７で割り切れる値の合計を計算する
　　関数sumDiv7 を完成させる。

授業内課題4：

「色をランダムに設定し、四角を描く」部分を関数としてまとめる。
引数として、四角の始点の座標と終点の座標を与える。
返す値は特にないので、void 型とする。
つまり、
void mykansu(int sx, int sy, int ex, int ey)
の関数を完成させる。
この関数を使うプログラムとして、段々と大きくなりながら
ランダムの色の四角を描く。



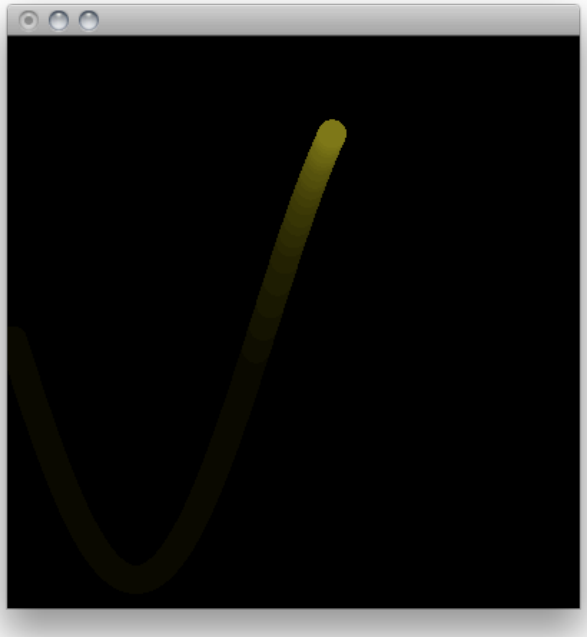
アニメーション

openFrameworksでは、最初からアニメーションするようにプログラムが
構成されています。
setup() 関数は、プログラムの開始時に一回だけ行われるため、
全体の初期化などの準備処理を行います。
update() および draw() 関数は、ofSetFramerate()で設定したレートで自動的に
繰り返し呼び出されます。
この関数の中身を記述することで、アニメーションを描画できます。

```
int x = 0;
int y = 0;

void testApp::setup(){
   ofSetWindowShape(400, 400);
   ofBackground(0,0,0);
   ofEnableAlphaBlending();   // アルファを有効にする
   ofSetFrameRate(30);
}

void testApp::draw(){

   ofSetColor(0, 0, 0, 10);         // アルファを指定して塗りつぶす
```

```
    ofRect(0, 0, ofGetWidth(), ofGetHeight());

    ofSetColor(128,128,0);
    ofRect(x, y, 10, 10);
    x++;
    y++;
}
```

授業内課題5：
三角関数 sin() を使って、上下に動きながら移動する円を描いてみる。
三角関数を扱うには、値をfloat（浮動小数点）で使うため、
座標値もfloat で宣言してしまった方が良いかも。



マウスイベント

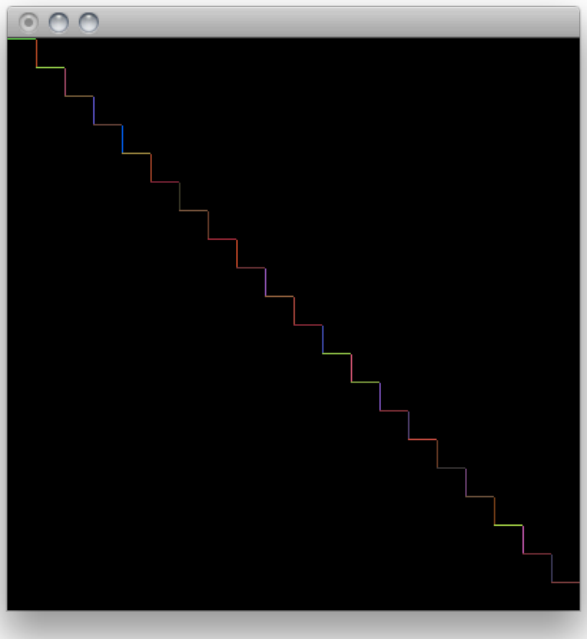
マウスの座標値は、mouseX, mouseYに自動的に格納される。
マウスのクリックの検出は、用意されている関数によって行う。

```
void testApp::mousePressed() {     // マウスをクリックすると
                                   // おこる事を記述する
}
```

マウスがクリックされると、mousePressed関数が呼ばれるように
なっている。
プログラムとしては、マウスのイベントを常に監視していて、
イベントが発生すると、関数を呼ぶようになっている。

グラフィカルなインターフェイスを持つプログラムでは、
このようなイベント駆動（event driven）によるプログラムが
一般的に使われている。

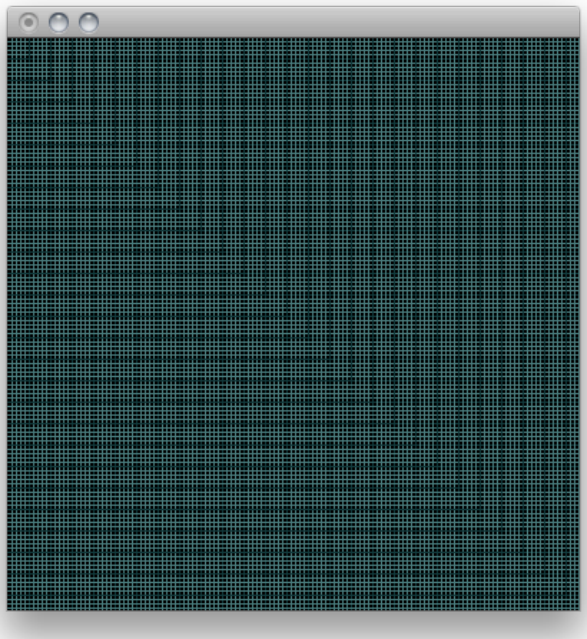
課題１：
条件判断(if)とループを使って、階段状に線を描いていく。
階段の各々の段に別の色を付けてみる。



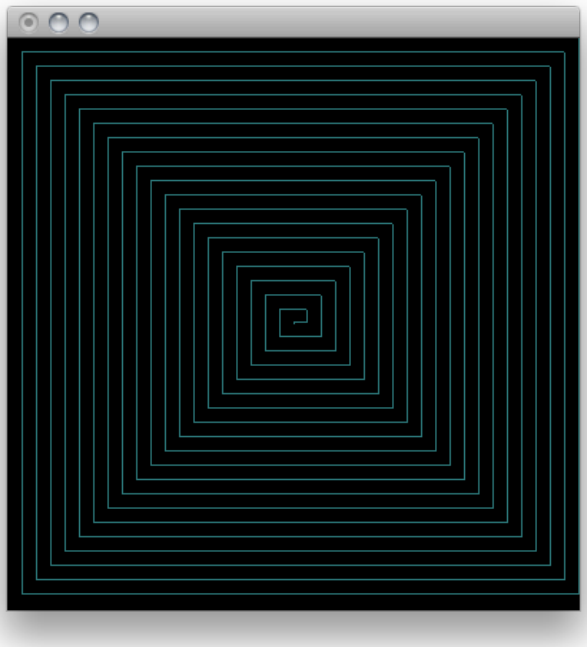
どうやって描いていけば良いか考えてみる。
描いていく順番をイメージし、書き出してみる。
描くための手順を繰り返しや条件を意識しながら書き出してみる。
プログラムとして使えるもの(ifやfor)によって書き直す。
プログラムを書いて試してみる。エラーが出たら直す。
試行錯誤する。

このようにプログラムとして実現可能な手順を見つけることが、
アルゴリズムを考えること。

課題２：
400x400のキャンバスを作成し、その上に
5本毎に違う色を付けた線を引く事で、グリッドを描画する。



課題３：
渦巻き状の線を描く。
線毎に色を変えていく。
width , height という変数には、ウインドウの幅と高さが
自動的にセットされています。
これを使うと、ウインドウの大きさを変えても
大きさに応じて描画できるようになります。
ヒント：while文、絵を描いて何が変化しないといけないか確認する。
　　各線で視点と終点の座標を式で表してみる。



課題４：
マウスポインタの周りを円を描くようにグルグル回る円のアニメーションを作成する。
マウスをクリックすると、違う色の円が回り始める。

ofSetVerticalSync(true);
をsetup()内に入れるとアニメーションが落ち着く。