

プログラミング基礎 #11

反復と配列2

担当：向井 智彦

第5回のおさらい

- for (初期化式; 継続条件式; 更新式) { }
- while (継続条件式) { }
- 反復の中断には break
- 反復対象ブロック内の後略には continue
- 変数のスコープは宣言した中括弧 { } 内
- スコープに対応したインデントで整形を
 - ついでに単語・演算子の前後の空白も

演算子の書き換え

算術演算＋代入

- $a = a + 5;$
- $a = a - 2;$
- $a = 6 * a;$
- $a = a / 4;$
- インクリメント
- $a = a + 1; a += 1;$
- デクリメント
- $a = a - 1; a -= 1;$

複合代入

- $a += 5;$
- $a -= 2;$
- $a *= 6;$
- $a /= 4;$
- インクリメント
- $++a;$ か $a++;$
- デクリメント
- $--a;$ か $a--;$

while 文

```
#include <iostream>
```

```
int main() {
```

```
    int x = 0;
```

```
    while (x >= 0)
```

```
    {
```

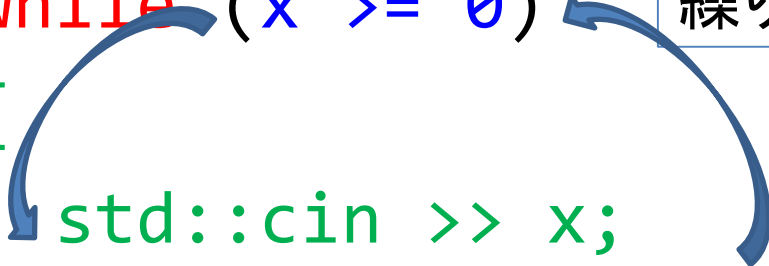
```
        std::cin >> x;
```

```
        std::cout << x << std::endl;
```

```
    }
```

```
}
```

青字の条件式(入力が正の数)を満たす限り、緑字のブロックを繰り返し実行



「N回繰り返す」をwhile文で

```
#include <iostream>
```

```
int main() {
```

```
    int x = 0; //カウント0スタート
```

```
    while (x < 10)
```

```
    {
```

```
        std::cout << x << std::endl;
```

```
        ++x; //カウントを1つずつ増やす
```

```
    }
```

```
}
```

青字の条件式(カウント10未満)を満たす限り、緑字のブロックを繰り返し実行

条件式の更新

```
#include <iostream>
int main() {
    int x = 0;
    while (x >= 0) {
        std::cin >> x;
        std::cout << x <<
            std::endl;
    }
```

} ポイント:

```
#include <iostream>
int main() {
    int x = 0;
    while (x < 10) {
        std::cout << x <<
            std::endl;
        x += 1;
    }
```

}

条件式に登場する変数が、反復のたびに更新される

「n回だけ繰り返す」をfor文で

```
#include <iostream>
int main()
{
    for (int x = 0; x < 10; ++x)
    {
        std::cout << x << std::endl;
    }
}
```

for文の動作は反時計回りに読む

```
#include <iostream>
int main()
{
    for (int i = 0; i < 10; ++i)
    {
        std::cout << i << std::endl;
    }
}
```

① ② ③ ④

for文の動作図解

```
#include <iostream>
int main()
{
    ①初期設定
    for (int i = 0; i < 10; ++i)
    {
        std::cout << i << std::endl;
    }
}
```

for文の動作図解

```
#include <iostream>
int main()
{
    for (int i = 0; i < 10; ++i)
    {
        std::cout << i << std::endl;
    }
}
```

①反復対象処理の実行

for文の動作図解

```
#include <iostream>
```

```
int main()
```

```
{
```

```
    for (int i = 0; i < 10; ++i)
```

```
    {
```

```
        std::cout << i << std::endl;
```

```
    }
```

```
}
```

②次の反復に
移る際に
実行される式

for文の動作図解

```
#include <iostream>
int main()
{
    for (int i = 0; i < 10; ++i)
    {
        std::cout << i << std::endl;
    }
}
```

③反復の継続判定

for文の動作図解

```
#include <iostream>
int main()
{
    for (int i = 0; i < 10; ++i)
    {
        std::cout << i << std::endl;
    }
}
```

①反復対象処理の実行 に戻る

for文の動作は反時計回りに読む

```
#include <iostream>
int main()
{
    for (int i = 0; i < 10; ++i)
    {
        std::cout << i << std::endl;
    }
}
```

The diagram illustrates the execution flow of the for loop. Four numbered boxes (①, ②, ③, ④) are placed around the loop structure, connected by blue arrows indicating the sequence of execution. The flow starts at box ① (initialization), moves to box ② (condition check), then to box ③ (loop body), and finally to box ④ (increment), which loops back to box ②.

配列と反復：配列各要素の処理

```
#include <iostream>
int main(void) {
    double a[10], b[10];
    for (int i = 0; i < 10; ++i)
    {
        a[i] = i + 0.5;
        b[i] = a[i] * 2.0;
        std::cout << b[i] << std::endl;
    }
}
```

多重ループ

```
#include <iostream>
int main(void)
{
    for (int x = 0; x < 5; ++x)
    {
        for (int y = 0; y < 5; ++y)
        {
            std::cout << x + y << std::endl;
        }
        std::cout << x << std::endl;
    }
}
```

xが0の状態で yが0～4まで反復し、
次にxが1の状態でyが0～4まで(略)
次にxが2の状態でyが(略)
最後にxが4の状態でyが(略)

多次元配列 (NEW!)

```
#include <iostream>
using namespace std;
int main() {
    int x[3][4]; //2次元、3×4、12要素
    x[0][0] = 0;
    x[0][1] = 1;
    x[0][2] = 2;
    x[0][3] = 3;
    ...
    x[2][3] = 10;
    x[2][3] = 11;
}
```

x[0][0]	x[0][1]	x[0][2]	x[0][3]
x[1][0]	x[1][1]	x[1][2]	x[1][3]
x[2][0]	x[2][1]	x[2][2]	x[2][3]

多重ループと多次元配列

```
#include <iostream>
int main(void)
{
    double data[5][4]; // 2次元、5×4, 20要素
    for (int i=0; i<5; ++i)
    {
        for (int j=0; j<4; ++j)
        {
            data[i][j] = i*10+j;
        }
    }
}
```

0	1	2	3
10	11	12	13
20	21	22	23
30	31	32	33
40	41	42	43

反復のグラフィックス応用

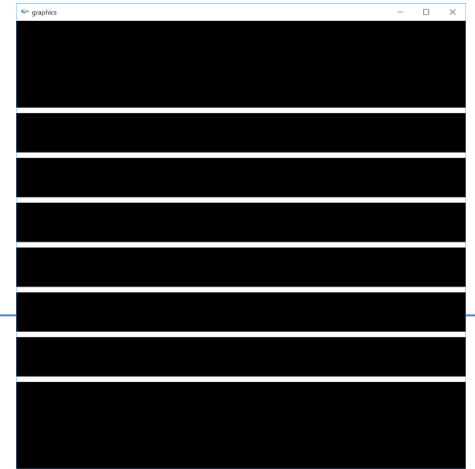
複数線の表示

```
glBegin(GL_LINES);  
glVertex3d(-100.0, -60, 0); glVertex3d(100.0, -60, 0);  
glVertex3d(-100.0, -40, 0); glVertex3d(100.0, -40, 0);  
glVertex3d(-100.0, -20, 0); glVertex3d(100.0, -20, 0);  
glVertex3d(-100.0, 0, 0); glVertex3d(100.0, 0, 0);  
glVertex3d(-100.0, 20, 0); glVertex3d(100.0, 20, 0);  
glVertex3d(-100.0, 40, 0); glVertex3d(100.0, 40, 0);  
glVertex3d(-100.0, 60, 0); glVertex3d(100.0, 60, 0);  
glEnd();
```

$y = 60$

$y = 0$

$y = -60$



複数線の表示 - 反復版1

```
glBegin(GL_LINES);  
glVertex3d(-100.0, -60, 0); glVertex3d(100.0, -60, 0);  
glVertex3d(-100.0, -40, 0); glVertex3d(100.0, -40, 0);  
glVertex3d(-100.0, -20, 0); glVertex3d(100.0, -20, 0);  
glVertex3d(-100.0, 0, 0); glVertex3d(100.0, 0, 0);  
glVertex3d(-100.0, 20, 0); glVertex3d(100.0, 20, 0);  
glVertex3d(-100.0, 40, 0); glVertex3d(100.0, 40, 0);  
glVertex3d(-100.0, 60, 0); glVertex3d(100.0, 60, 0);  
glEnd();
```

```
glBegin(GL_LINES);  
for (int i = -60; i <= 60; i = i + 20) {  
    glVertex3d(-100.0, i, 0); glVertex3d(100.0, i, 0);  
}  
glEnd();
```

-60～60まで20刻みで

複数線の表示 - 反復版2

```
glBegin(GL_LINES);  
glVertex3d(-100.0, 20 * -3, 0); glVertex3d(100.0, 20 * -3, 0);  
glVertex3d(-100.0, 20 * -2, 0); glVertex3d(100.0, 20 * -2, 0);  
glVertex3d(-100.0, 20 * -1, 0); glVertex3d(100.0, 20 * -1, 0);  
glVertex3d(-100.0, 20 * 0, 0); glVertex3d(100.0, 20 * 0, 0);  
glVertex3d(-100.0, 20 * 1, 0); glVertex3d(100.0, 20 * 1, 0);  
glVertex3d(-100.0, 20 * 2, 0); glVertex3d(100.0, 20 * 2, 0);  
glVertex3d(-100.0, 20 * 3, 0); glVertex3d(100.0, 20 * 3, 0);  
glEnd();
```

```
glBegin(GL_LINES);  
for (int i = -3; i <= 3; ++i) {  
    glVertex3d(-100.0, 20 * i, 0); glVertex3d(100.0, 20 * i, 0);  
}  
glEnd();
```

-60～60を20の倍数で表現

複数線の表示 - 反復版3

```
glBegin(GL_LINES);  
glVertex3d(-100.0, -60 + 0, 0); glVertex3d(100.0, -60 + 0, 0);  
glVertex3d(-100.0, -60 + 20, 0); glVertex3d(100.0, -60 + 20, 0);  
glVertex3d(-100.0, -60 + 40, 0); glVertex3d(100.0, -60 + 40, 0);  
glVertex3d(-100.0, -60 + 60, 0); glVertex3d(100.0, -60 + 60, 0);  
glVertex3d(-100.0, -60 + 80, 0); glVertex3d(100.0, -60 + 80, 0);  
glVertex3d(-100.0, -60 + 100, 0); glVertex3d(100.0, -60 + 100, 0);  
glVertex3d(-100.0, -60 + 120, 0); glVertex3d(100.0, -60 + 120, 0);  
glEnd();
```

```
glBegin(GL_LINES);  
for (int i = 0; i <= 120; i += 20) {  
    glVertex3d(-100.0, -60 + i, 0);  
    glVertex3d(100.0, -60 + i, 0);  
}  
glEnd();
```

–60から20刻みで増やす

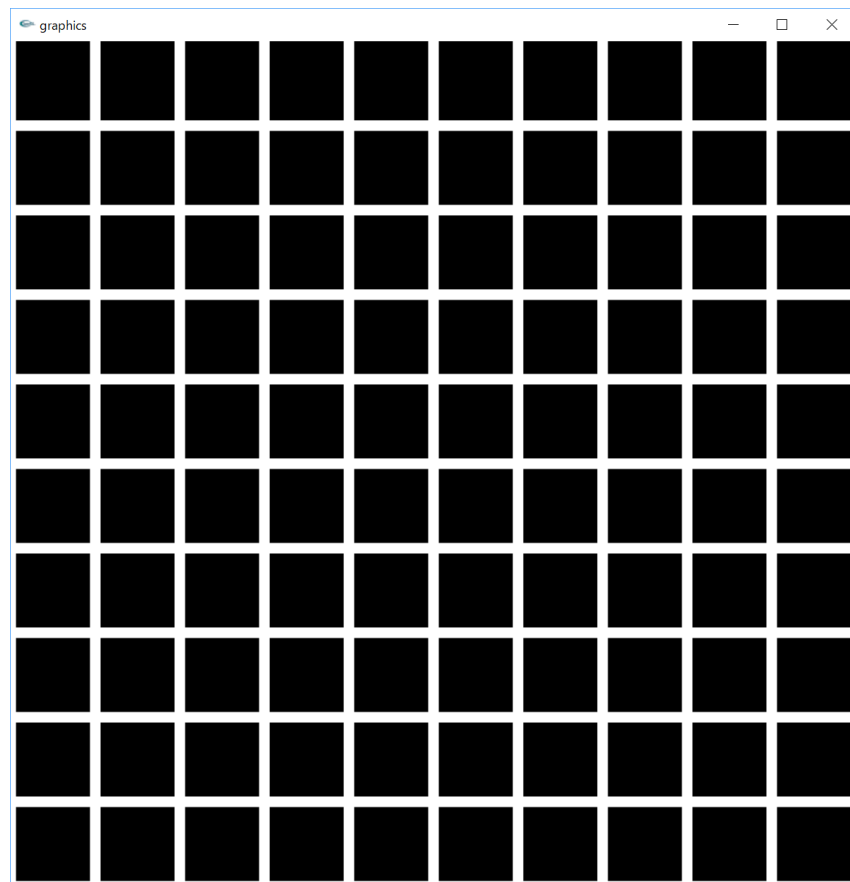
複数線の表示 - 反復版4

```
glBegin(GL_LINES);
glVertex3d(-100.0, -60 + 0, 0); glVertex3d(100.0, -60 + 0, 0);
glVertex3d(-100.0, -60 + 20, 0); glVertex3d(100.0, -60 + 20, 0);
glVertex3d(-100.0, -60 + 40, 0); glVertex3d(100.0, -60 + 40, 0);
glVertex3d(-100.0, -60 + 60, 0); glVertex3d(100.0, -60 + 60, 0);
glVertex3d(-100.0, -60 + 80, 0); glVertex3d(100.0, -60 + 80, 0);
glVertex3d(-100.0, -60 + 100, 0); glVertex3d(100.0, -60 + 100, 0);
glVertex3d(-100.0, -60 + 120, 0); glVertex3d(100.0, -60 + 120, 0);
glEnd();
```

```
glBegin(GL_LINES);                                -60から20刻みで増やす(倍数表現)
for (int i = 0; i < 7; ++i) {
    glVertex3d(-100.0, -60 + i * 20, 0);
    glVertex3d( 100.0, -60 + i * 20, 0);
}
glEnd();
```

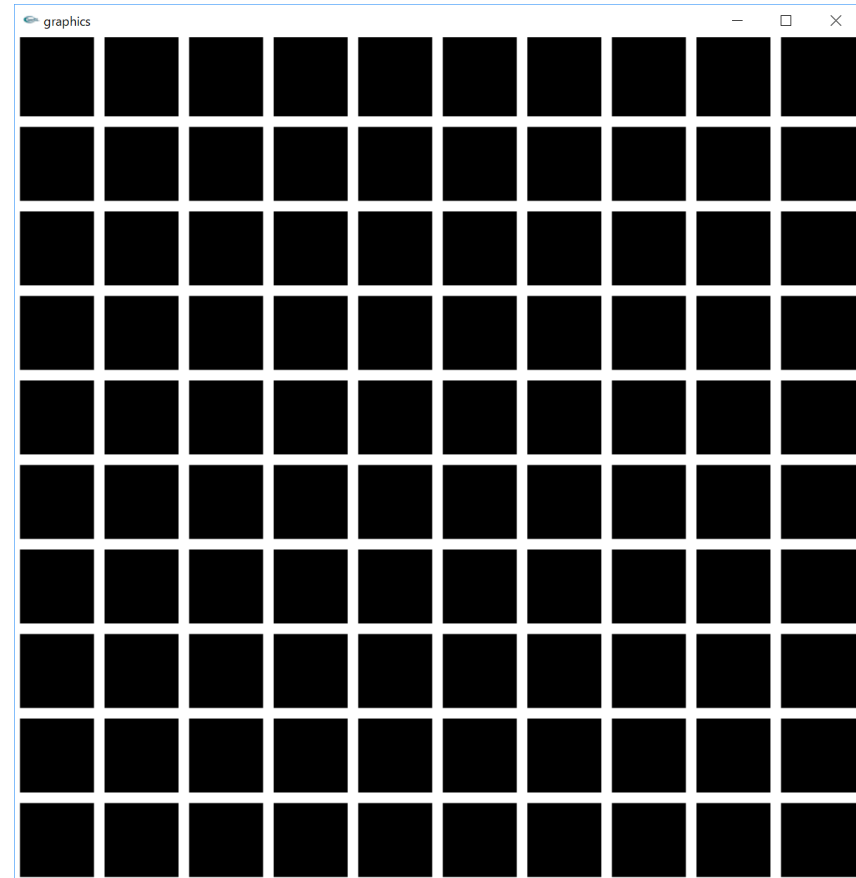

演習課題BASIC

- 右図のような格子模様を完成させる
 - 縦線 × 9本
 - 横線 × 9本
 - 線の太さや色は自由



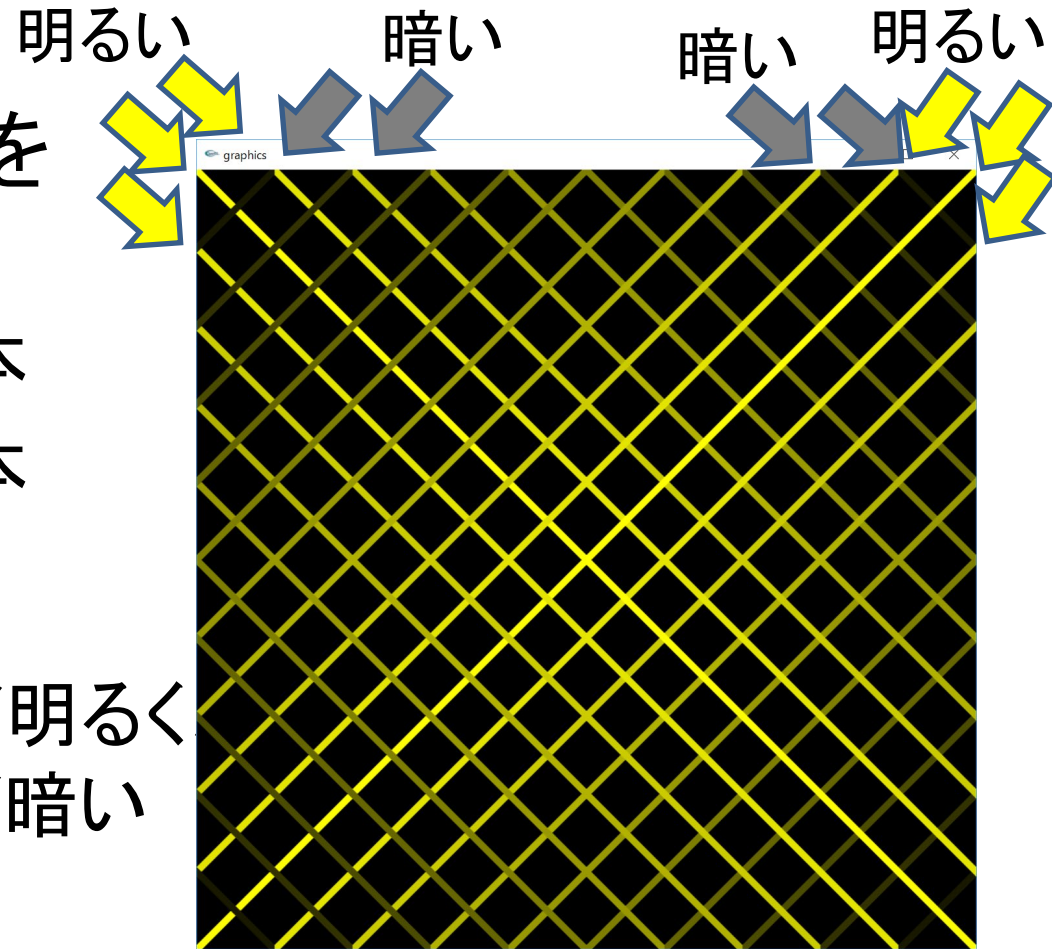
演習課題 EXTRA-0

- BASIC課題を見栄え良くアレンジ
 - 面白いアレンジを施したBASIC課題は、EXTRAに格上げして評価



演習課題 EXTRA-1

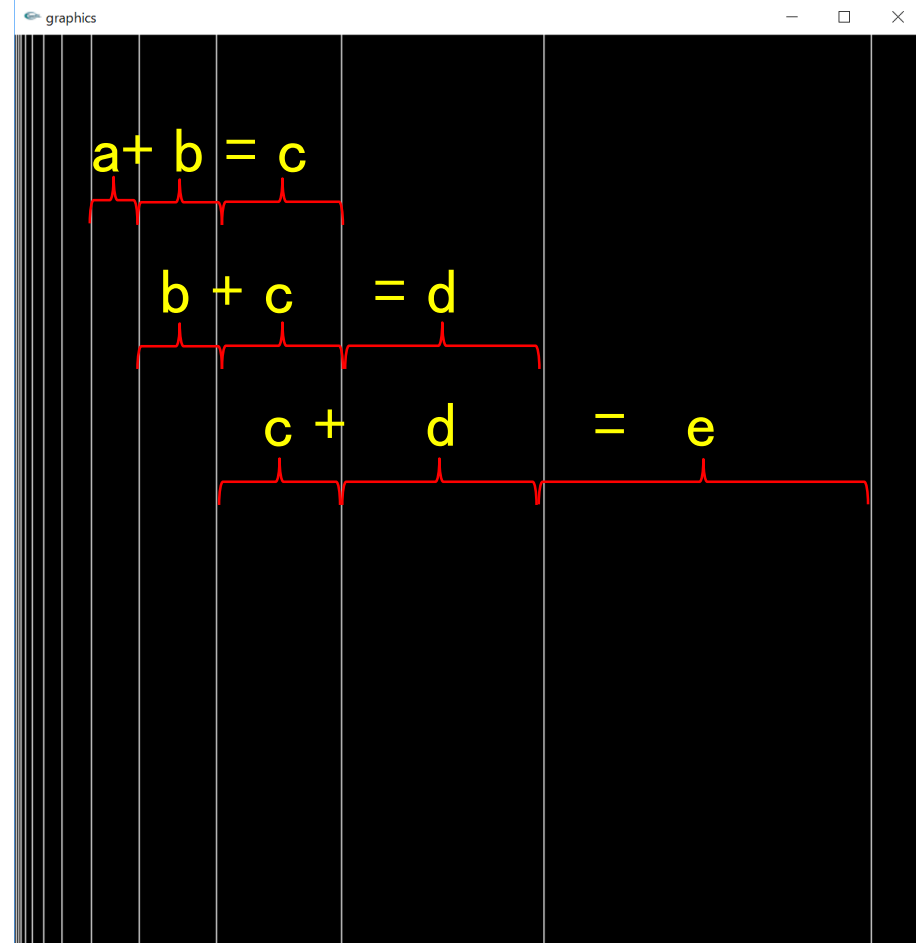
- 右図のような模様を完成させる
 - 右下がり線 × 19本
 - 右上がり線 × 19本
 - 線の太さは自由
 - 対角線に近いほど明るく
画面隅に近いほど暗い
色をつける
 - 多少アレンジしてもOK



演習課題 EXTRA-2

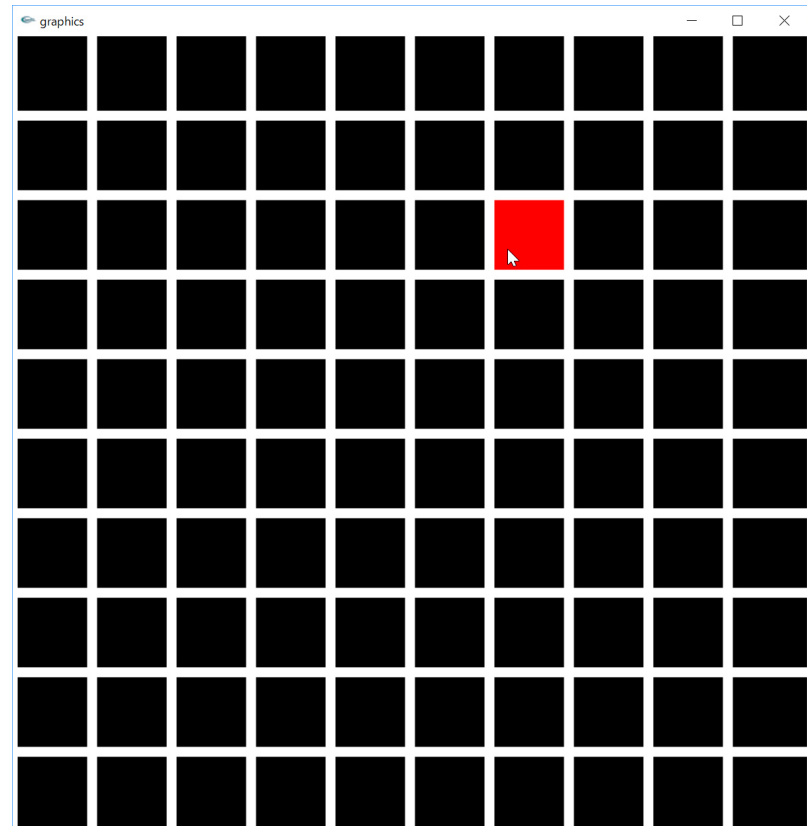
- 右図のような模様を完成させる
 - フィボナッチ数列
 - 線の下図や太さ、色は自由
 - 多少アレンジしてもOK

※右側に進むほど黄金比に近づく



演習課題 ADVANCE

- BASIC課題を拡張し、マウスカーソルが位置するマスのみ色を変えて表示するプログラムを作成



<https://youtu.be/qkAkvs94BcQ>