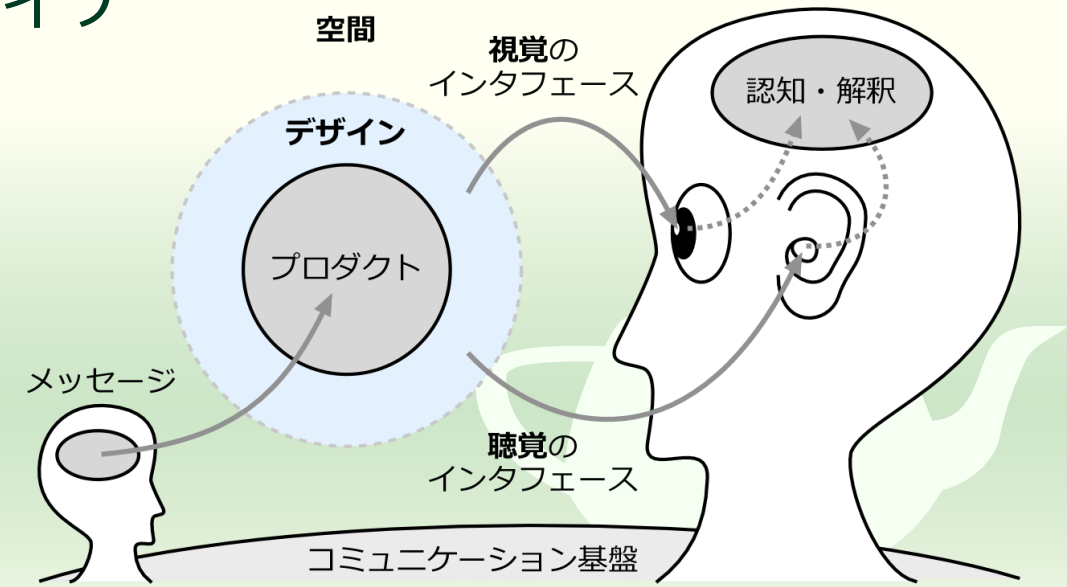
メディアプログラミング演習

第1回

本演習の目的

- プログラミングで**メディア**を取り扱う
  - メディアとは
    - media（メディア）は medium（メディウム）の複数形
    - すなわち中間にあるもの・間に入って媒介するものたち
  - コミュニケーションにおけるメディア
    - 音，音声，音楽
    - 画像，映像
    - デザイン
  - 人と人の**間**にあるもの



プログラミングとは

あなたがすべきことがどこかに書いてあるわけではない

プログラムするということ・序

- 課題を解決する手順や方法を考える
  - つまり**アルゴリズム**を考える
    - 問題を解く手順を定式化したもの
    - 算法ともいう
- コンピュータが取り扱い可能な形式で記述する
  - つまり**ソースプログラム**を作成する
    - アルゴリズムをもとに問題を解く手順を記述したもの
    - 原始プログラム，ソースコードともいう
    - かつては算譜という和製漢語が用いられていたが廃れた

プログラムするということ・破

- 実行したプログラムが正しく動作するか確かめる
  - つまり**テスト**する
    - プログラムの機能が期待通り動作するかどうか
    - プログラムの性能が期待通り得られるかどうか
    - プログラムが悪意のある操作によって予想外の動作をしないか
- プログラムが正しく動作するよう修正する
  - つまり**デバッグ**する
    - プログラムが正しく動作しない原因や理由を見つける
    - プログラムを正しく動作させる修正方法を考える

プログラムするということ・急

- まず課題を分析して正しい設計をする
- それでも書いたプログラムはたいてい正しく動かない
- 正しく動かない理由もたいてい分からない
- だから正しく動くようになるまで**試行錯誤**する

試行錯誤できないとプログラミングできない

プログラミングの学習

- 教科書を読んだだけでは書けるようにならない
  - 文法を学んでも目的を達成する方法は分からない
  - プログラムは書かないと書けるようにならない
- プログラミングが苦手という人のパラドックス
  - 書けないのに書くことはできないと思う
  - 最初に何をしたらいいのか分からない
- まず**何のため**（目的）に**何をする**（処理）のか**決める**
  - プログラミングは**処理の手順**（手続き）を考えること

「プログラミング的思考」

問題の発見

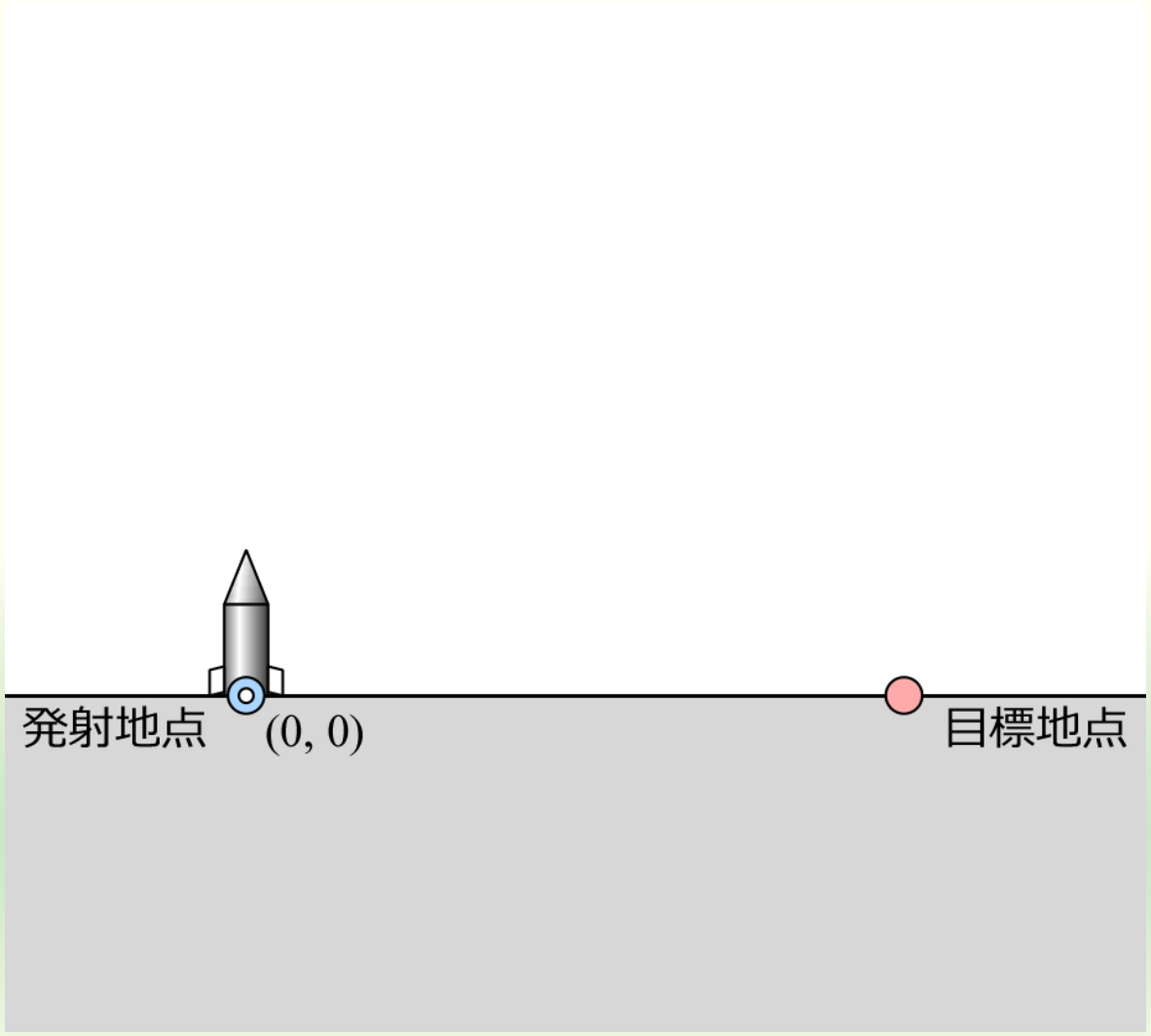
分解，抽象化・
一般化，定式化

解法の考案，
コーディング

テスト，実験，
運用

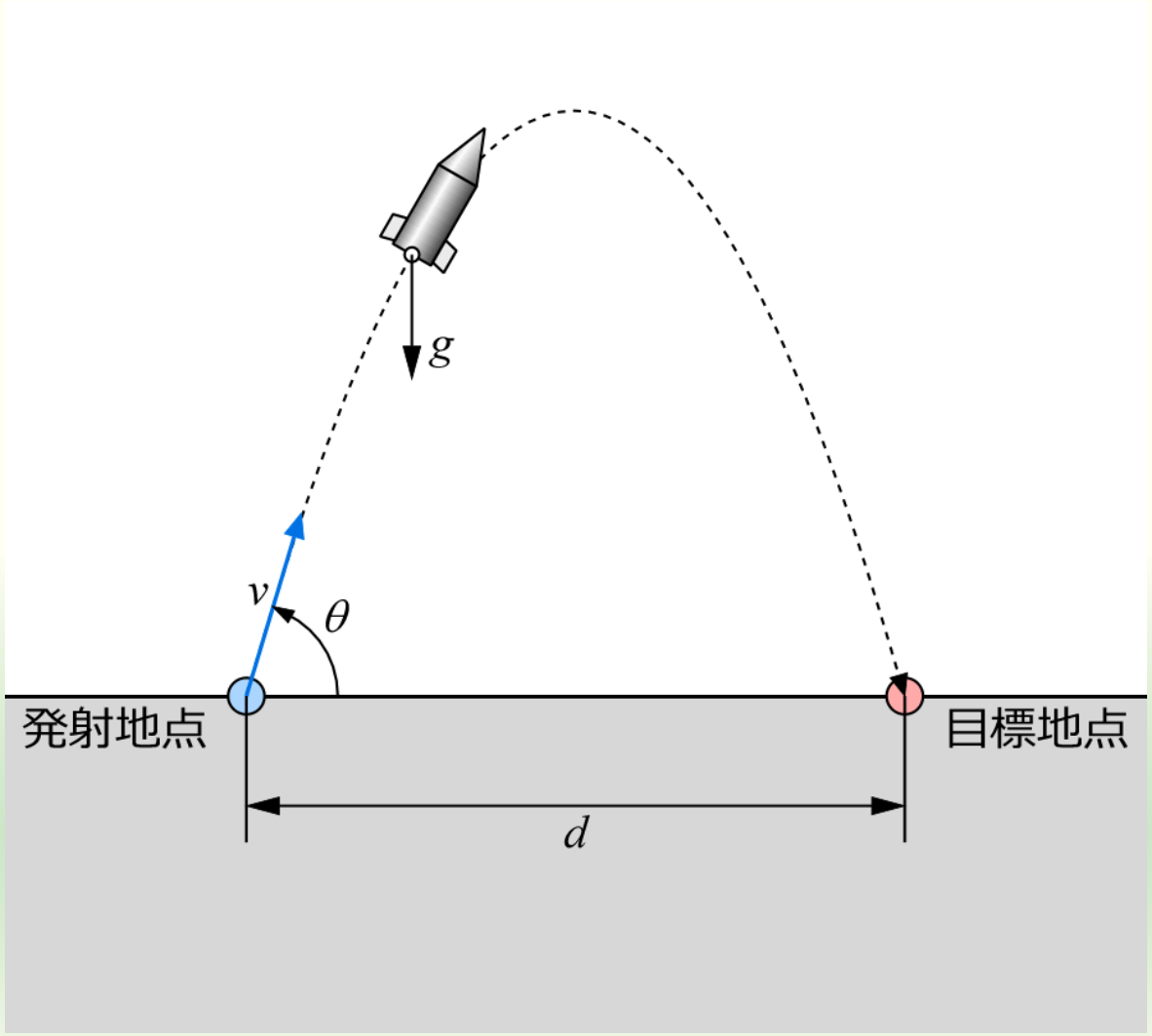
課題の例

- 弾道ミサイルを目標地点に到達させたい
- 考えること
  - 既知のことは何か
  - 未知なことは何か
  - 求めることは何か



分析

- 弾道ミサイルの速度は v
- 目標地点までの距離は d
- 重力加速度は g
- 条件
  - 空気抵抗は考慮しない
  - 自分で速度を変えない
  - 自分で向きを変えない
- 発射角 θ を求める

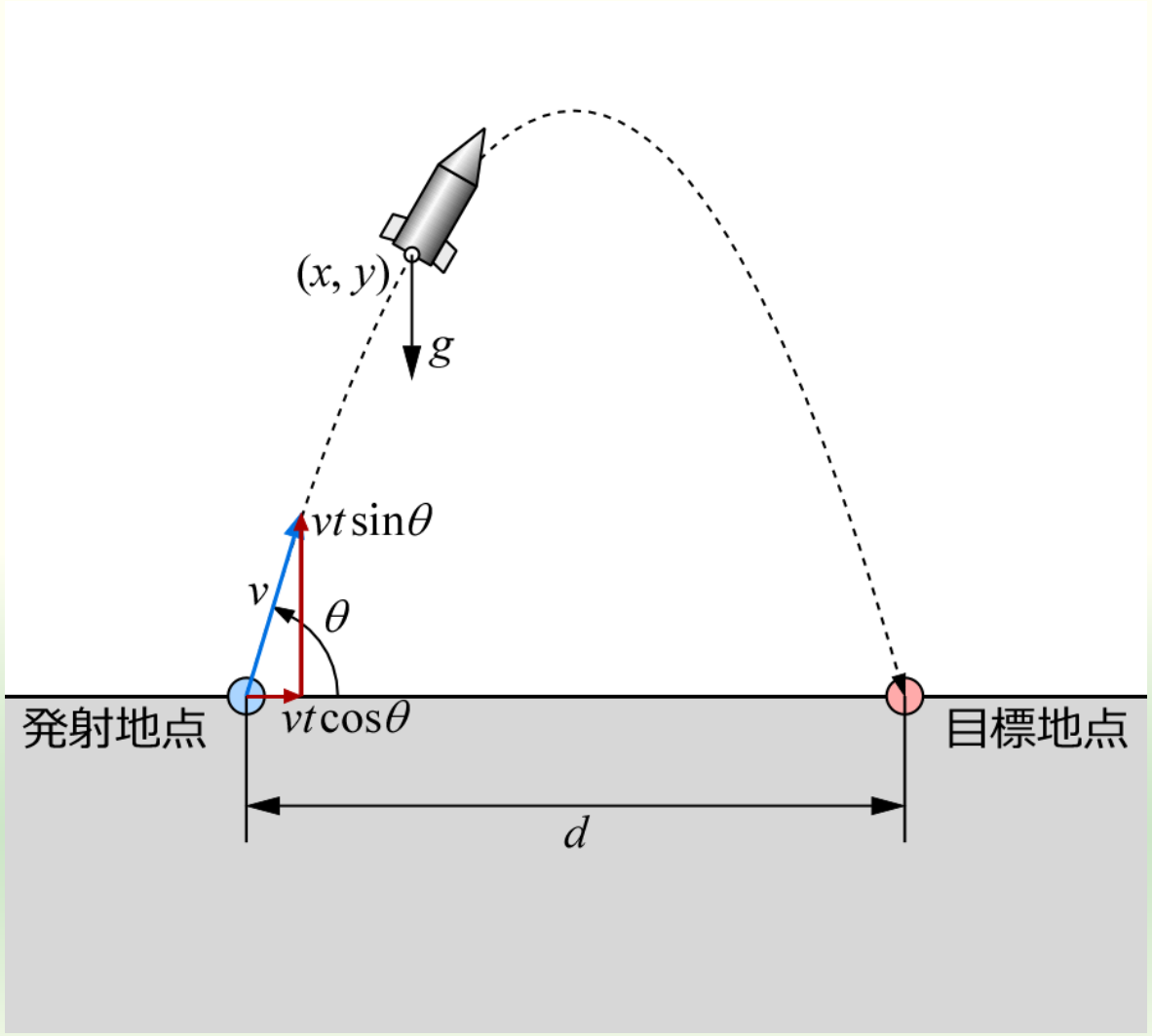


モデル化

- 弾道ミサイルの軌跡
  - 軌跡は放物線を描く
  - 時刻を t とする

$$x = vt\cos\theta$$

$$y = vt\sin\theta - \frac{1}{2}gt^2$$

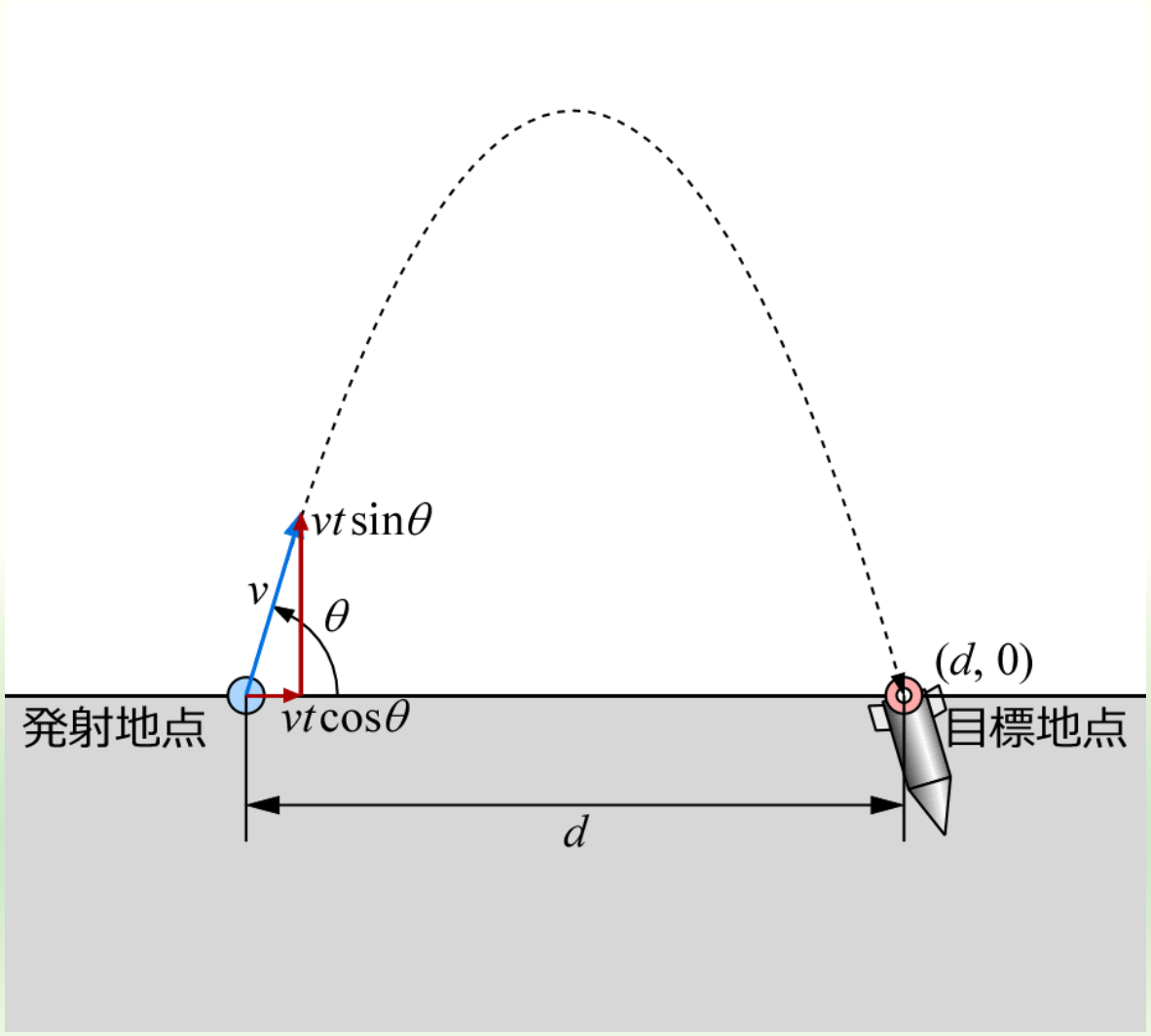


実装（解法の考案）

- 目標地点に到達したとき

$$x = vt\cos\theta = d$$

$$y = vt\sin\theta - \frac{1}{2}gt^2 = 0$$



実装（解法の考案）

- x の式より

$$t = \frac{d}{v\cos\theta}$$

- y の式の t に代入して

$$\frac{d\sin\theta}{\cos\theta} - \frac{gd^2}{2v^2\cos^2\theta} = 0$$

- $(2\cos^2\theta)/d$ 倍して移項

$$2\sin\theta\cos\theta = \frac{gd}{v^2}$$

- 二倍角の公式より

$$\sin 2\theta = \frac{gd}{v^2}$$

$$\theta = \frac{1}{2}\sin^{-1}\frac{gd}{v^2}$$

これをプログラムで計算するということを**決めて**ようやくコーディング

実装（コーディング）

ソースプログラム

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
  // 重力, 速度, 距離
  double gravity, velocity, distance;

  // パラメータの入力
  cout << "重力加速度:";
  cin >> gravity;
  cout << "発射速度:";
  cin >> velocity;
  cout << "目標までの距離:";
  cin >> distance;

  // パラメータのチェック
  if (gravity <= 0.0) {
    cout << "重力が小さすぎます¥n";
    return 1;
  }
  if (velocity <= 0.0) {
    cout << "速度が小さすぎます¥n";
    return 1;
  }
  if (distance <= 0.0) {
    cout << "距離が近すぎます¥n";
    return 1;
  }
```

解法のコードはこれだけ

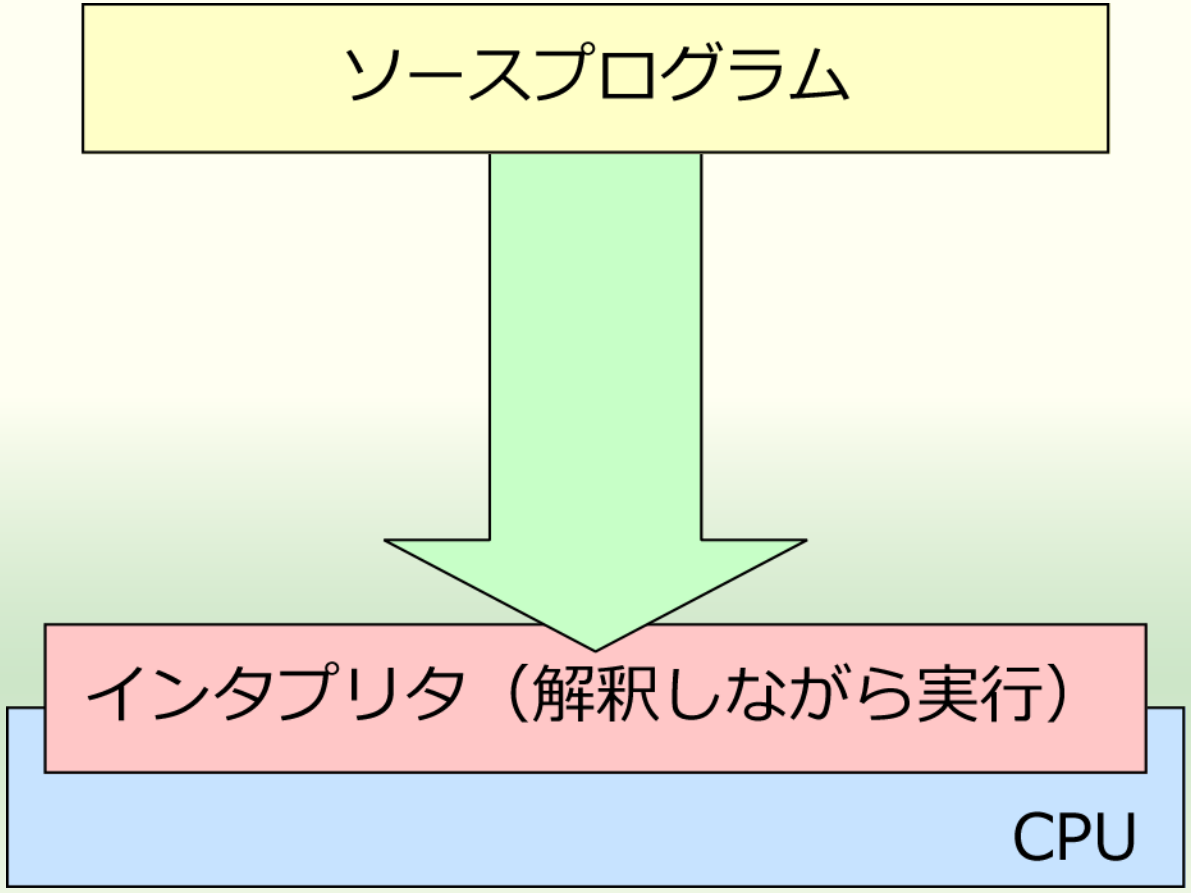
```
  double s = gravity * distance;
  double t = velocity * velocity;

  if (s > t) {
    cout << "目標に届きません¥n";
    return 1;
  }

  // 結果の出力
  cout << "発射角度:" << asin(s / t) * 0.5 << "¥n";
}
```

プログラミング言語

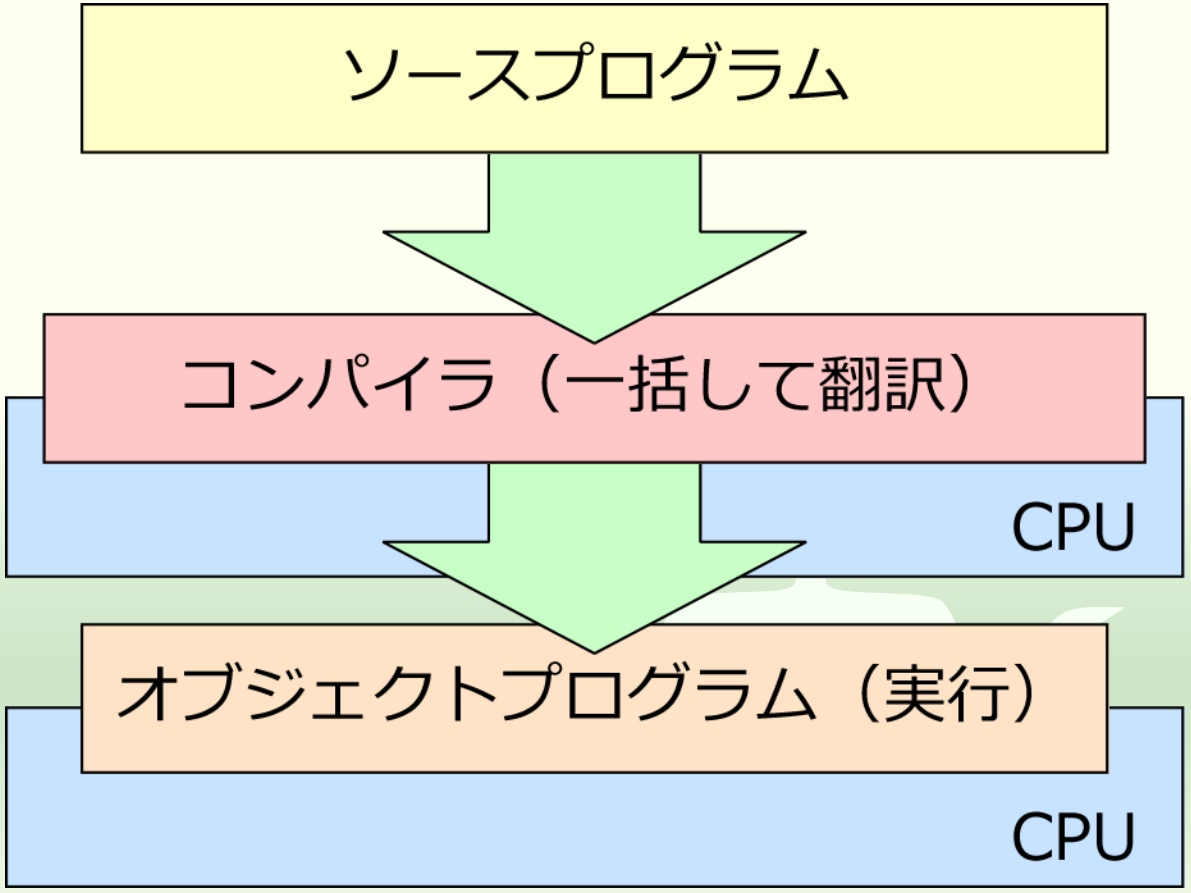
- ソースプログラムの記述に用いる言語
  - 言語なので**文法**が決められている
  - 文法に違反した記述は**エラー**になる
- ソースプログラムは人間が記述する
  - コンピュータが理解可能な**機械語**への**翻訳**が必要
    - 逐次的に解釈しながら実行
      - インタプリタ方式
    - 一括して翻訳した後に実行
      - コンパイラ方式

インタプリタ方式とコンパイラ方式

インタプリタ方式



コンパイラ方式



ソースプログラムのコンパイル

C++ 言語のソースプログラム

```
//
// 関数 add の定義
//
int add(int x, int y)
{
  // 変数 z の宣言
  int z;

  // 引数 x と引数 y を足して z に代入する
  z = x + y;

  // z を関数の戻り値（関数の値）とする
  return z;
}
```

コンパイル
（一括翻訳）

アセンブリ言語（機械語に対応）

```
add:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $16, %rsp
        movl    %ecx, 16(%rbp)
        movl    %edx, 24(%rbp)
        movl    16(%rbp), %edx
        movl    24(%rbp), %eax
        addl    %edx, %eax
        movl    %eax, -4(%rbp)
        movl    -4(%rbp), %eax
        addq    $16, %rsp
        popq    %rbp
        ret
```

アセンブリ言語は数値の羅列である機械語の命令を文字による簡略表記（ニーモニック）で表したもの

インタプリタ方式のプログラミング言語

- Python, Ruby, Perl, JavaScript, PHP, Prolog, …
  - 軽量言語 (Light Weight Language, LL)，スクリプト言語
  - Web や機械学習などサービス開発で利用されるものが多い
- 特徴
  - CPU によって実行されるのはインタプリタ
  - インタプリタがソースプログラムを解釈しながら実行する
  - コンパイル（翻訳）の必要がない

コンパイラ方式のプログラミング言語

- C, C++, Objective-C, Swift, Go, Fortran, COBOL, …
  - OS やアプリケーションの開発に使用されるものが多い
  - プログラミング言語自体の開発にも使用される
- 特徴
  - CPU は機械語のオブジェクトプログラムを直接実行する
  - 実行前にあらかじめコンパイル（翻訳）する必要がある
  - 実行時にソースプログラムの解釈を行わないため高速

中間言語方式のプログラミング言語

- C#, Java, Kotlin, Scala, Pascal, Smalltalk, …
  - 近年のアプリケーション開発に用いられることが多い
- 特徴
  - コンパイラを使って中間言語に翻訳
  - 中間言語のプログラムをインタプリタによって実行する
  - インタプリタ方式とコンパイラ方式の中間的な特徴をもつ
  - コンパイルが速く実行速度もコンパイル方式に迫る
  - 中間言語のインタプリタが用意された多様な環境で使える

JIT (Just In Time) コンパイラにより解釈しながら機械語への変換を行うものが主流

プログラミングパラダイム

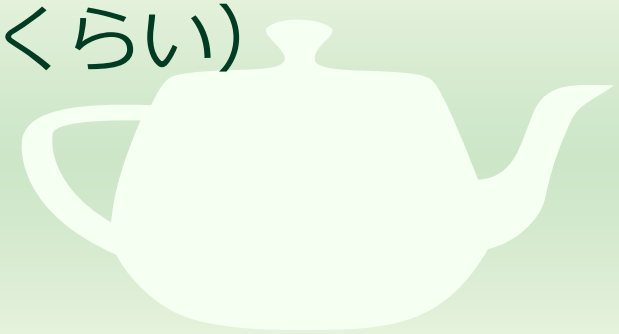
- 命令型プログラミング言語
  - 手続き型プログラミング言語
    - C, C++, C#, Java, JavaScript など大半のプログラミング言語
- 宣言型プログラミング言語
  - 関数型プログラミング言語
    - LISP, Ocaml, Erlang, Scala, Haskell, F#
  - 論理型プログラミング言語
    - Prolog, GHC

パラダイム

ある時代や分野において
支配的規範となる
「物の見方や捉え方」

オブジェクト指向

- プログラミングパラダイムの一つ
  - 命令型・宣言型の分類などとは直交する概念
  - C++, C#, Java, Python など近年のプログラミング言語で採用
- **オブジェクト**
  - 何らかの役割を与えられた実体（メモリ）
- 長くなるので細かい話は**割愛**
  - 別に「オブジェクト指向」という講義がある（くらい）
  - この演習では便利な機能として利用するだけ



C++ 言語によるプログラミング

C++ 言語は初心者に向いていないらしい

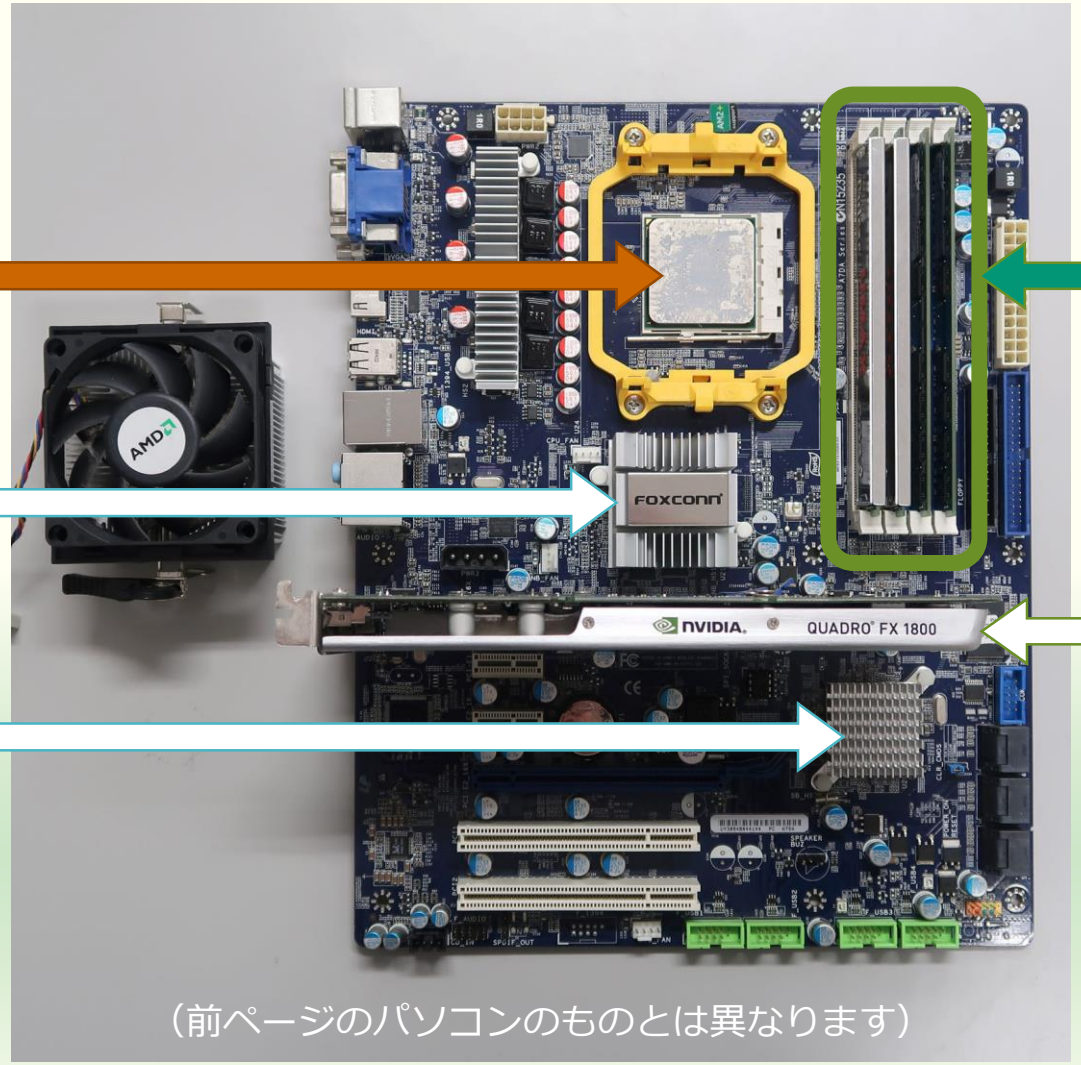
パソコンの外観と内部

マザーボード

CPU

チップセット
（ノースブリッジ）

チップセット
（サウスブリッジ）



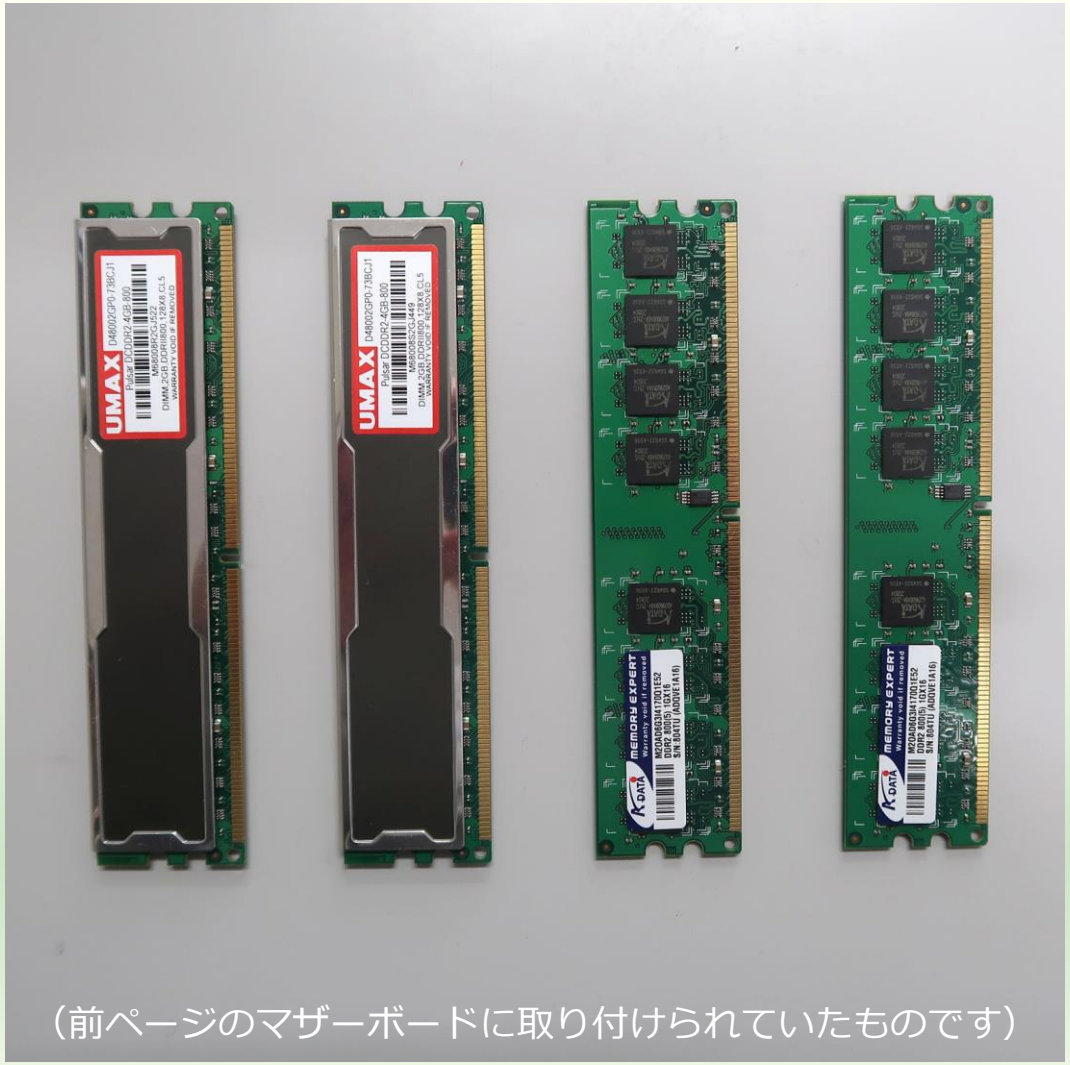
（前ページのパソコンのものとは異なります）

メモリ

ビデオカード
（GPU）

CPU とメモリ

（前ページのマザーボードのものとは異なります）



（前ページのマザーボードに取り付けられていたものです）

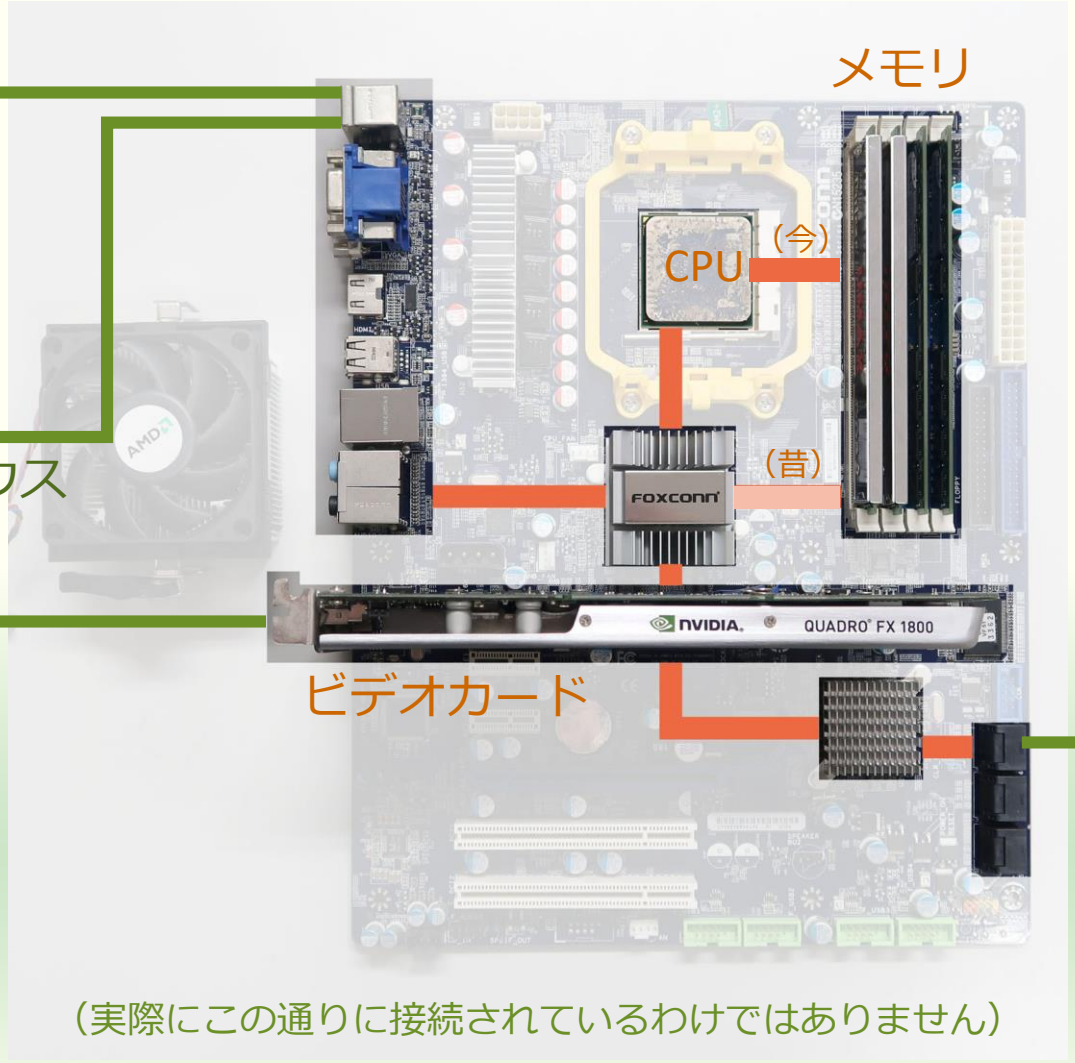
ビデオカード (GPU)

周辺機器との接続

キーボード

マウス

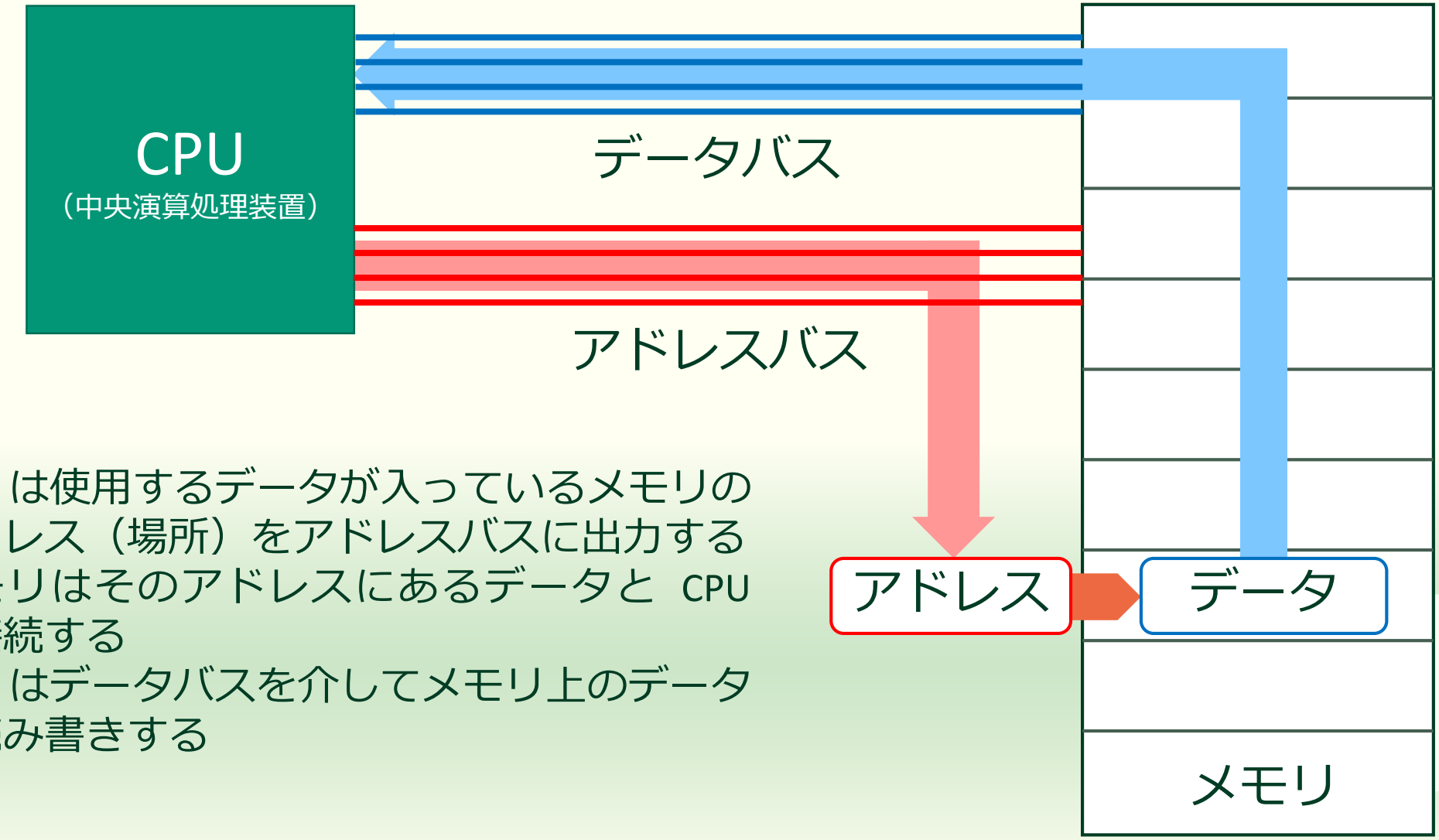
ディスプレイ



（実際にこの通りに接続されているわけではありません）

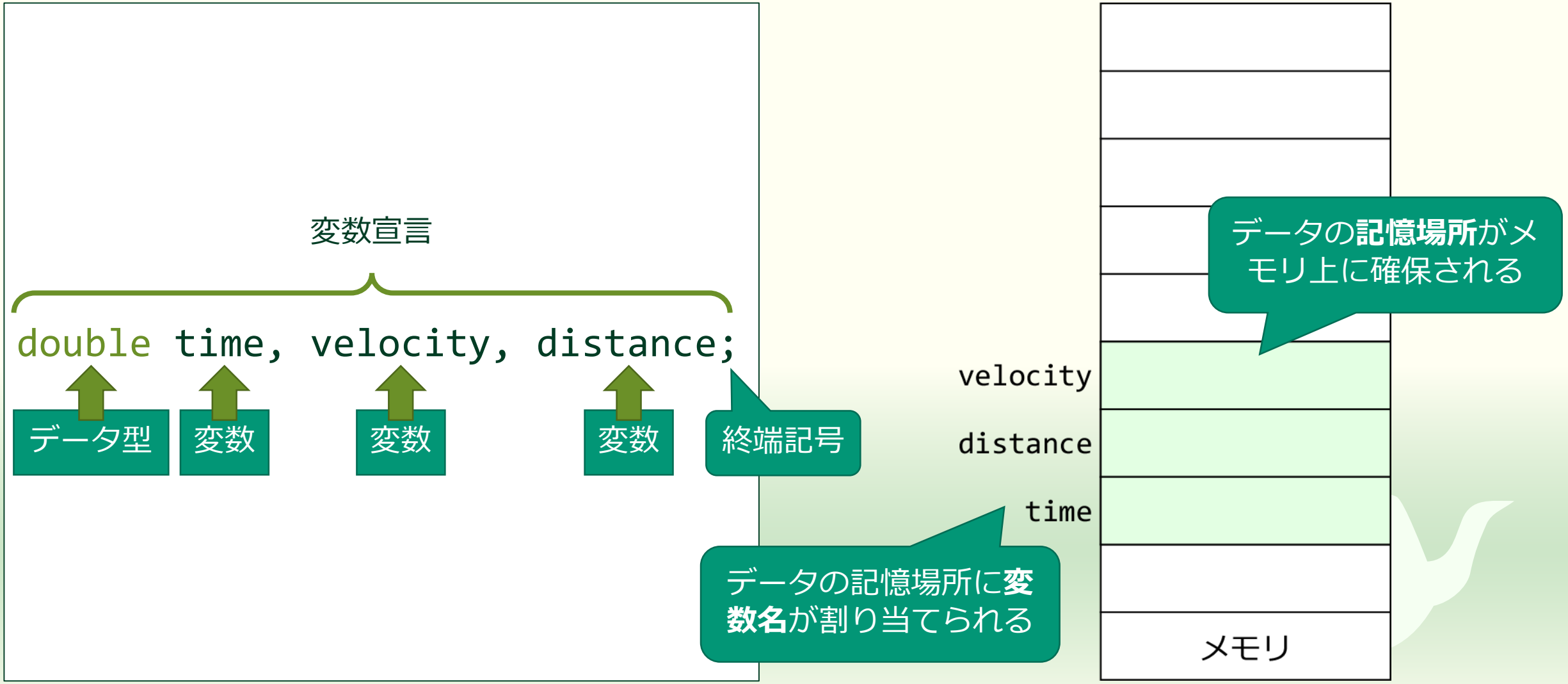
HDD / SSD

CPU とメモリの関係



1. CPU は使用するデータが入っているメモリのアドレス（場所）をアドレスバスに出力する
2. メモリはそのアドレスにあるデータと CPU を接続する
3. CPU はデータバスを介してメモリ上のデータを読み書きする

メモリを使うには変数を宣言する



データ型（基本）

整数型	
char	■ 8ビット精度符号付き整数 ■ -128～127
int	■ 32ビット精度符号付き整数 ■ -2,147,483,648～2,147,483,647

実数型	
float	■ 32ビット単精度浮動小数点 ■ 精度は10進で約6.92桁
double	■ 64ビット倍精度浮動小数点 ■ 精度は10進で約15.65桁

整数型のバリエーション	
signed char	char と同じ
unsigned char	8ビット符号無し整数, 0～255
short, short int, signed short int	16ビット符号付き整数, -32,768～32,767
unsigned short, unsigned short int	16ビット符号無し整数, 0～65,535
signed, signed int, long, long int, signed long int	int と同じ
unsigned, unsigned int, unsigned long, unsigned long int	32ビット符号無し整数, 0～4,294,967,2965

そのほかのデータ型（基本）

bool	論理型, true (真) と false (偽) の値だけを持つ
enum	列挙型, 名前を付けた定数の組のうちのどれか一つの値を持つ
long long, signed long long	64ビット精度符号付き整数, -9,223,372,036,854,775,808～9,223,372,036,854,775,807
unsigned long long	64ビット精度符号無し整数, 0～18,446,744,073,709,551,615
long double	double と同じ

int と long が同じ理由

初期のパソコンの CPU は 8 ビットで、それが 16 ビット、32 ビット、64 ビットと発展してきました。C++ 言語のもとになった C 言語がパソコンで使われ始めたのは 16 ビットの頃だったため、その int は 16 ビット、long は 32 ビットになっていました。パソコンの CPU が 32 ビット化したことで C 言語の int にも 32 ビット割り当てられるようになりましたが、long は互換性のために 32 ビットのままに据え置かれました。CPU が 64 ビット化したときも互換性のために int、long ともに 32 ビットに据え置かれました。なお、64 ビットの整数を扱う場合は long long、unsigned long long が使用されます。

long double と double が同じ理由

これは Visual Studio の C++ 言語処理系である Visual C++ の仕様で（int と long の関係も Visual C と x86 系の CPU に依存した話）、他の言語処理系では同じ x86 系でも long double が 80 ビットになっているものもあります。これは初期の x86 系 CPU にオプションで追加する浮動小数点演算ハードウェアが内部的に 80 ビットで計算していたのをそのまま使えるようにするためですが、データサイズは 128 ビットとなり 48 ビット無駄に使います。128 ビットをフルに使い切る精度の実数データを扱えるように考えられてはいるものの、現時点では実装がまちまちの状態になっているようです。

くどいようだが整数型と実数型がある

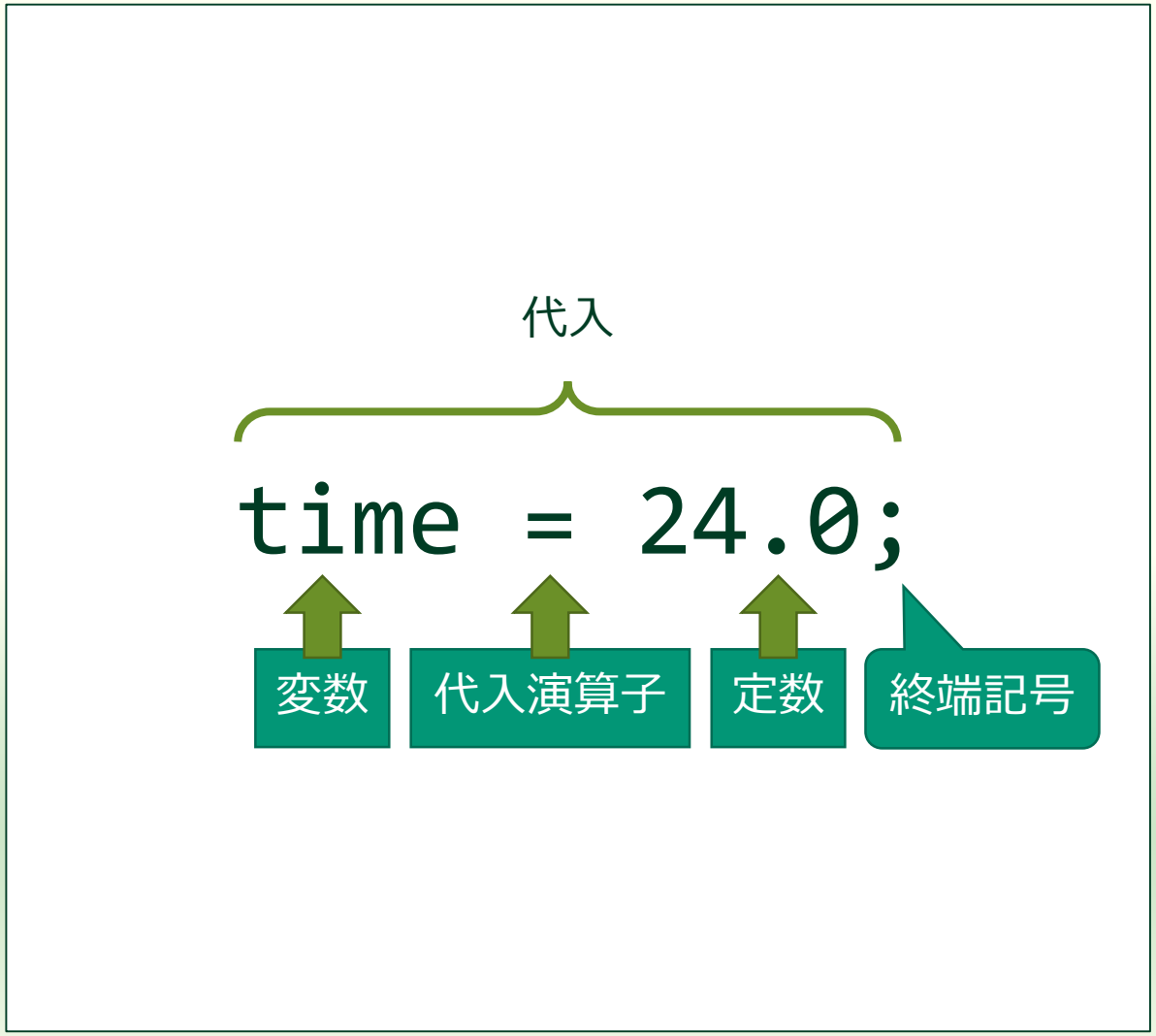
整数型（int, char 等）

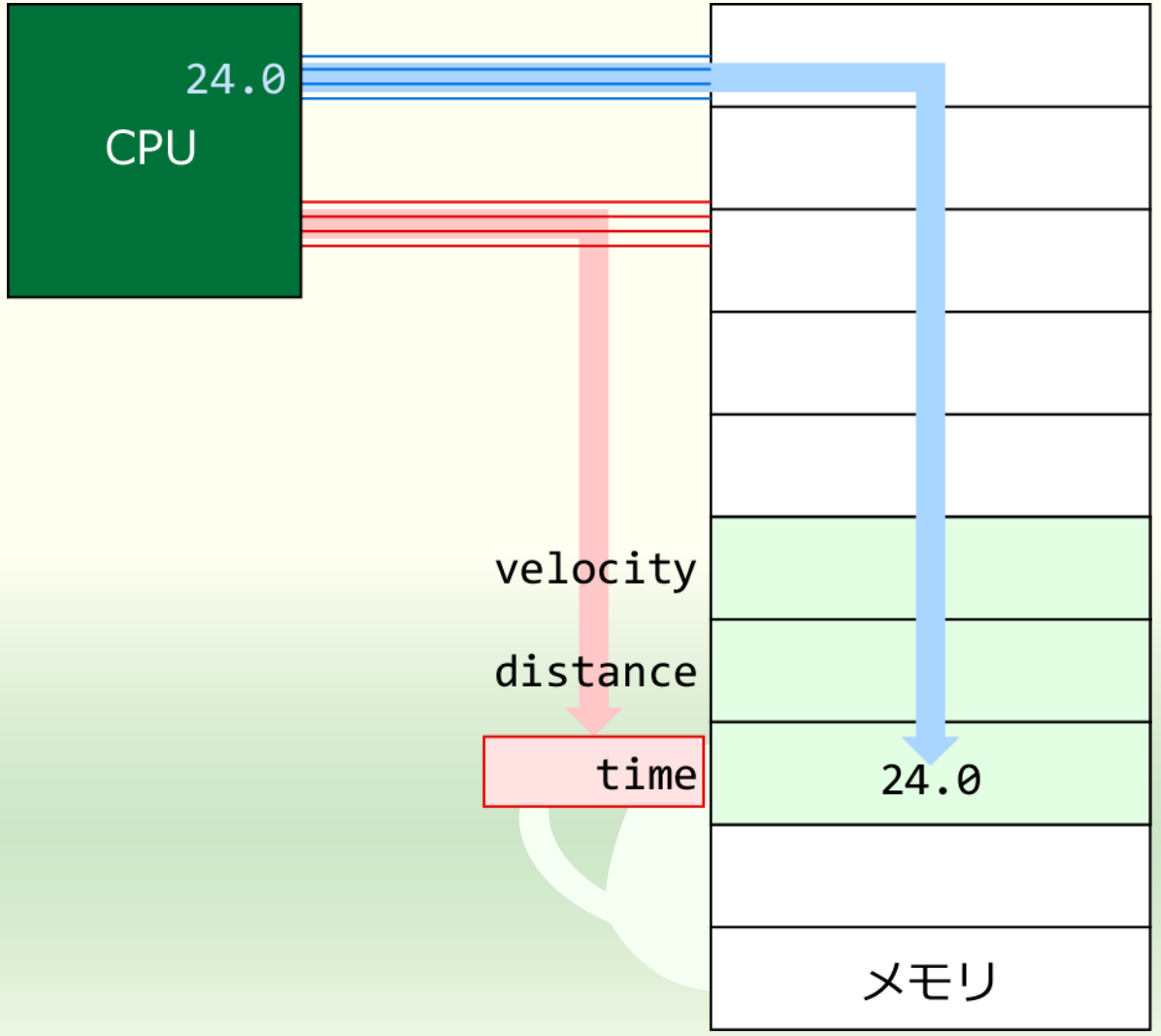
- （主に）数を数えるのに使う
  - 小数点以下を持たない
  - 隣り合う数値の間隔は 1
  - 除算は小数部が**切り捨てられる**
    - 5 / 3 == 1
    - -5 / 3 == -1
    - 1 / 2 == 0
  - 実数と混在するときは実数になる
    - 5 / 3.0f ≒ 1.66667
    - 1.0f / 2 == 0.5f

実数型（float, double 等）

- 数値計算に使う
  - 小数点以下を持つ
  - 隣り合う数値の間隔が一定でない
    - 0.1f は実は 0.1 の**近似値**
  - 非常に大きな数や非常に小さな数が表現できる
  - 指数表記が可能
    - 1.23×10^5 は 1.23e5f
    - 末尾に f が付いているものは float 型の定数、無ければ double 型の定数

データを記憶するには変数に代入する





手続き（処理）は関数として記述する

- 数学
  - 関数を定義する

$$f(x) = x^2$$

  - 関数を評価する

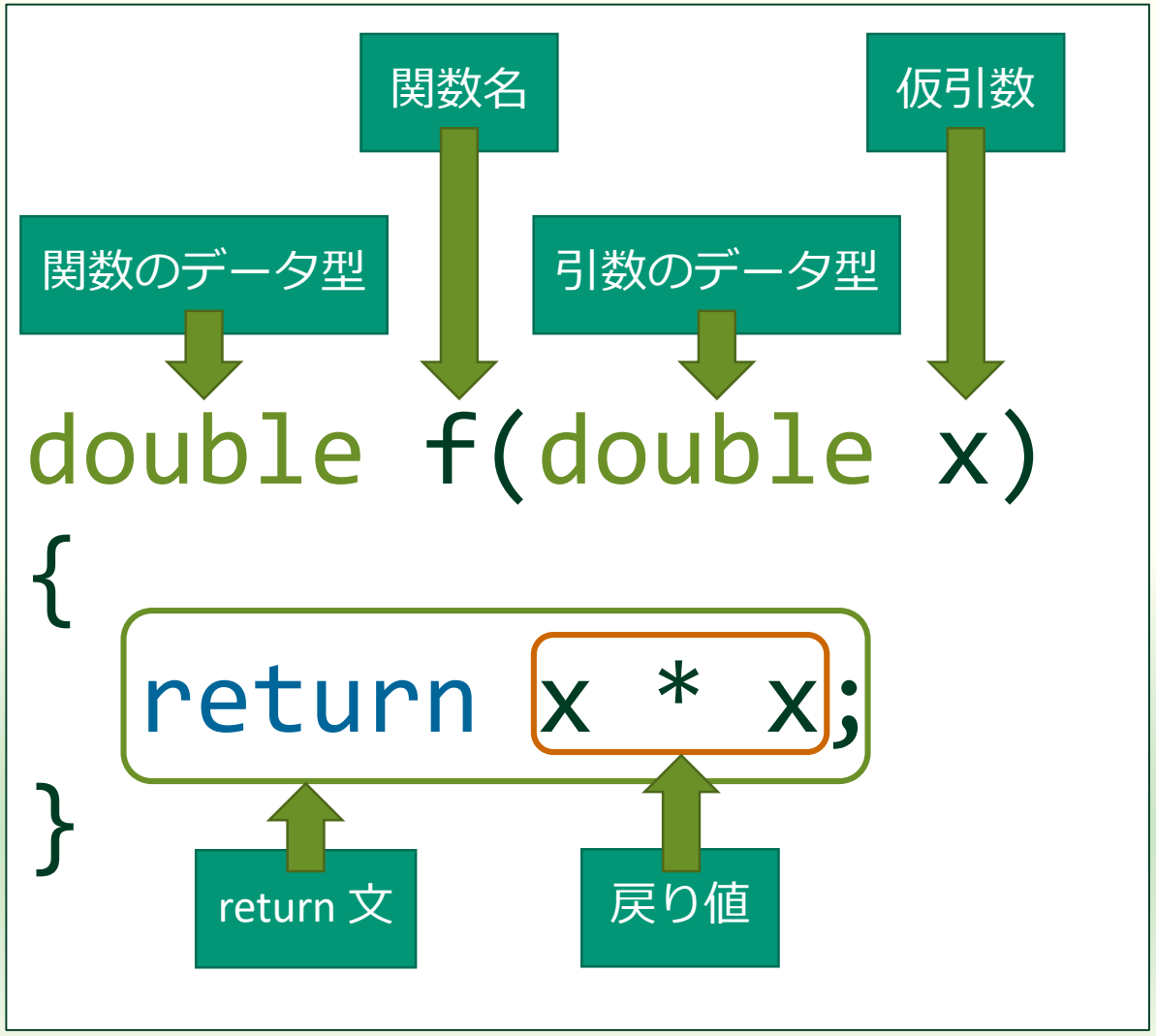
$$y = f(2)$$

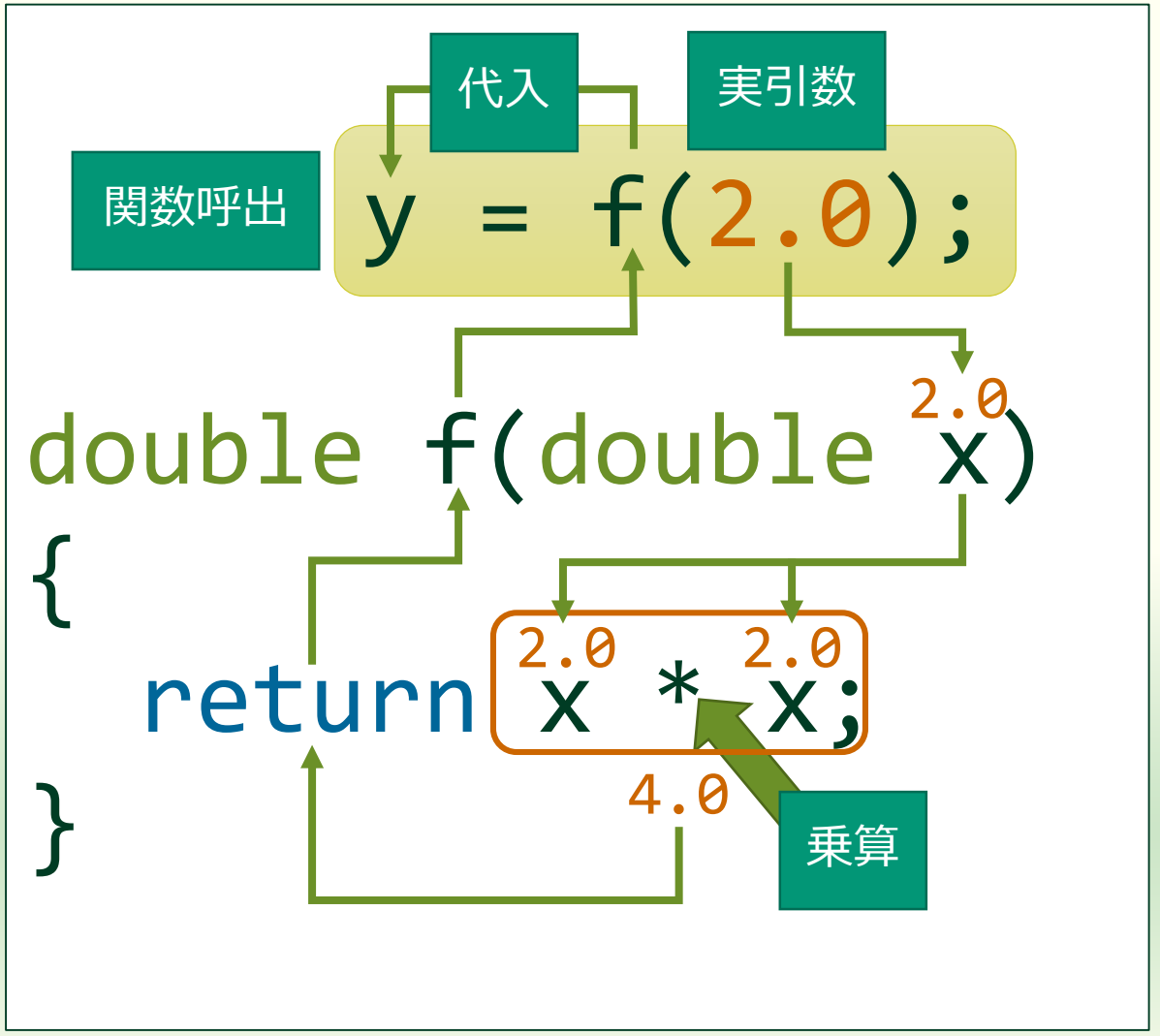
これは y が $f(2)$ の値と**等しい**という意味

```
//
// 関数名が f の関数の定義
//
double f(double x)
{
  // 関数の戻り値として引数 x の二乗を返す
  return x * x;
}

int main()
{
  double y;

  // 関数 f を評価した値を y に代入する
  y = f(2.0);
}
```

これは y に f(2.0) の値を**代入**（格納）**する**という意味

関数の定義と関数の呼出し





“//” より右はコメント

```
//
// 関数名が f の関数の定義
//
double f(double x)
{
  // 関数の戻り値として引数 x の二乗を返す
  return x * x;
}
```

コメント
プログラムの動作に
影響を与えない
メモや注釈

最初に評価される関数 main()

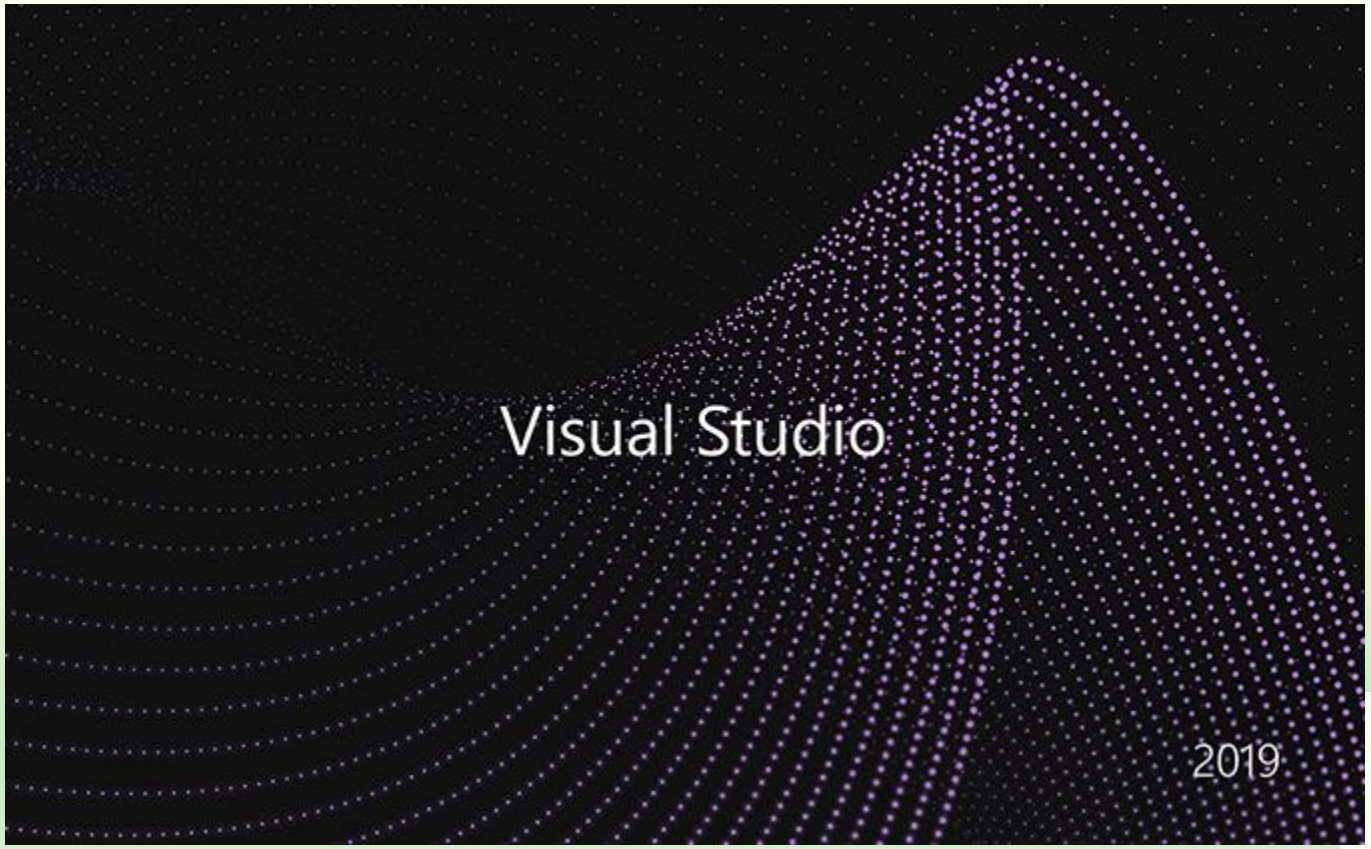
```
//
// 関数名が f の関数の定義
//
double f(double x)
{
  // 関数の戻り値として引数 x の二乗を返す
  return x * x;
}

//
// メインプログラム
//
int main()
{
  // 変数宣言
  double y;

  // 関数 f を評価した値を y に代入する
  y = f(2.0);
}
```

- プログラムは main() 関数から実行を開始する
- main() から関数 f() が呼び出される
- 関数 f() が実行された後 main() に戻る
- main() の return 文を省略すると 0 を返す

例題

Visuall Studio による C++ プログラム作成

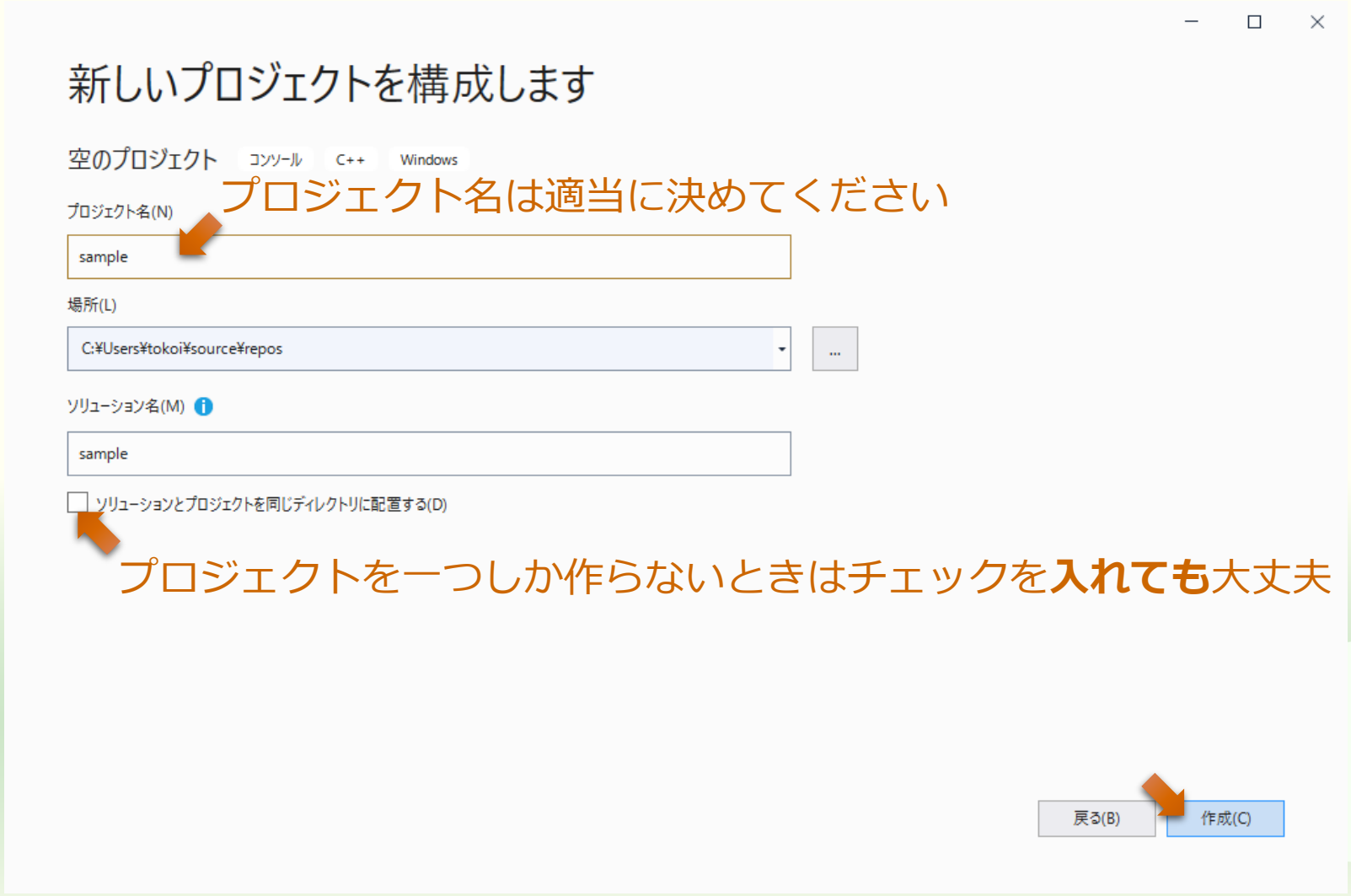
Visual Studio 2019 を起動する



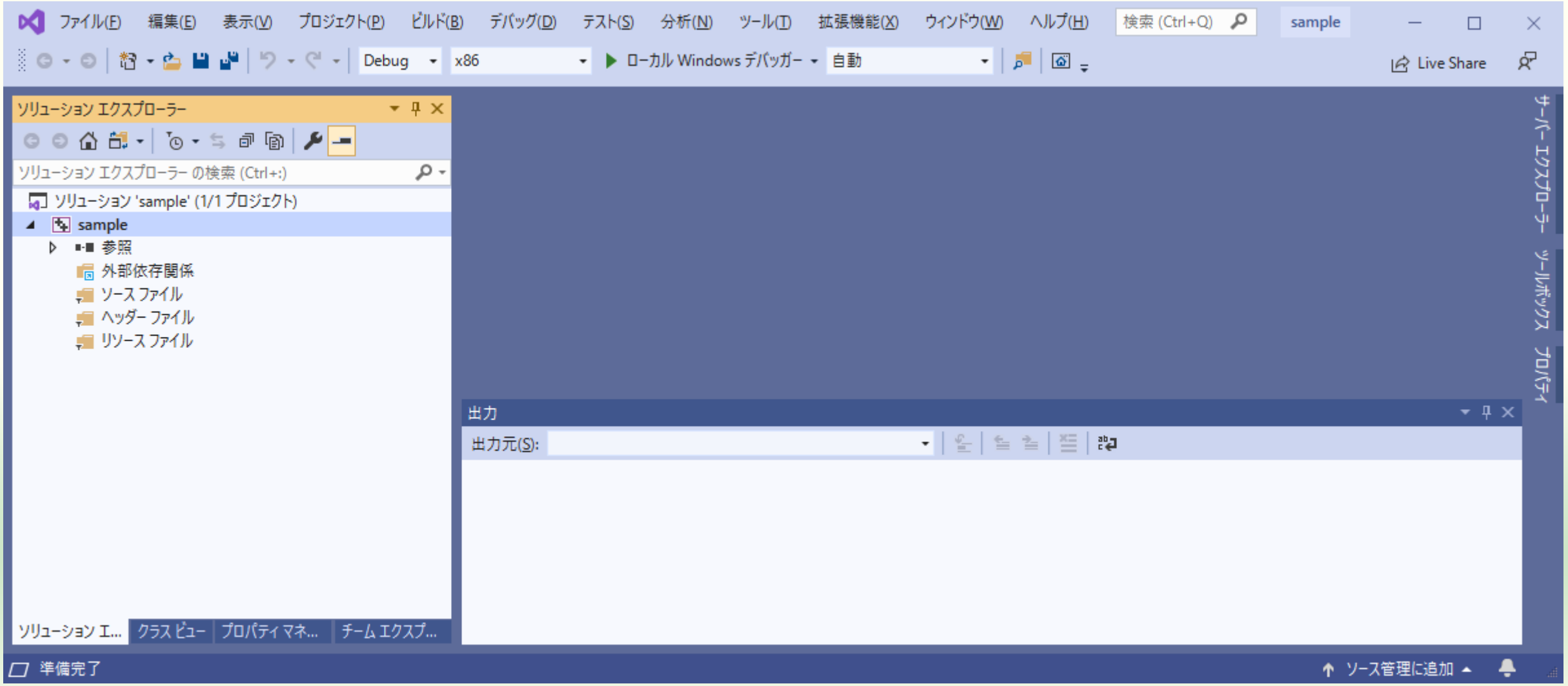
「新しいプロジェクトの作成 (N)」を選ぶ

「空のプロジェクト」を作成する

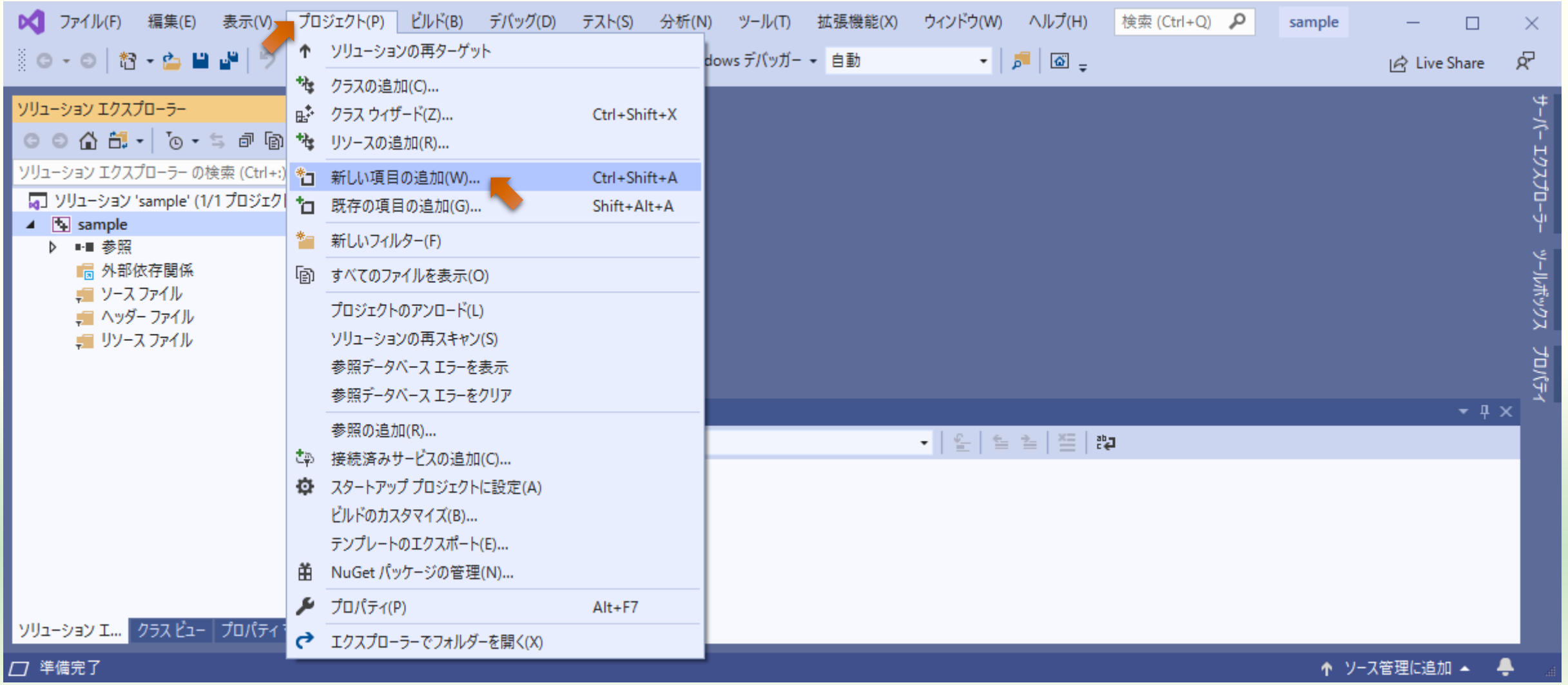
「プロジェクト名 (N)」を入力する



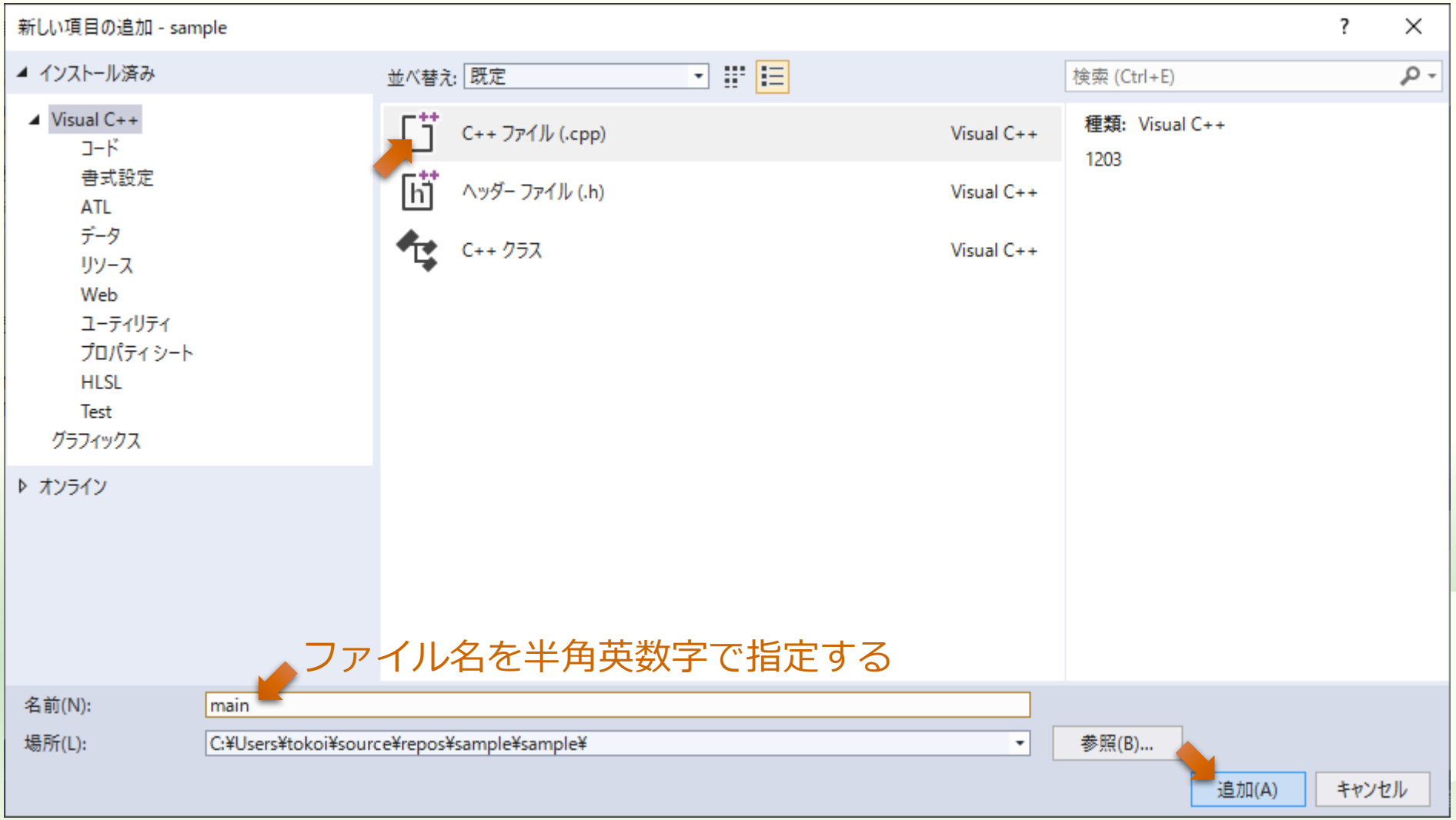
新しいプロジェクトが作成される



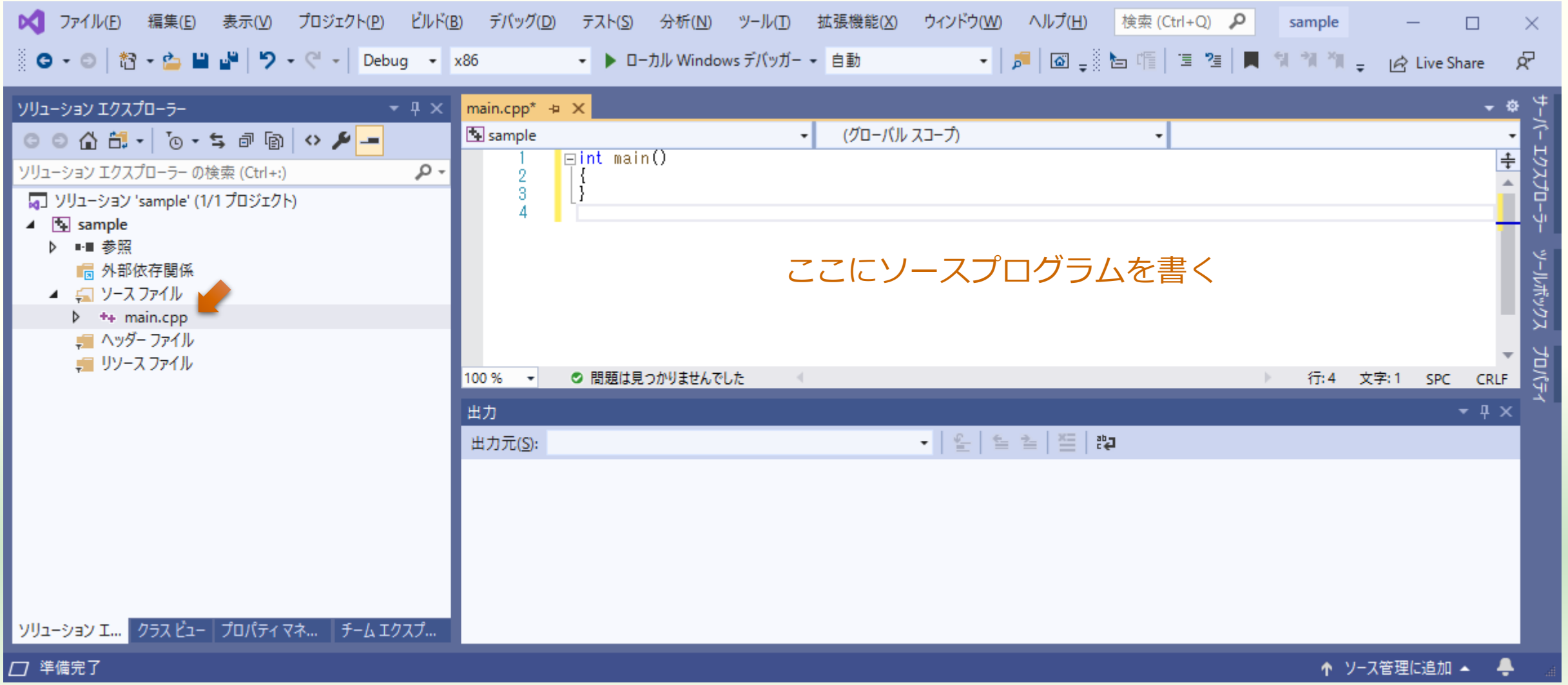
「新しい項目の追加 (W)」を選ぶ



「C++ ファイル (.cpp)」を追加する



ソースプログラムの編集

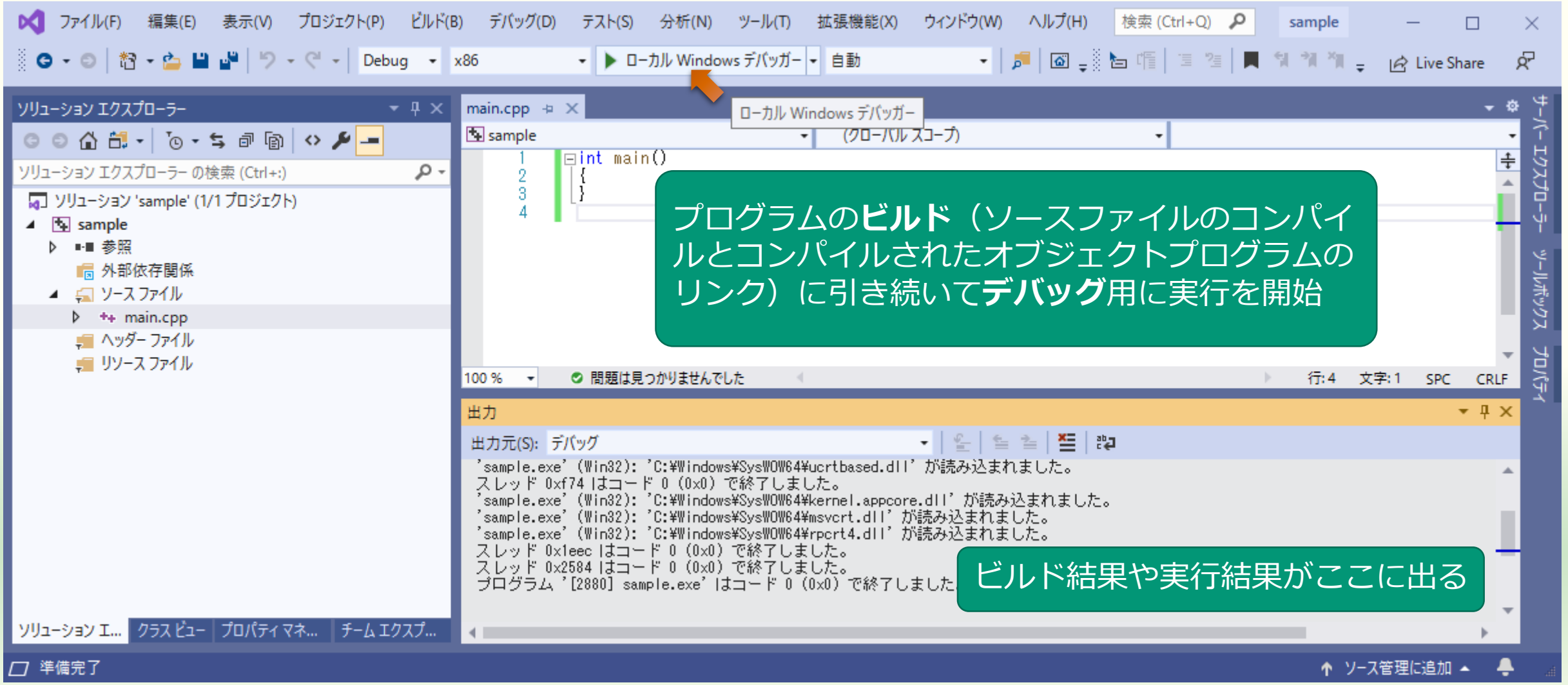


作成するプログラム

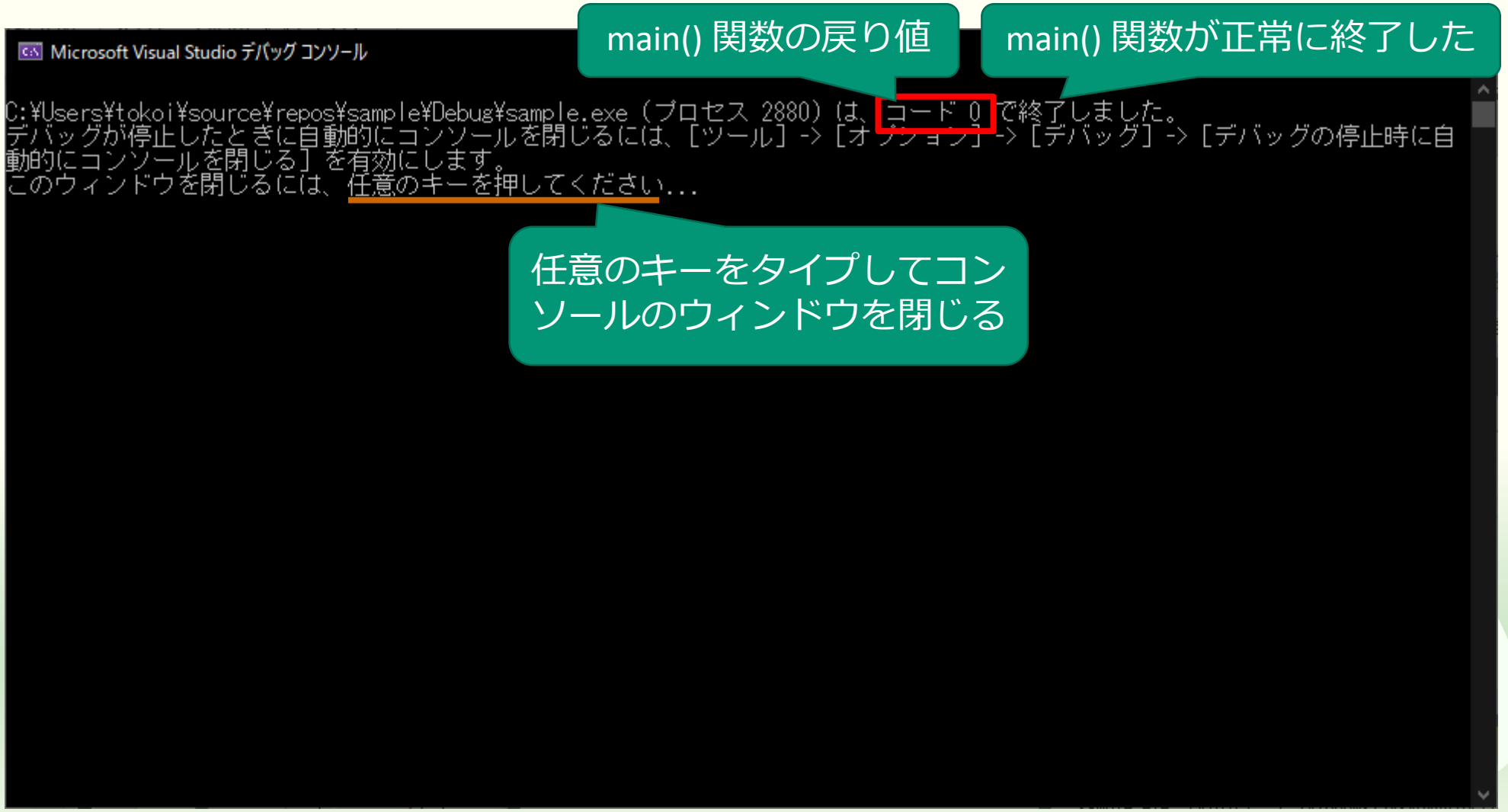
```
int main()
{
}
```

- **main() 関数**を定義する
  - このプログラムには中身がないので何もしない
  - main() 関数の戻り値のデータ型は int 型
- **main() 関数に限り** return を省略しても戻り値として 0 を返す

プログラムのビルドと実行



プログラムのコンソール出力

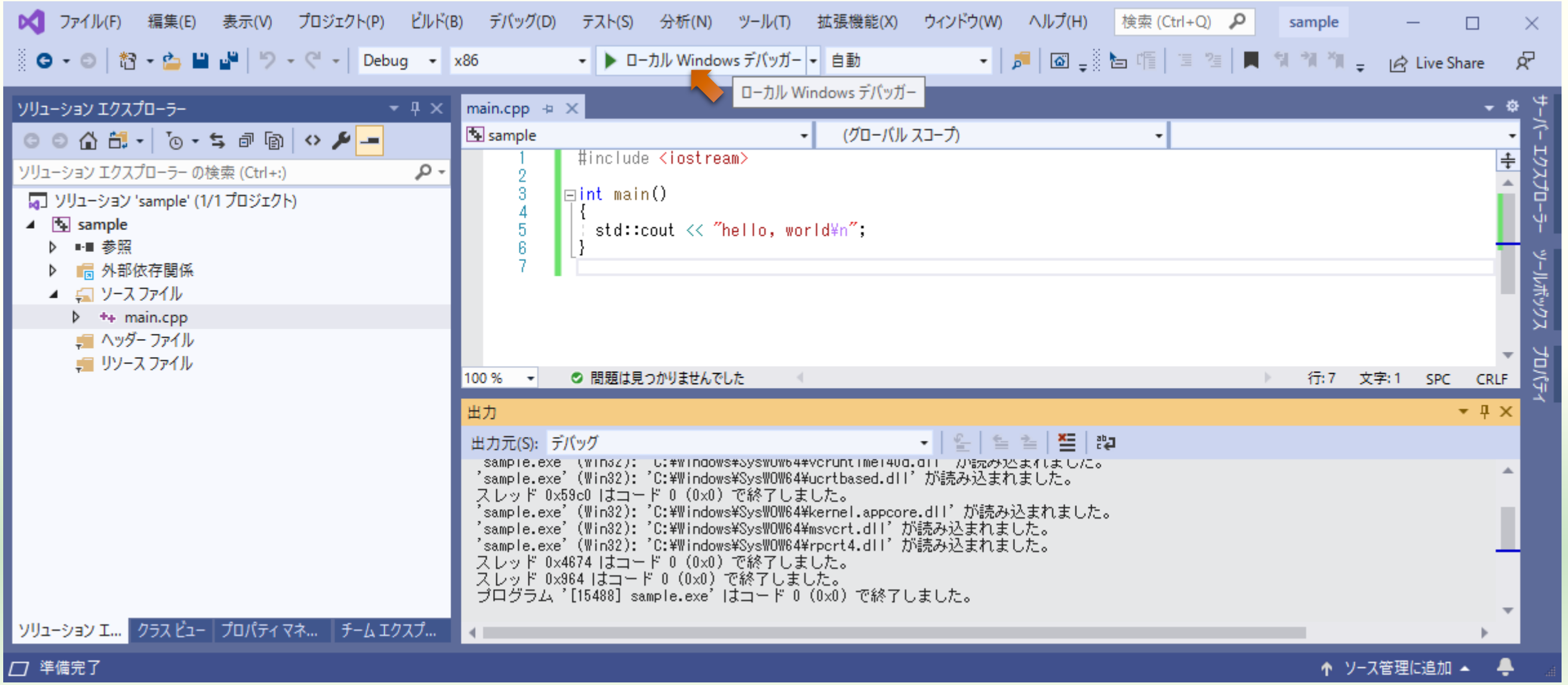


コンソールに文字を出力する

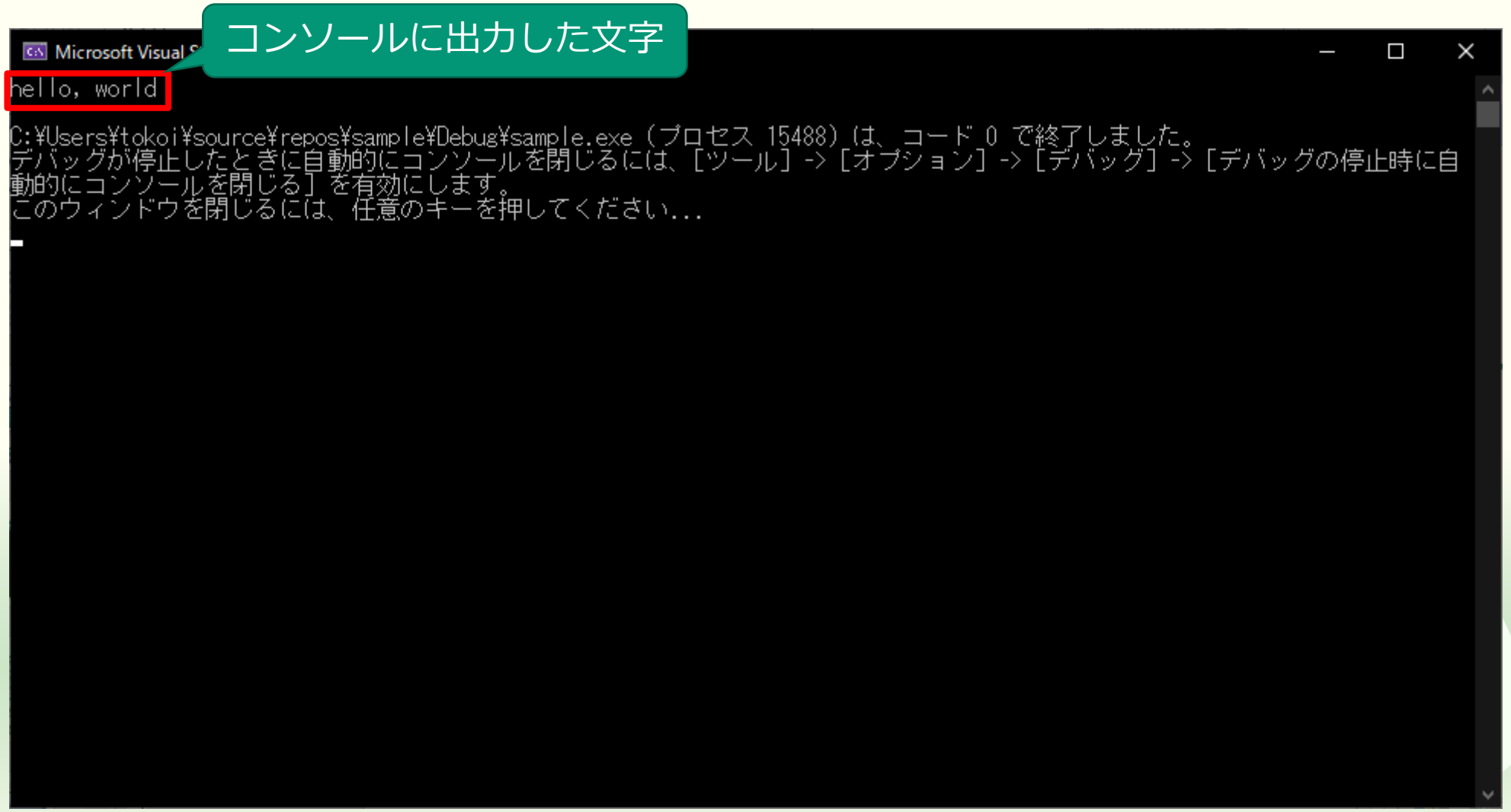
```
#include <iostream>

int main()
{
  std::cout << "hello, world¥n";
}
```

- マーカー部分を追加する
  - #include <iostream>
    - iostream という**標準ライブラリ**の定義をソースプログラムのこの部分に埋め込む
    - 標準ライブラリは C++ 言語に最初から用意されている機能
  - std::cout << "hello, world¥n";
    - コンソールに hello, world を表示
    - std::cout は標準ライブラリに含まれるコンソール出力の機能

修正したプログラムのビルドと実行



修正したプログラムのコンソール出力

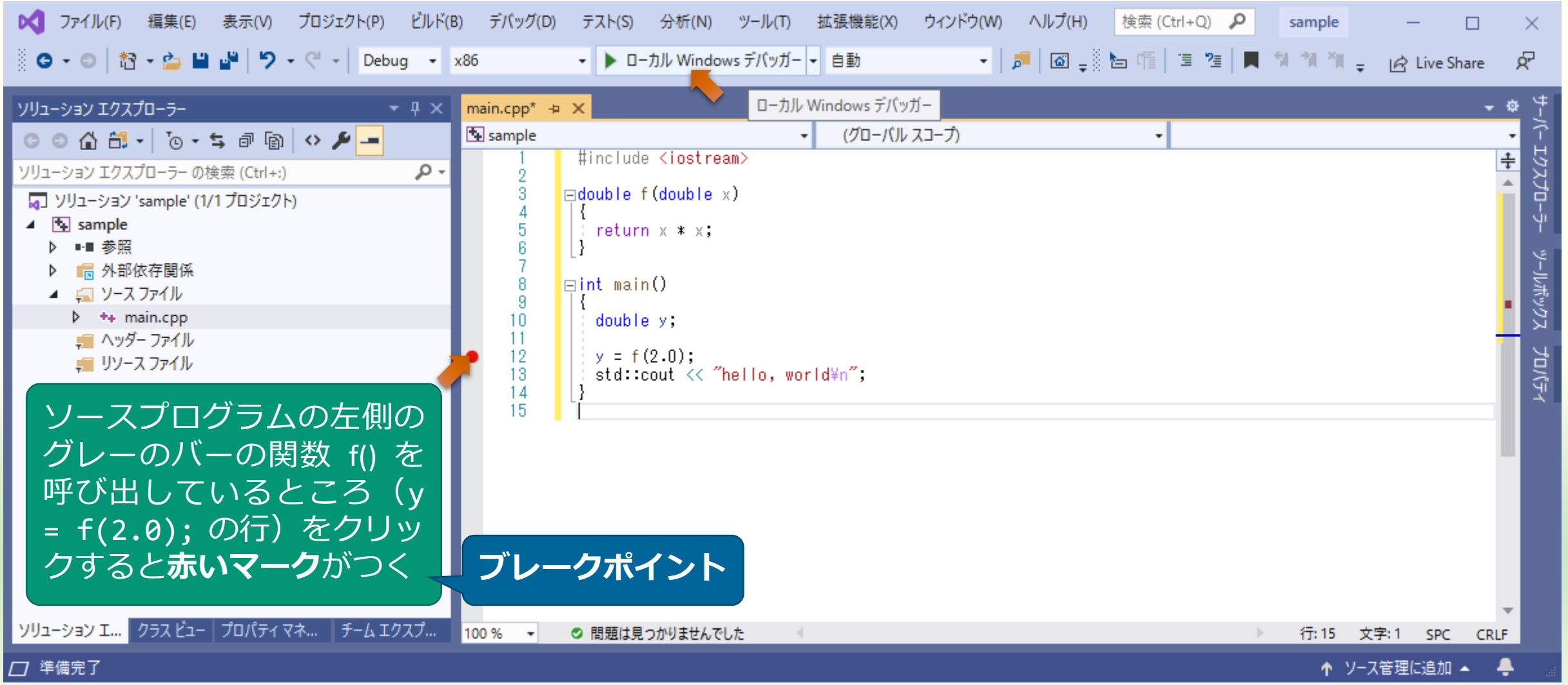


関数を定義する

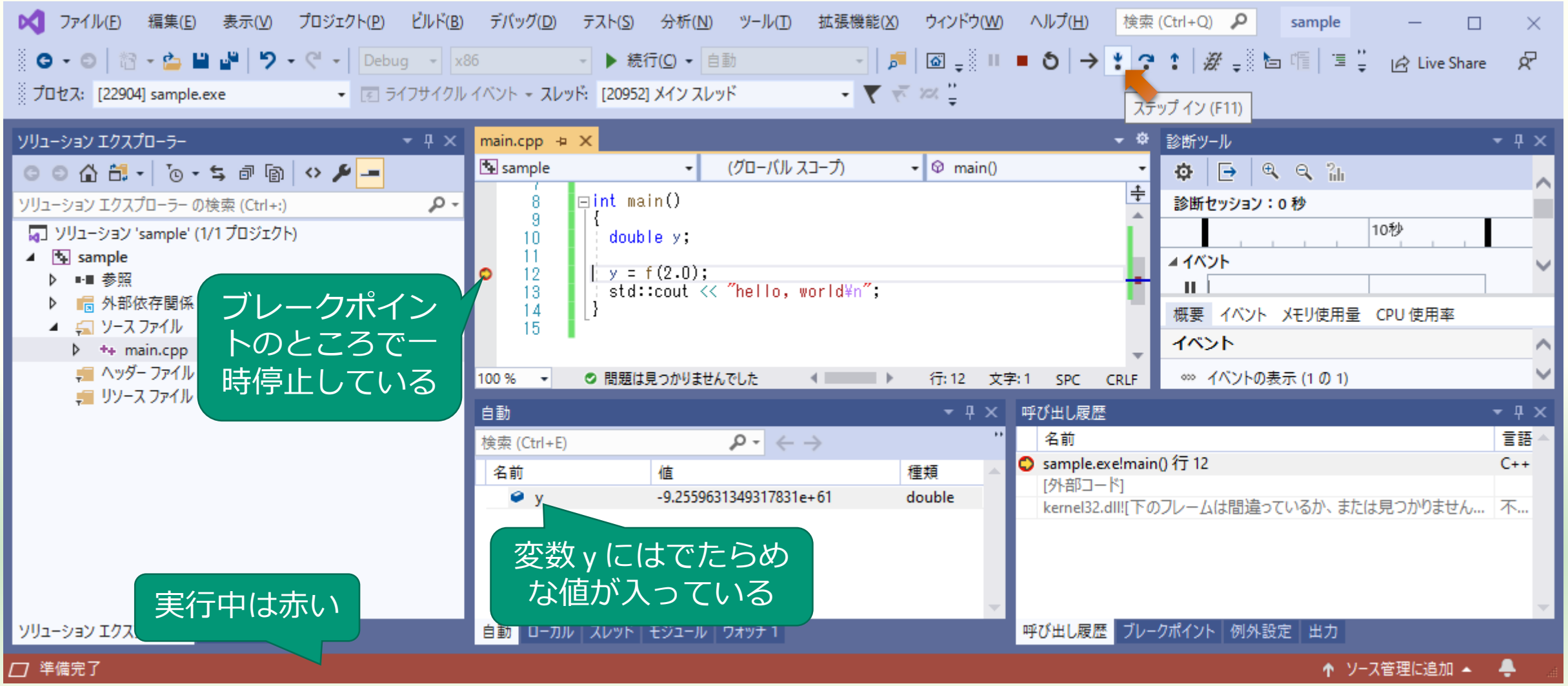
```
#include <iostream>

double f(double x)
{
  return x * x;
}

int main()
{
  double y;

  y = f(2.0);
  std::cout << "hello, world¥n";
}
```

- マーカー部分を追加する
  - double f(double x)
    - 戻り値のデータ型 double の関数 f() を定義する
    - 仮引数 x のデータ型は double
    - 戻り値として x * x を返す
  - y = f(2.0);
    - 実引数に 2.0 を指定して関数 f() を呼び出す
    - 関数 f() の戻り値は変数 y に代入する

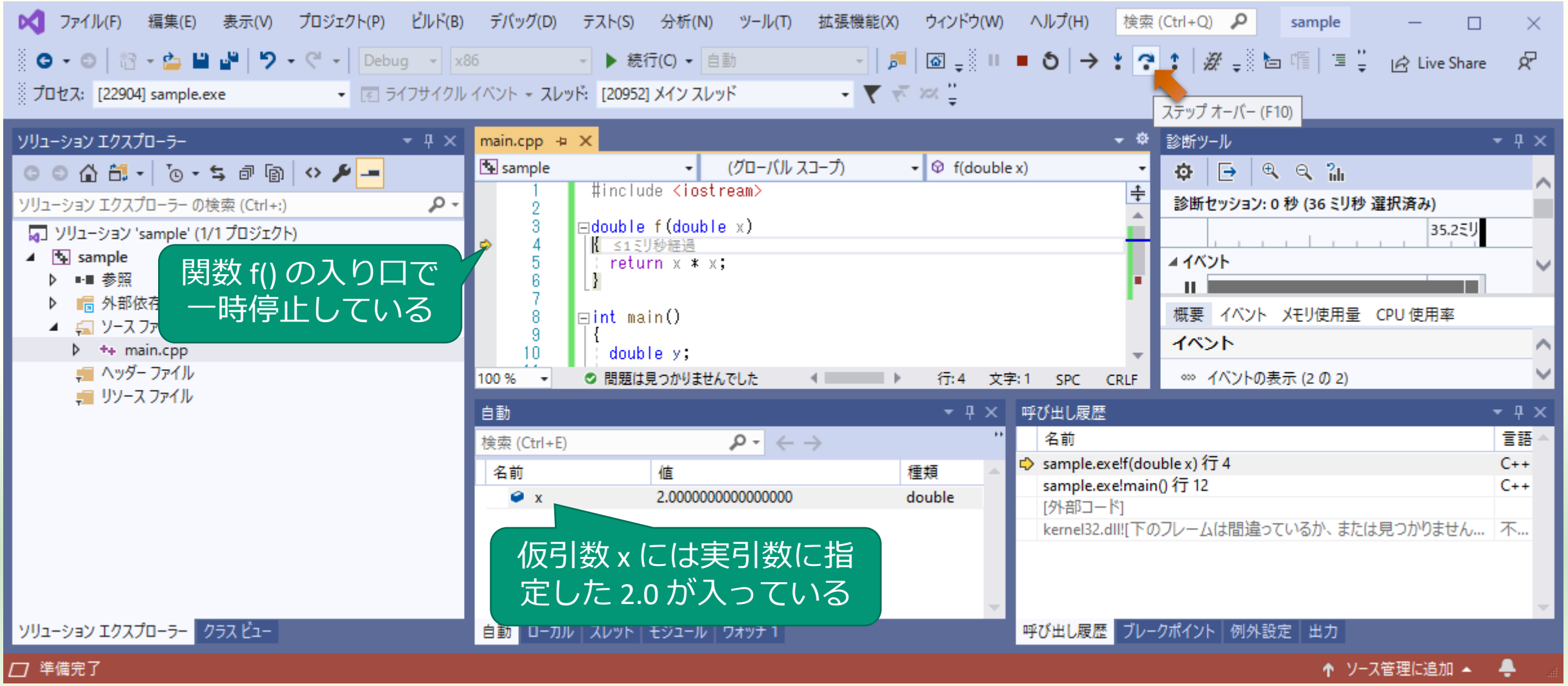
ブレークポイントを設定してビルドと実行



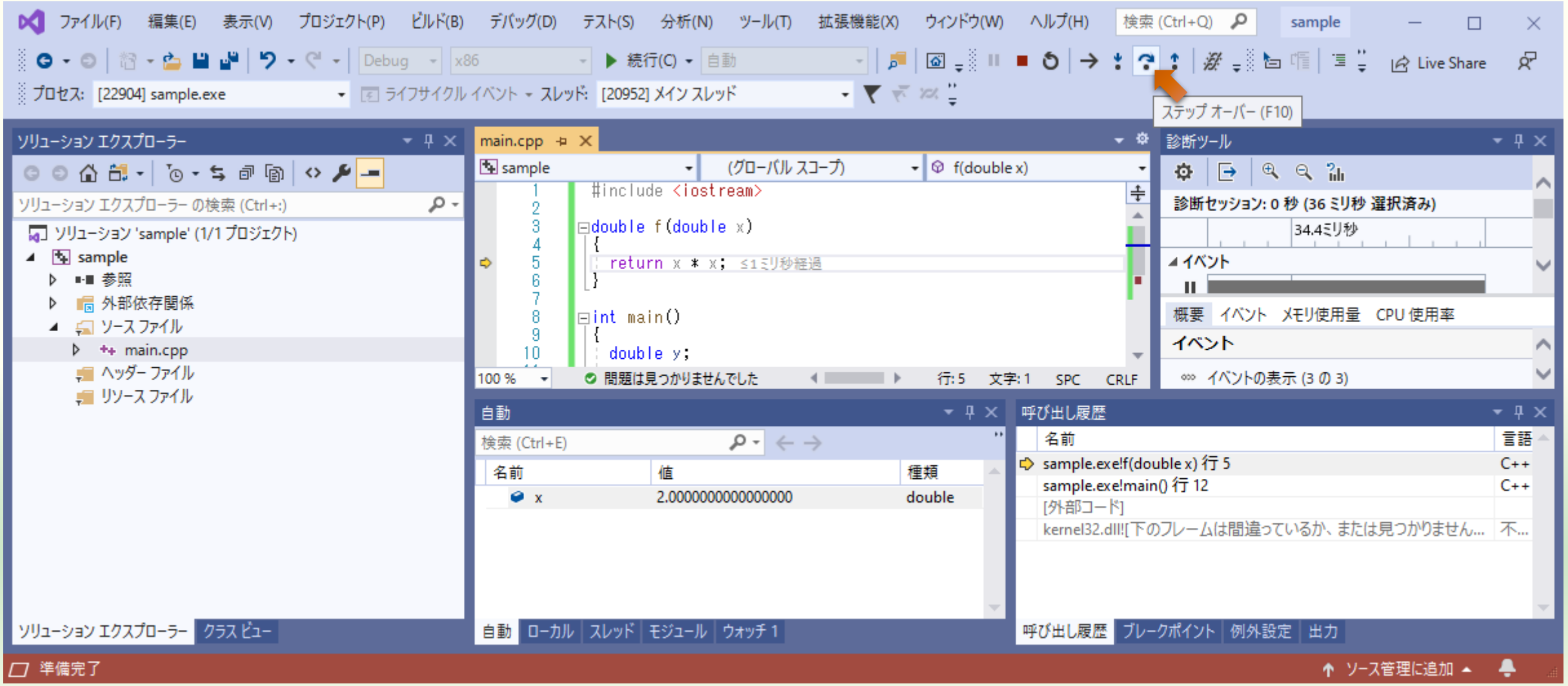
一時停止したらステップインする



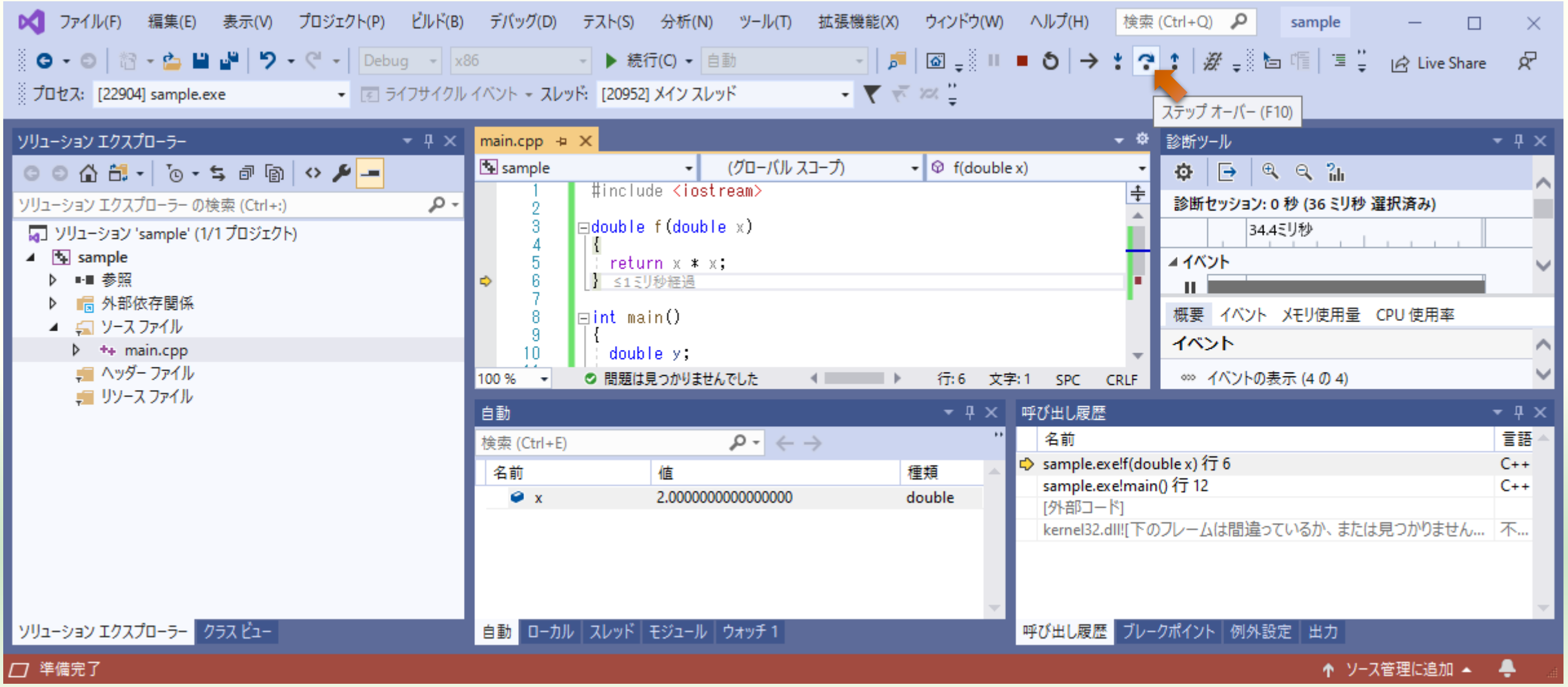
関数 f() に移動するのでステップオーバー



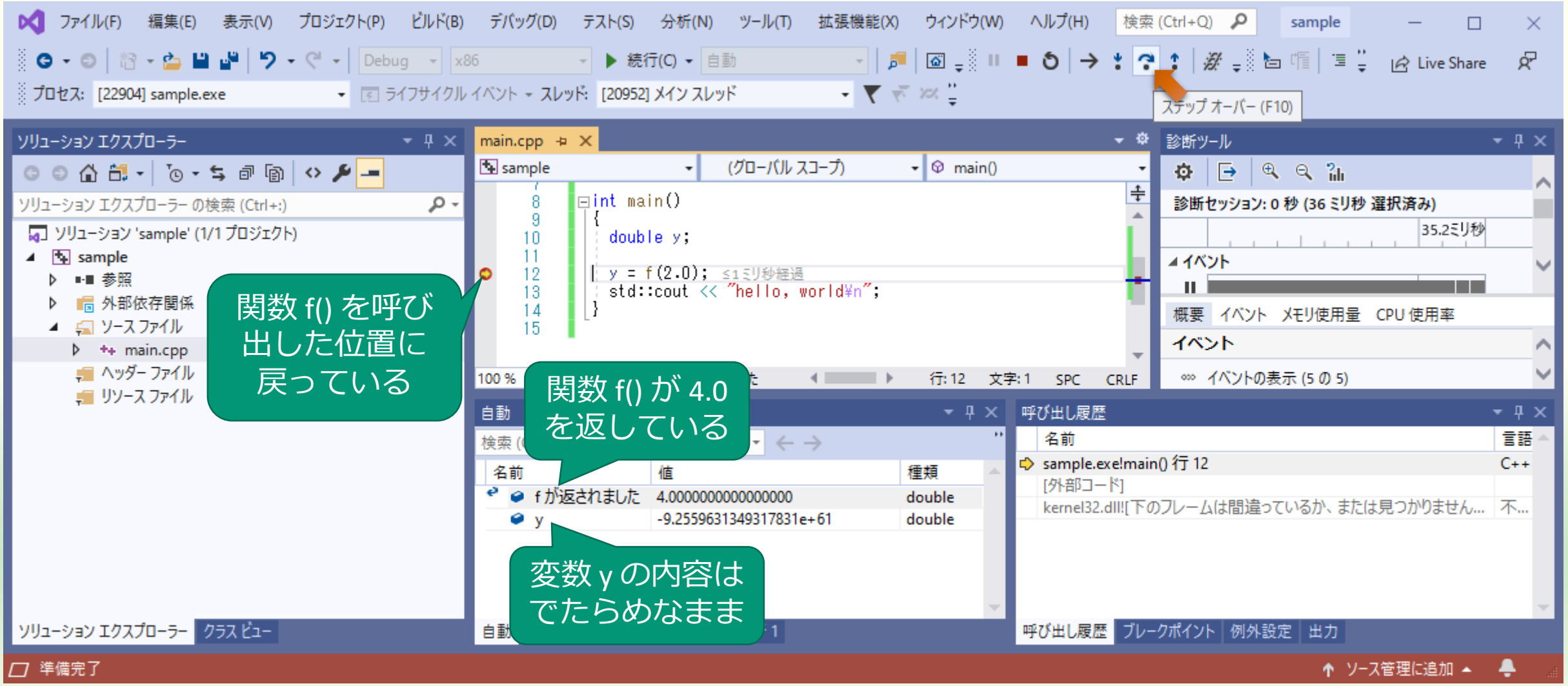
次の行に進んだらステップオーバー



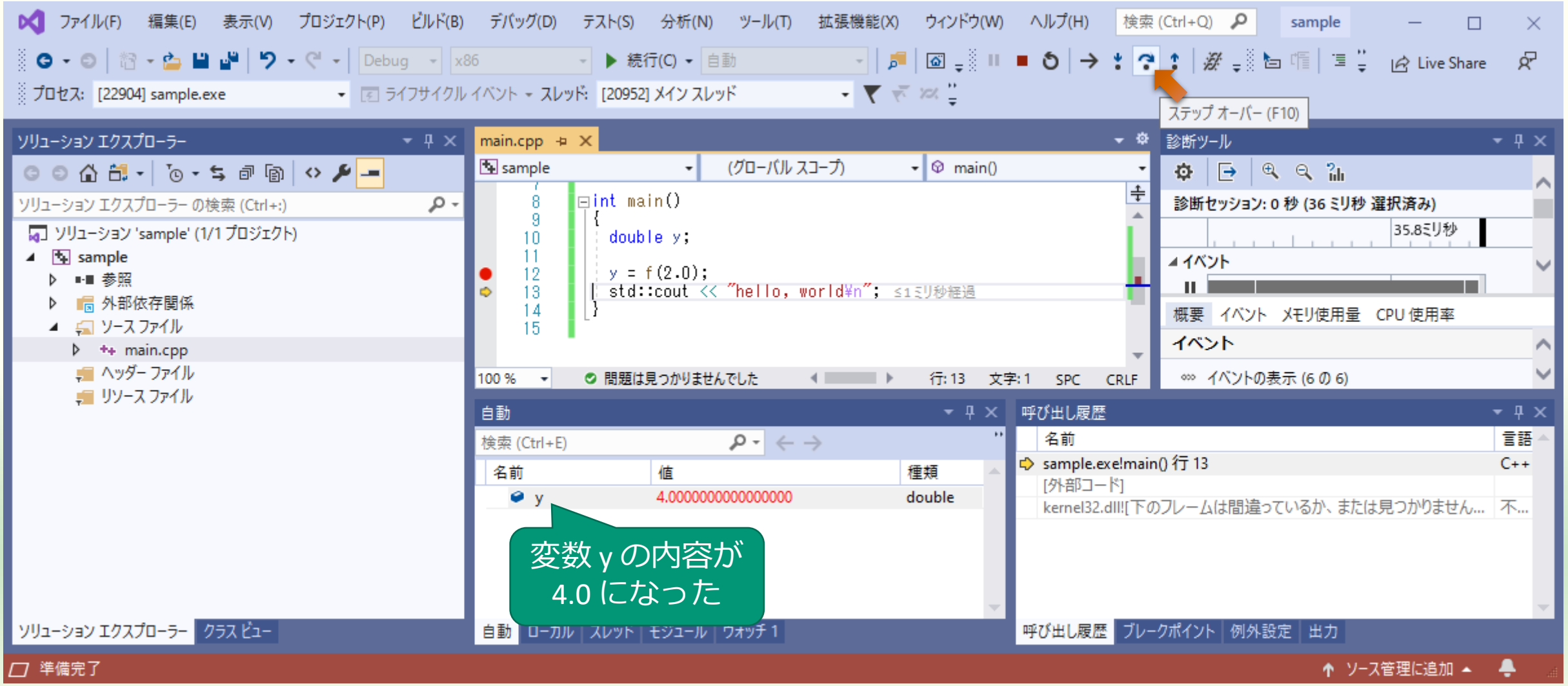
もう一度ステップオーバー



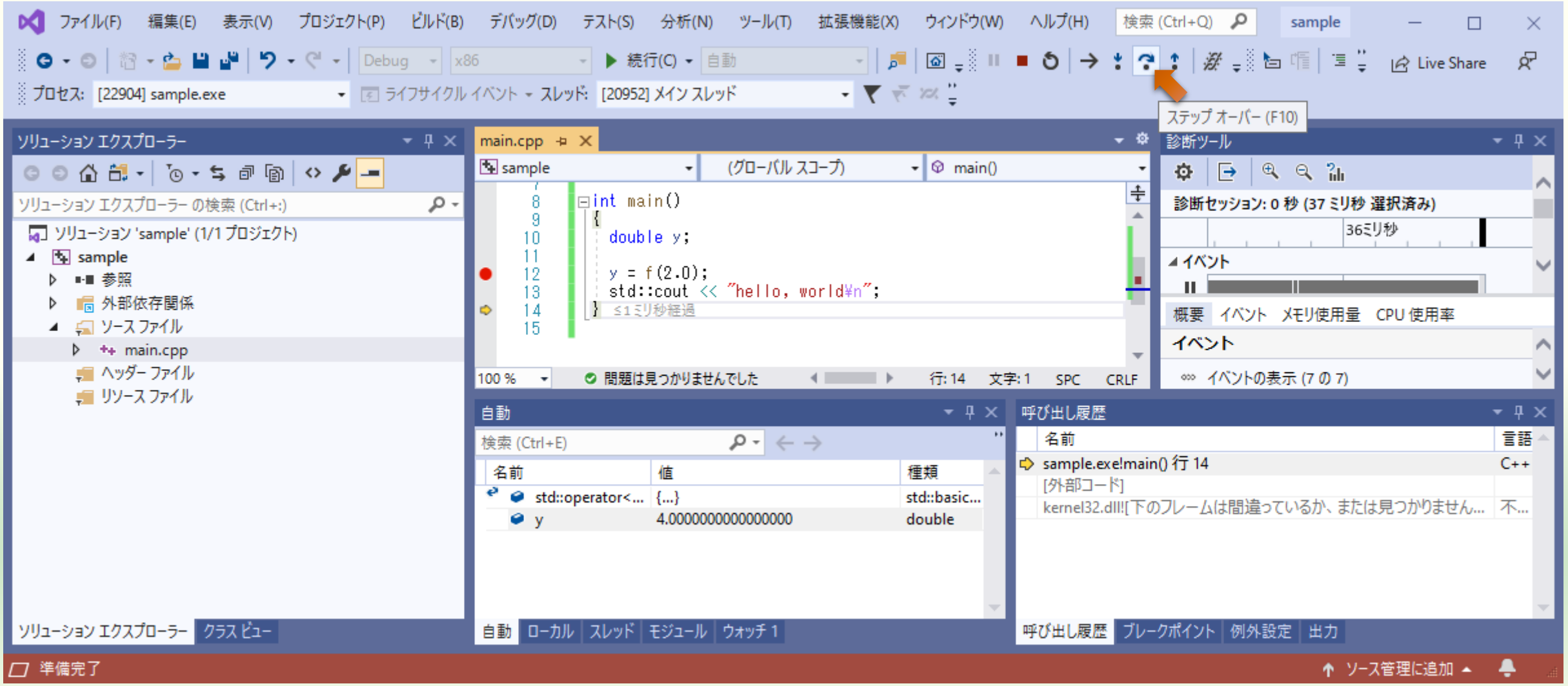
main() 関数の f() の呼び出し位置に戻る



ステップオーバーすると y に代入される

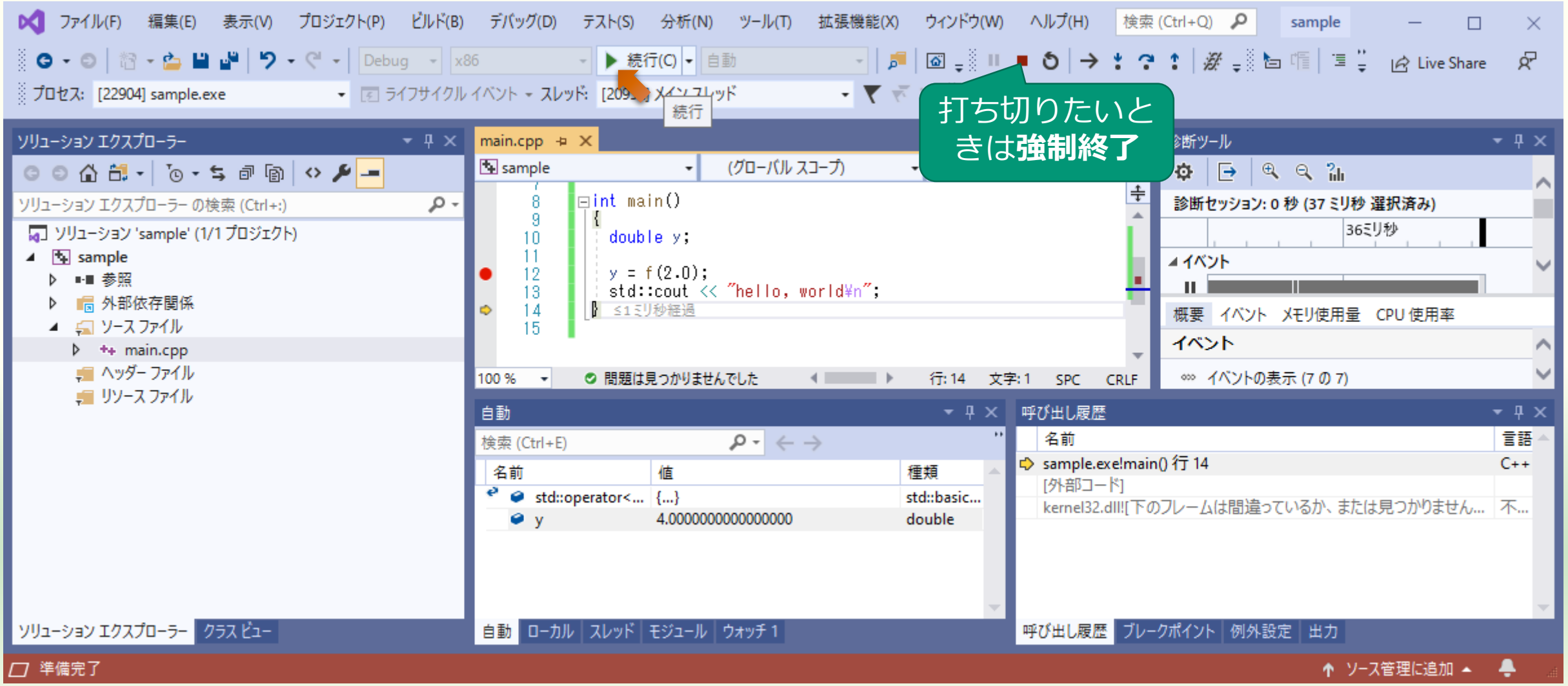


std::cout をステップオーバーする

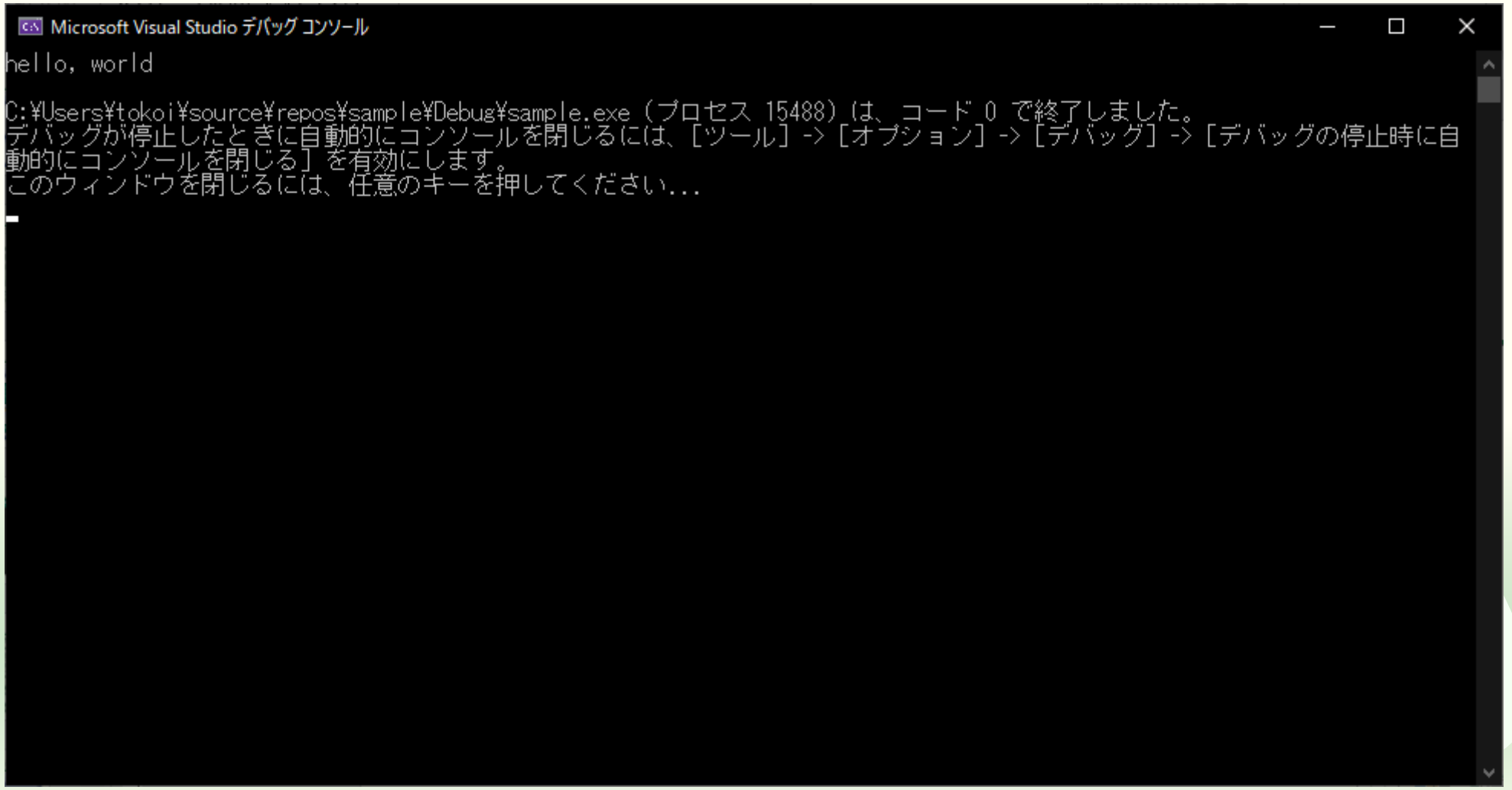


コンソールに出力される

「続行」すると残りを一気に実行する



プログラムの実行が終了する

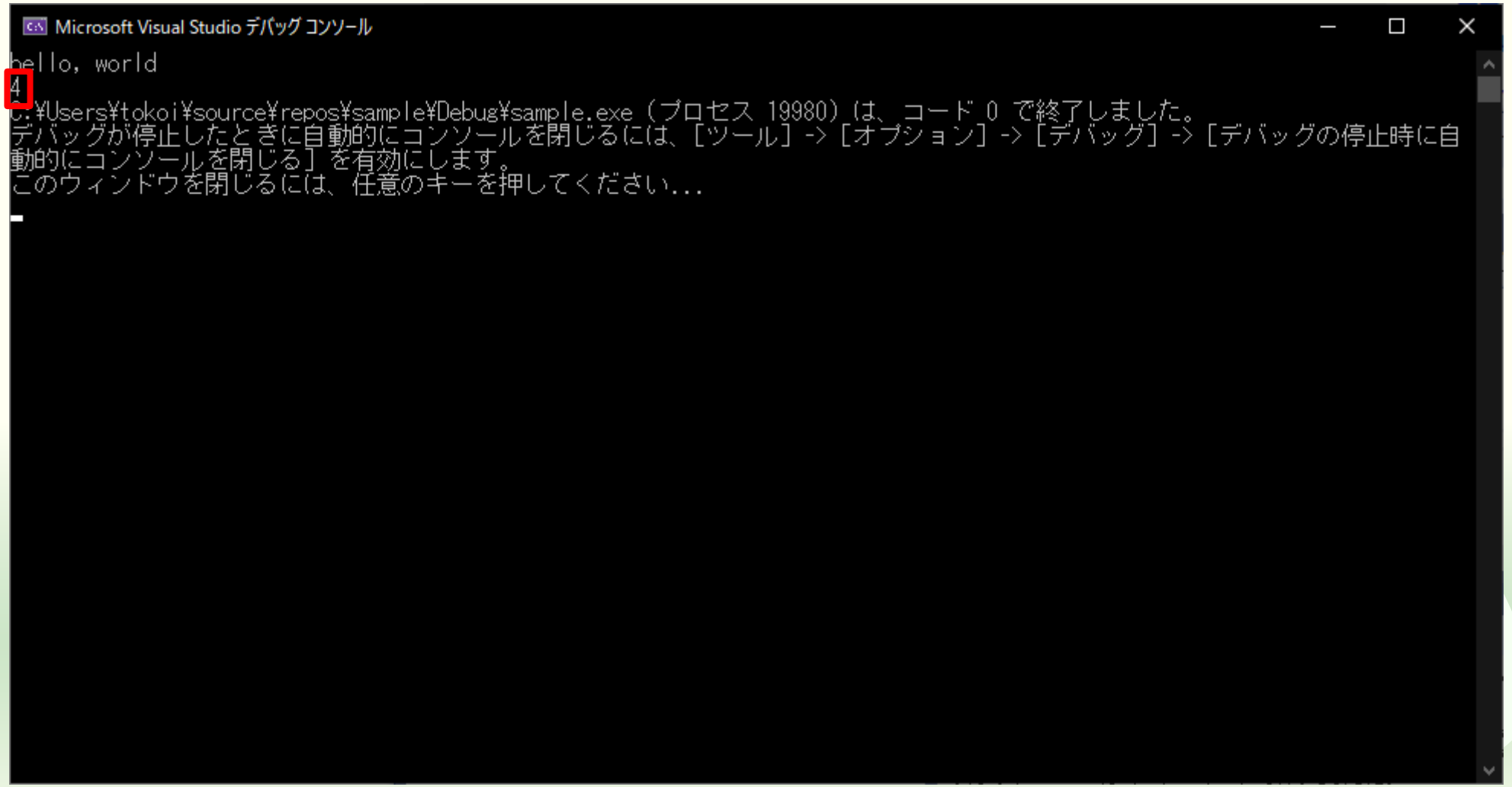


y をコンソールに出力する

```
#include <iostream>

double f(double x)
{
  return x * x;
}

int main()
{
  double y;

  y = f(2.0);
  std::cout << "hello, world¥n" << y;
}
```

- マーカー部分を追加する
  - 「std::cout << "..." 」全体も std::cout と同じ機能を持つ
  - したがって std::cout << "..." << "..." のように続けて書ける
  - std::cout << y とすると変数 y の値を文字に直してコンソールに出力する
  - したがって std::cout << "..." << y とすれば "..." の後ろに y の値を文字直して出力される

hello, world の次に 4 が出力される

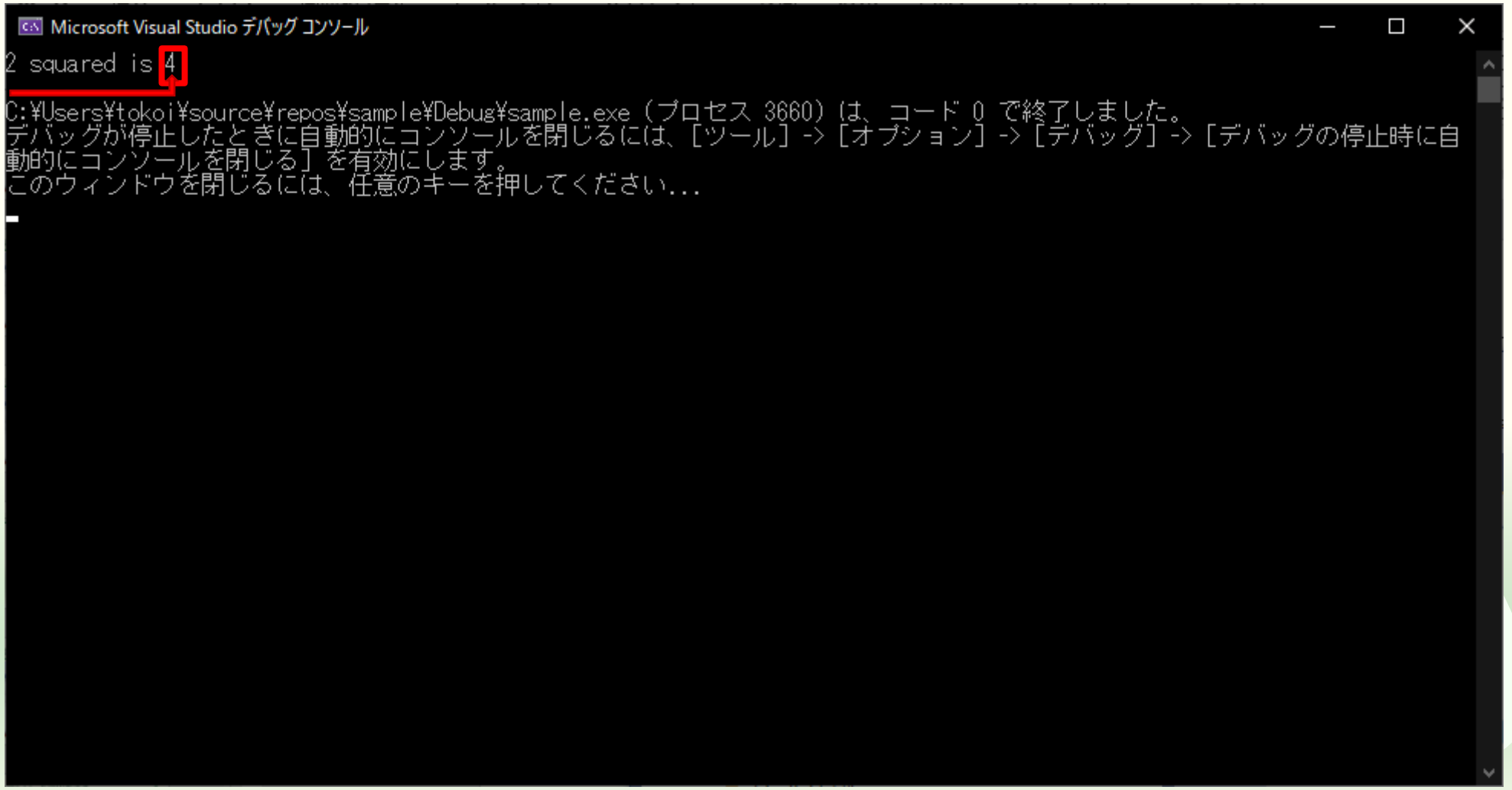


改行文字を移動する

```
#include <iostream>

double f(double x)
{
  return x * x;
}

int main()
{
  double y;

  y = f(2.0);
  std::cout << "2 squared is " << y << "¥n";
}
```

- マーカー部分を変更する
  - ¥n は改行文字
    - ¥はフォントや文字コードの環境によっては＼と表示される
    - 英語圏で用いられていた文字集合（ASCIIコード）の字形は＼だったが日本の文字集合の規格 JIS X 0201 では字形を¥にした

改行位置が移動する


Microsoft Visual Studio デバッグ コンソール
2 squared is 4
C:¥Users¥tokoi¥source¥repos¥sample¥Debug¥sample.exe (プロセス 3660) は、コード 0 で終了しました。
デバッグが停止したときに自動的にコンソールを閉じるには、[ツール] -> [オプション] -> [デバッグ] -> [デバッグの停止時に自動的にコンソールを閉じる] を有効にします。
このウィンドウを閉じるには、任意のキーを押してください...

ソフトウェアを作るのは色々大変

ちゃんとしたソフトウェアを作るなら道具から揃えよう

メディアを扱うプログラミング

- プログラミング言語自体は**メディアを扱う機能がない**
  - オペレーティングシステムの機能を呼び出す
    - 音声，音楽，画像，映像，グラフィックス
    - いろんなことができるが複雑で手間がかかることが多い
  - ライブラリを組み合わせる
    - 音声入出力，画像入出力，映像入出力，3Dモデル入力
    - 音声処理，画像処理，映像処理，グラフィックス処理
    - 物理シミュレーション，フォント処理，…
    - 一通り揃えるのは大変

ミドルウェアを利用する

- 特定用途向けのソフトウェア開発環境のパッケージ
  - シーングラフ／レンダリングエンジン
    - OpenSceneGraph, OGRE, Delta3D, …
  - ツールキット
    - openFrameworks, Cinder, P5.js, Three.js, freeglut, GLFW, FLTK, Qt, …
  - プログラミング環境
    - Max/MSP, PureData, SuperCollider, vvvv, TouchDesigner, Processing, …
  - ゲームエンジン
    - Unity, Unreal Engine, CRYENGINE, Lumberyard, OROCHI4, …
    - Irrlicht Engine, Armory Engine, Godot, Xenko, …

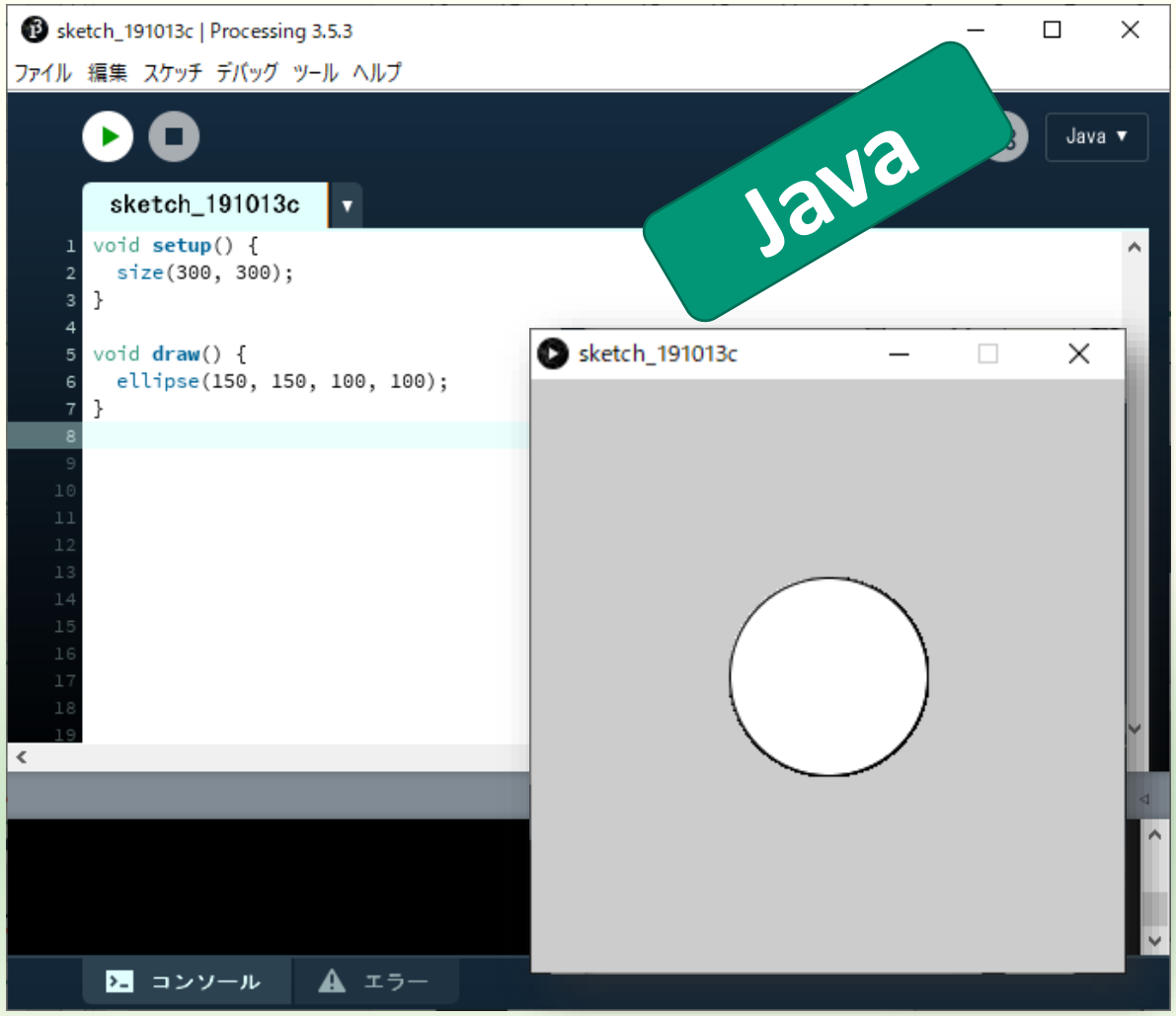
クリエーティブコーディング

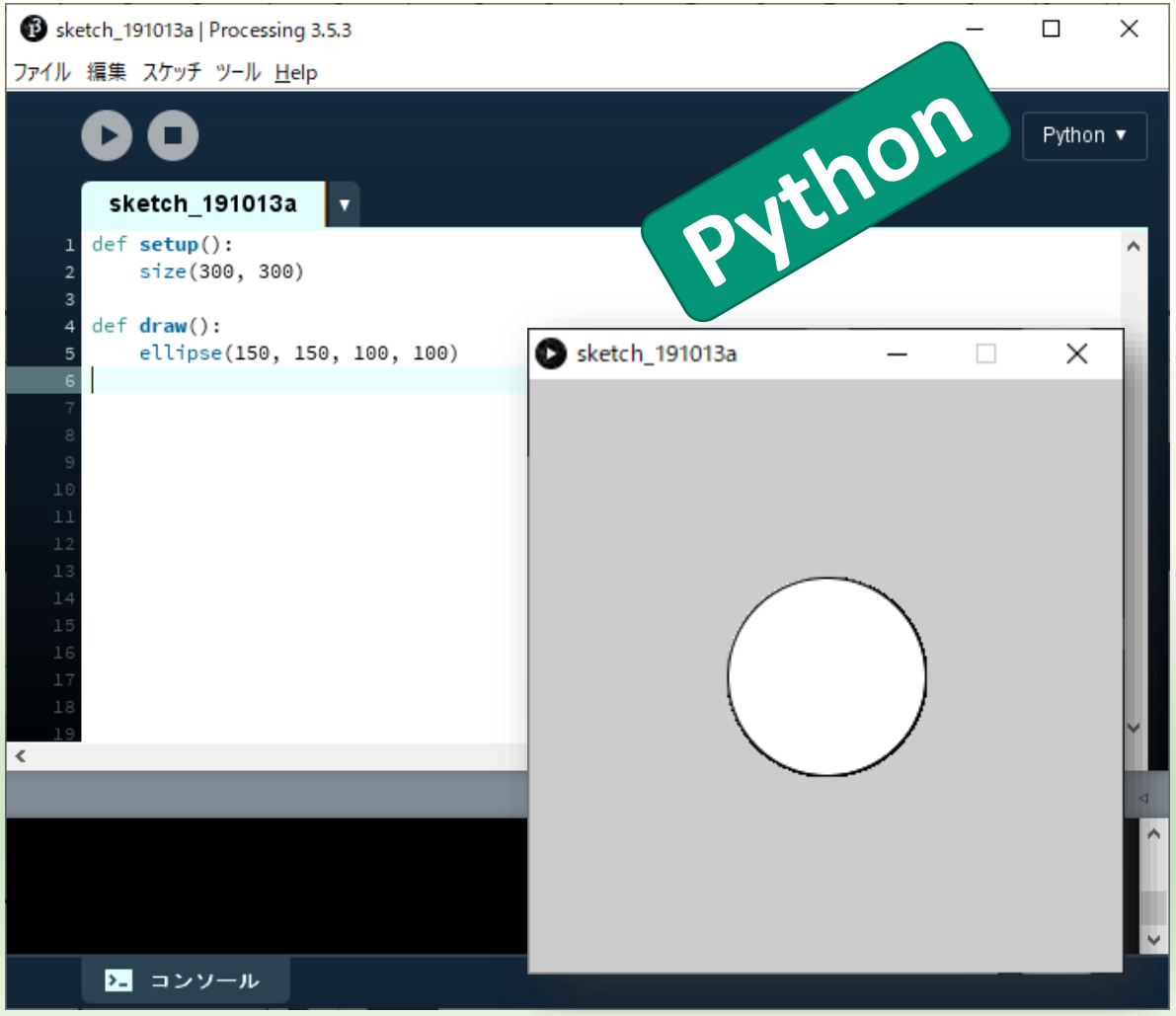
- 映像・音響による**表現のための**プログラミング
  - メディアアート，インタラクティブアート
    - “メディアアートの教科書” (多摩美術大学)
    - 関連企業
      - teamLab (チームラボ)
      - Rhizomatiks (ライゾマティクス)
      - Takram (タクラム)
      - MontBlanc Pictures (モンブラン・ピクチャーズ)
      - THE EUGENE Studio (ザ・ユージーン・スタジオ)
      - 株式会社 白

Processing

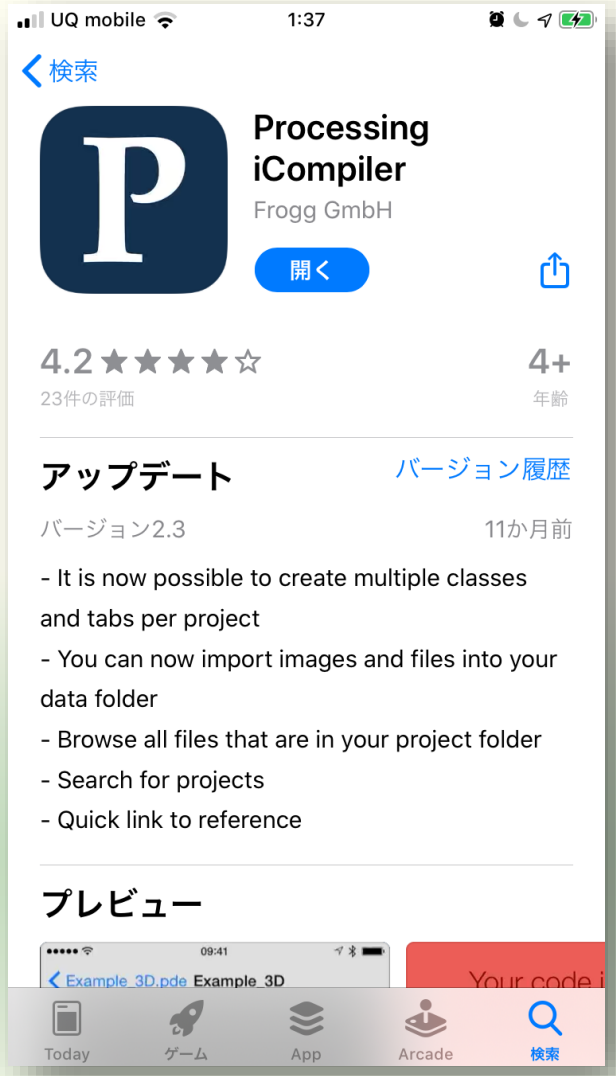
- プログラミングによる創作を行うためのツール
  - 誰でも簡単に視覚表現を行うプログラムを作れる
  - MIT Media Lab. の John Maeda の下で “Design by Numbers” の開発に携わった二人の大学院生 Casey Reas と Ben Fry によりオープンソースプロジェクトとして開発
- Java で実装されている
  - プログラムは Java でも Python でも書ける
- プログラミングで絵を描く

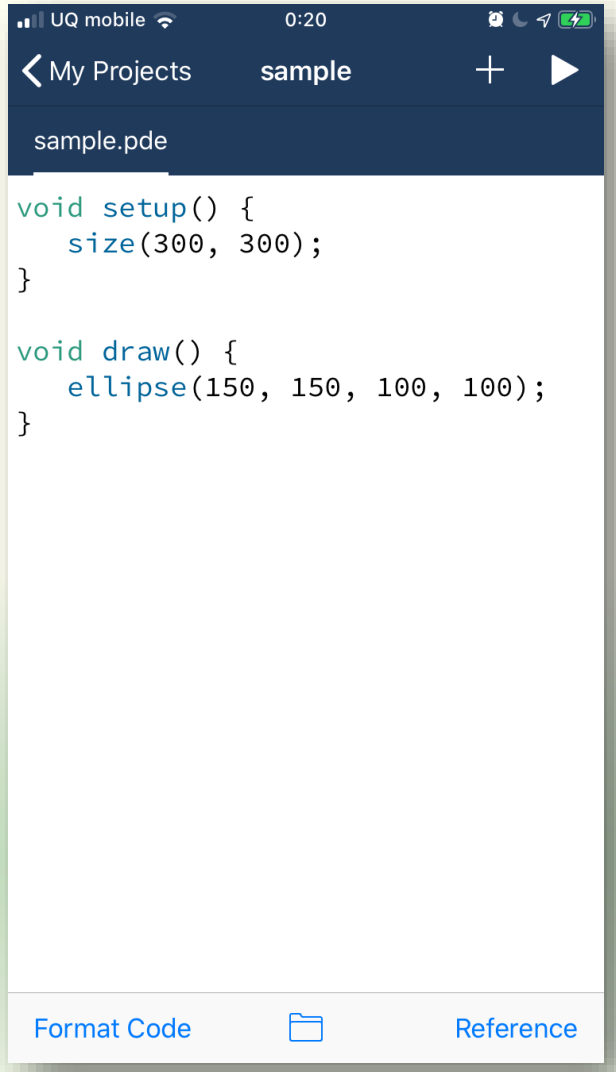
Processing の sketchbook (開発環境)

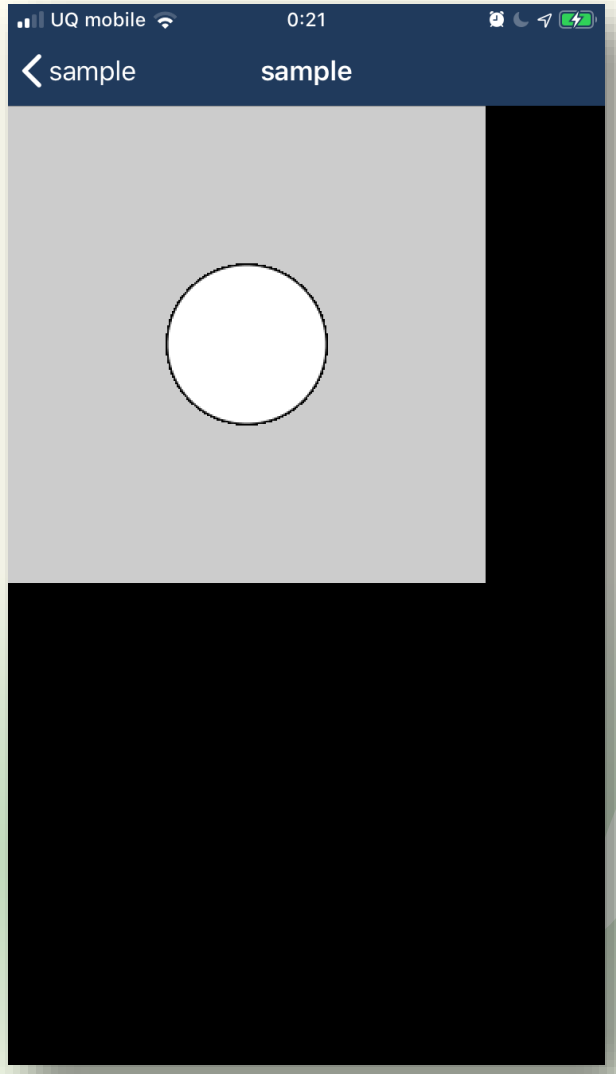




iPhone でも動く


UQ mobile
1:37
検索
Processing
iCompiler
Frogg GmbH
開く
4.2
23件の評価
4+
年齢
アップデート
バージョン履歴
バージョン2.3
11か月前
- It is now possible to create multiple classes and tabs per project
- You can now import images and files into your data folder
- Browse all files that are in your project folder
- Search for projects
- Quick link to reference
プレビュー
Today
ゲーム
App
Arcade
検索


UQ mobile
0:20
My Projects
sample
sample.pde
void setup() {
size(300, 300);
}
void draw() {
ellipse(150, 150, 100, 100);
}
Format Code
Reference


UQ mobile
0:21
sample
sample

しかし，この演習では openFrameworks を使います

メディアプログラミング演習

openFrameworks

- クリエーティブコーディングのための **C++ 言語による**オープンソースのフレームワーク
  - 様々なライブラリを統合
    - グラフィクス：OpenGL, GLEW, GLUT, libtess2, cairo
    - オーディオ：rtAudio, PortAudio, OpenAL, Kiss FFT または FMOD
    - フォント：FreeType
    - イメージの読込と保存：FreeImage
    - 動画の再生と取込：Quicktime, GStreamer, videoInput
    - 様々なユーティリティー：Poco
    - コンピュータビジョン：OpenCV
    - 3Dモデルの読み込み：Assimp

実は何を使うかすごく悩んだ

- Processing
  - Java ベースで簡単・処理系も軽い・Python でも書ける
- Python cgkit
  - Python で 3D CG を扱うパッケージやプラグインを集めた
- Three.js
  - JavaScript で WebGL を使うグラフィックスライブラリ
- Unity
  - Unreal Engine と並んでゲーム開発ミドルウェアの標準

目標

プログラミングを
知ること

アプリケーション
プログラムを
作ること

openFrameworks の特徴

- openFrameworks は C++ 用のツールキット
  - 既存のライブラリや SDK がそのまま使える
    - openFrameworks 自体はそれらをくっつける**糊 (glue)** の役割
  - 開発環境には一般のもの (Visual Studio, Xcode など) を使う
    - Processing は Processing 自体が開発環境（なので手軽ではある）
- 同様なものとして，他に Cinder がある
  - より新しい機能を使って作られている
    - 他に C# 用の OpenTK というツールキットがある

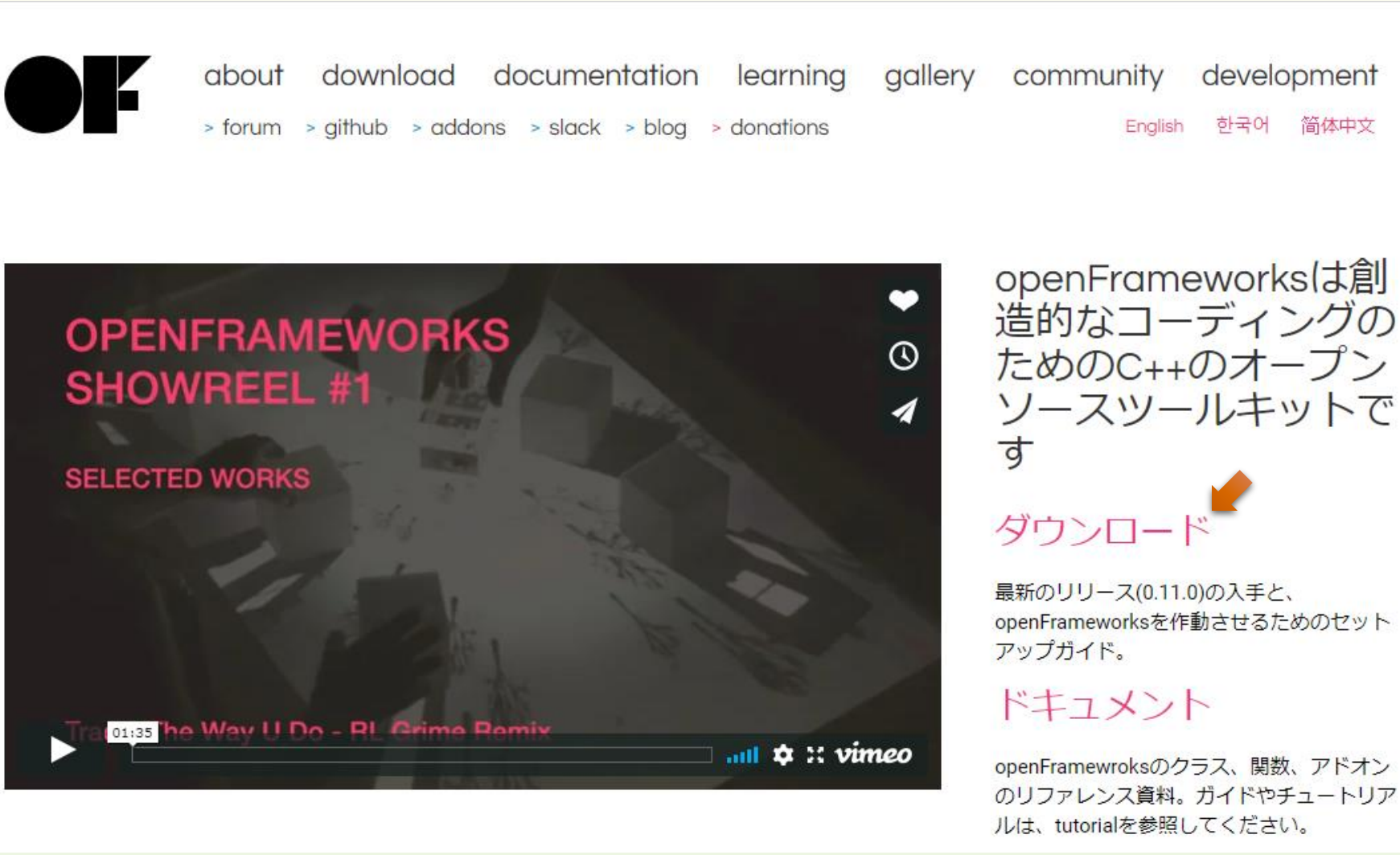
なぜ openFrameworks か

- 一般的な開発環境に追加して使える
  - Visual Studio, Xcode, eclipse など
- 他のライブラリや SDK と組み合わせやすい
  - デバイスの SDK は C や C++ で用意されていることが多い
  - 研究用としても使える？
- C++ ベースである
  - プログラミングの**初心者には向いていない**
  - まあ何とかなるでしょう？

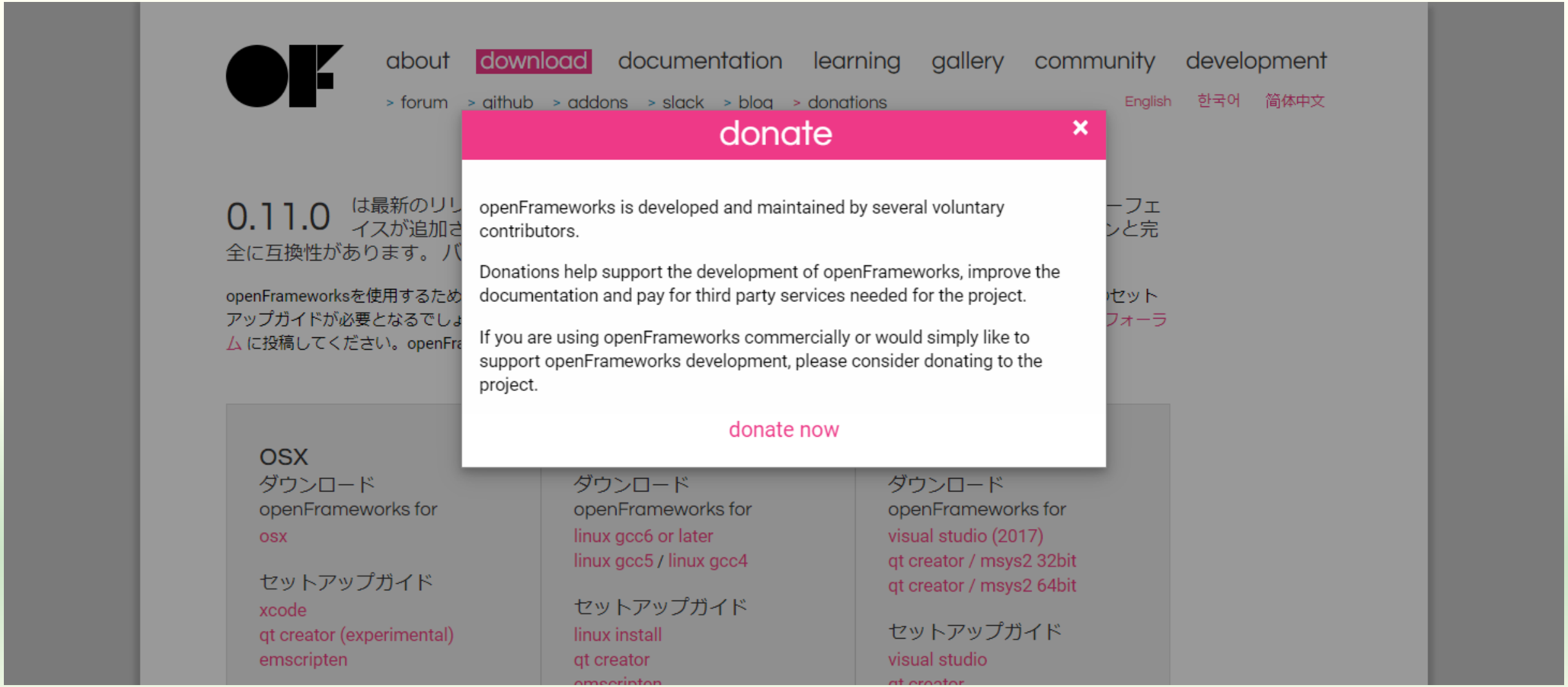
課題１－１

openFrameworks のパッケージのダウンロードと展開

https://openframeworks.cc/ja/ を開く



Download (寄付募集がポップアップする)



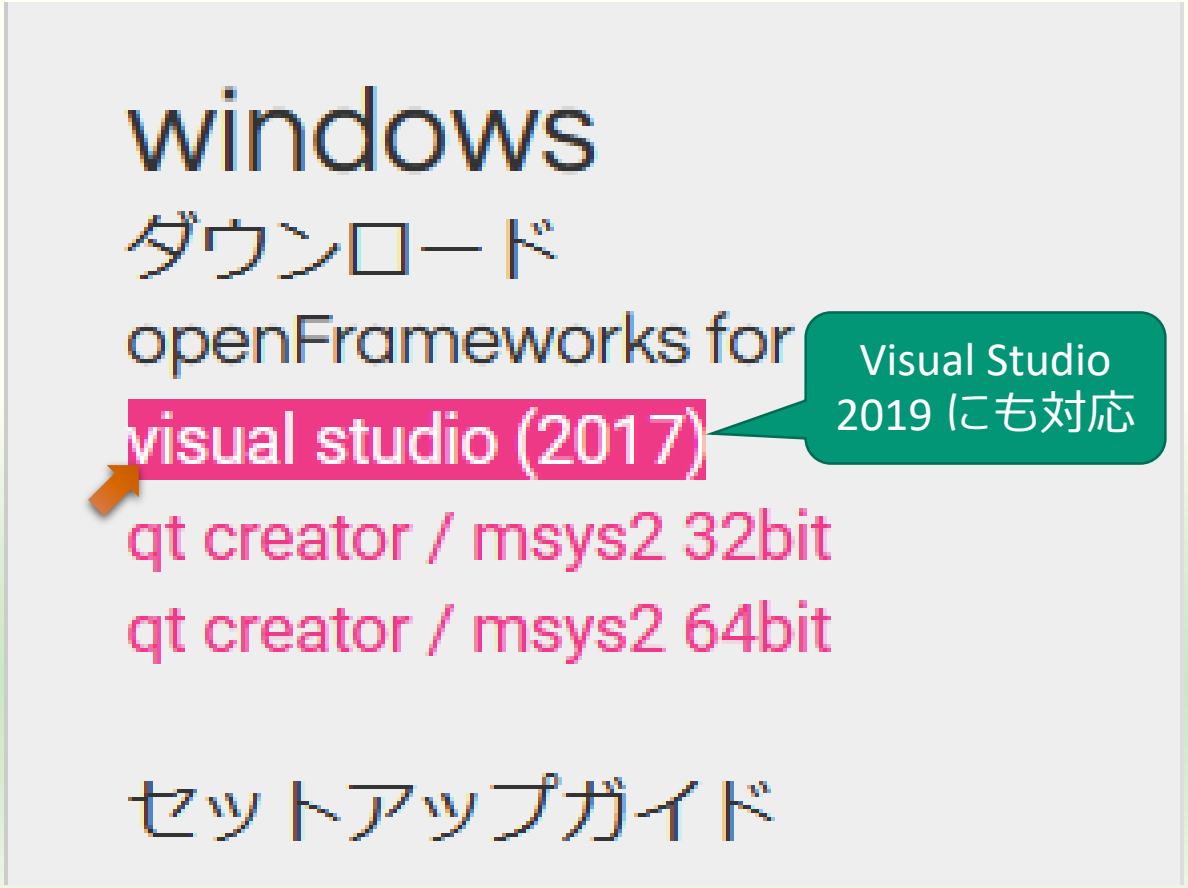
様々なプラットフォームのアプリが作れる

OSX / linux / windows に対応している

スマホや Raspberry Pi にも対応している

使用する PC に合ったものをダウンロード

Windows 版のダウンロード

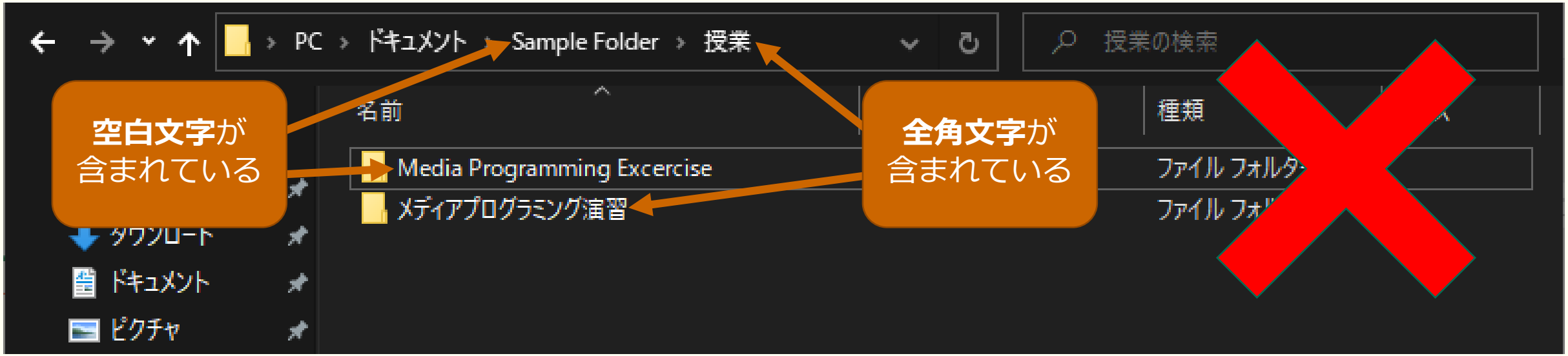


macOS 版のダウンロード

ダウンロードした ZIP ファイルを展開

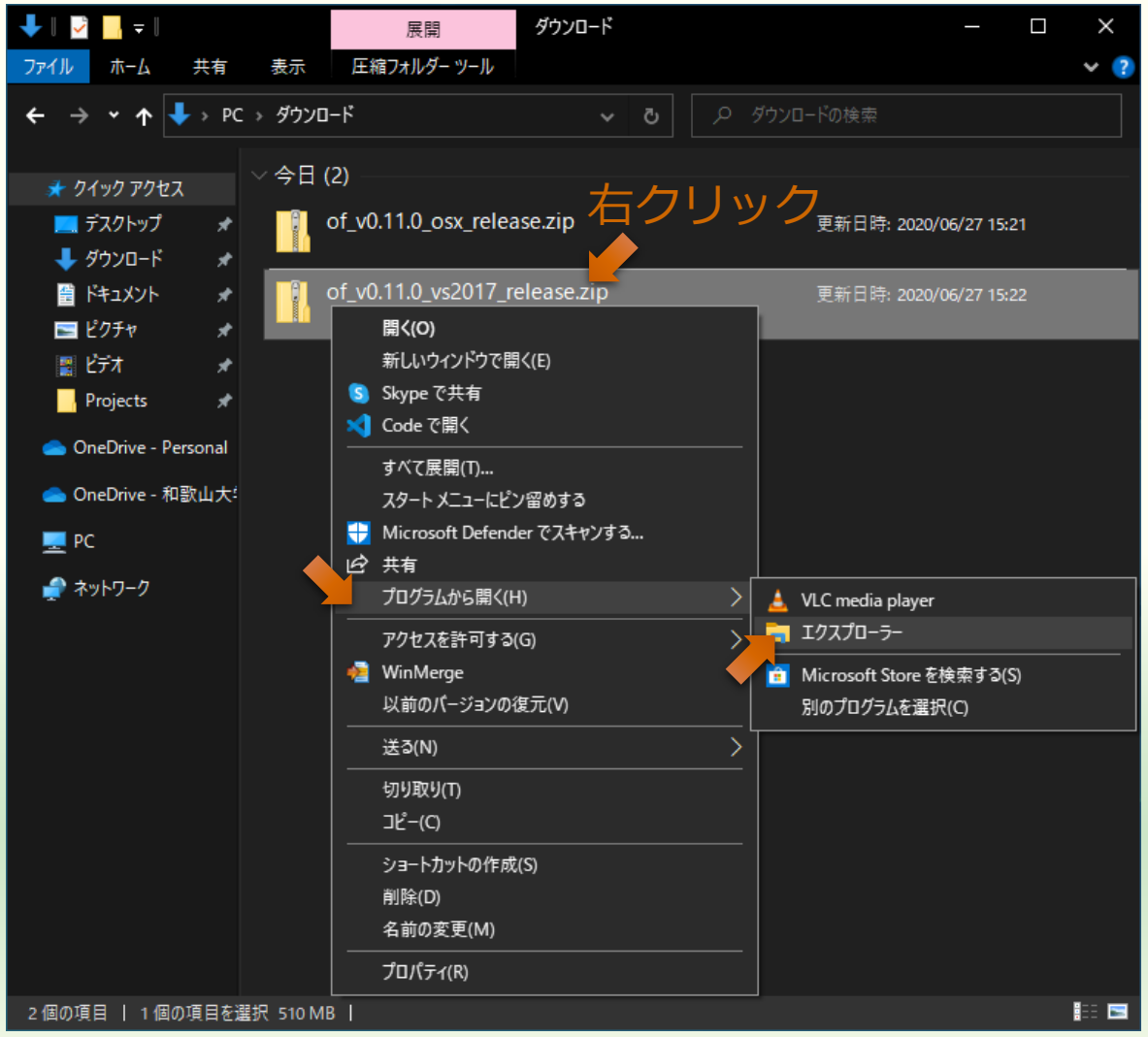
- 展開先のフォルダの条件
  - フォルダのパスに**空白文字が含まれていない**こと
    - ユーザ名に空白文字を含んでいると「ドキュメント」や「デスクトップ」のパスに空白文字が含まれることがある
  - フォルダのパスに**全角文字が含まれていない**こと
    - ユーザ名が日本語だと「ドキュメント」や「デスクトップ」のパスに全角文字が含まれることがある
- 個人所有の PC なら C: ドライブの直下 (C:¥) が確実
  - 条件を満たすことができるなら他の場所でも可

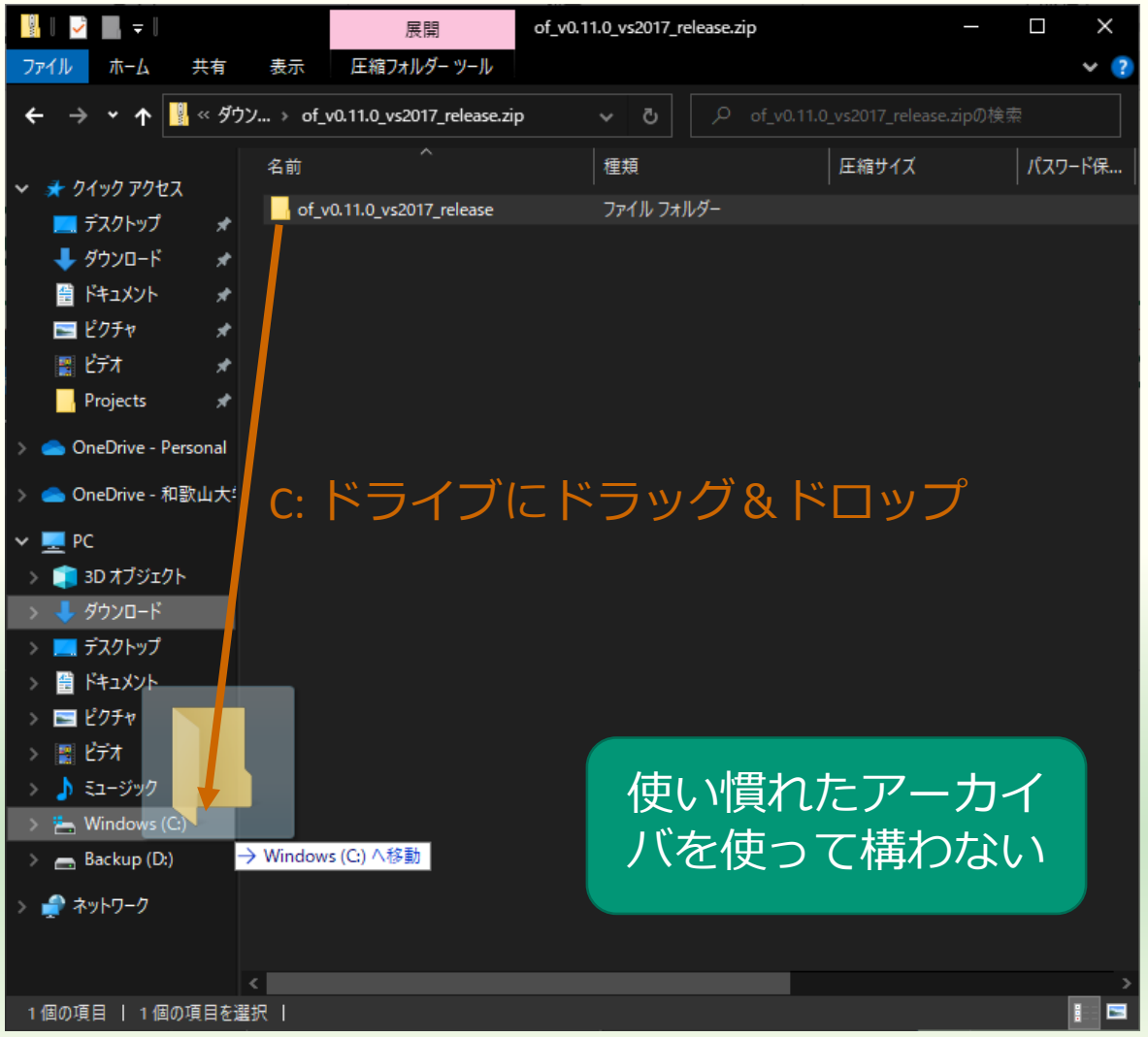
展開先として不適当なフォルダの例



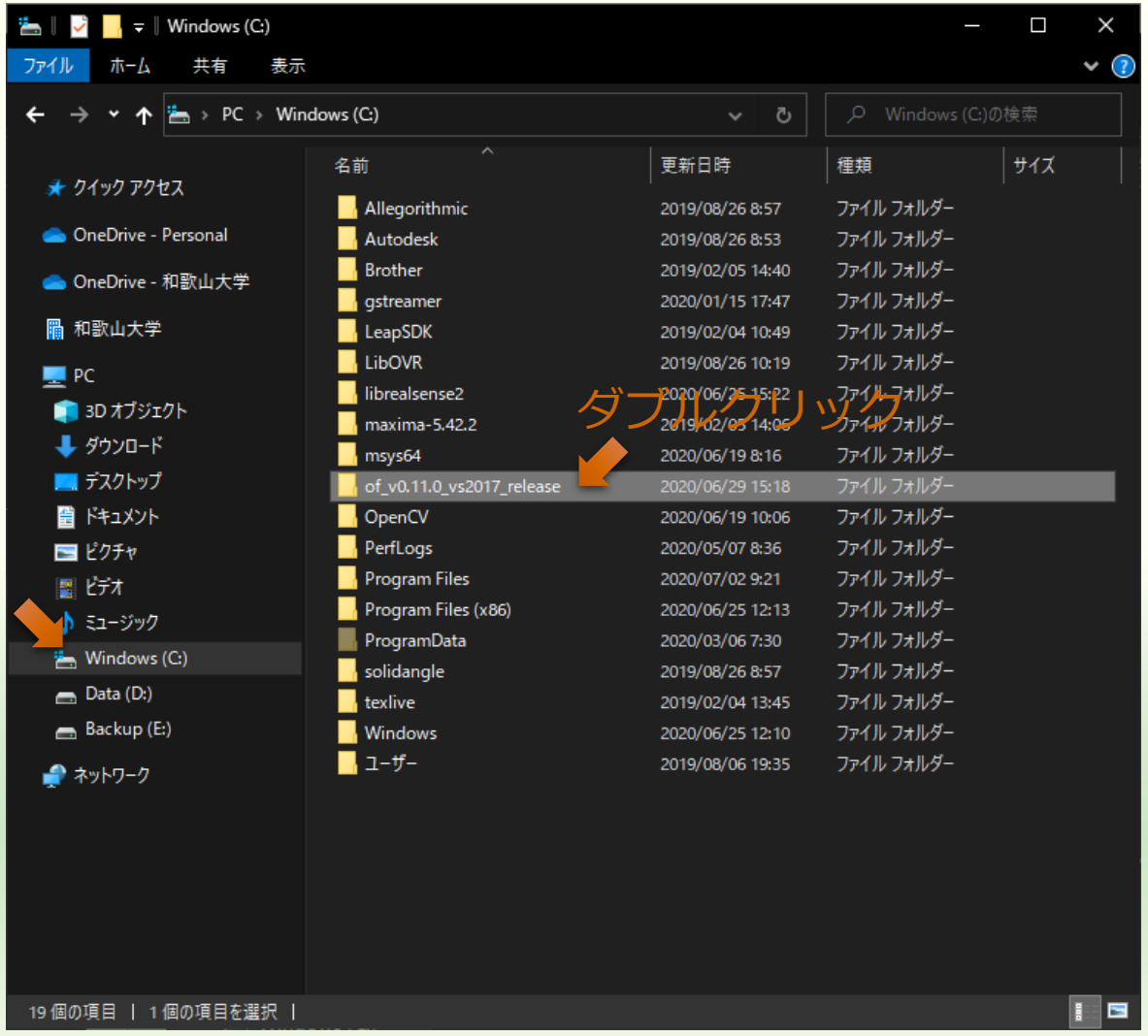
- “ドキュメント” は実体が “Documents” なので**可**
- “デスクトップ” も実体が “Desktop” なので**可**
- ユーザ名に**空白文字**を含む場合 (“Taro Yamada” など) **不可**
- ユーザ名に**全角文字**を含む場合 (“山田太郎” など) **不可**

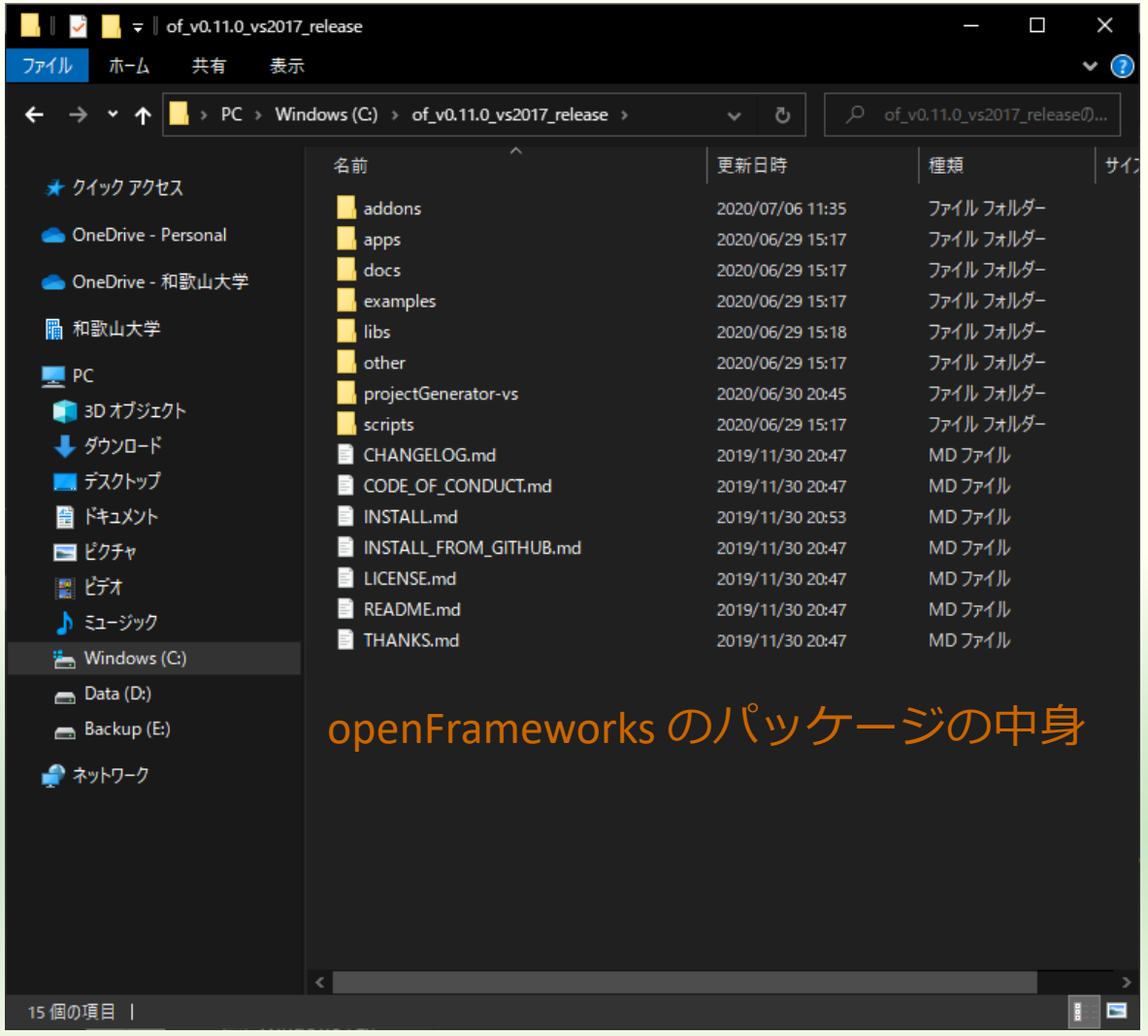
ZIP ファイルの中身を C:¥ にコピー





C: ドライブに展開したパッケージの中身

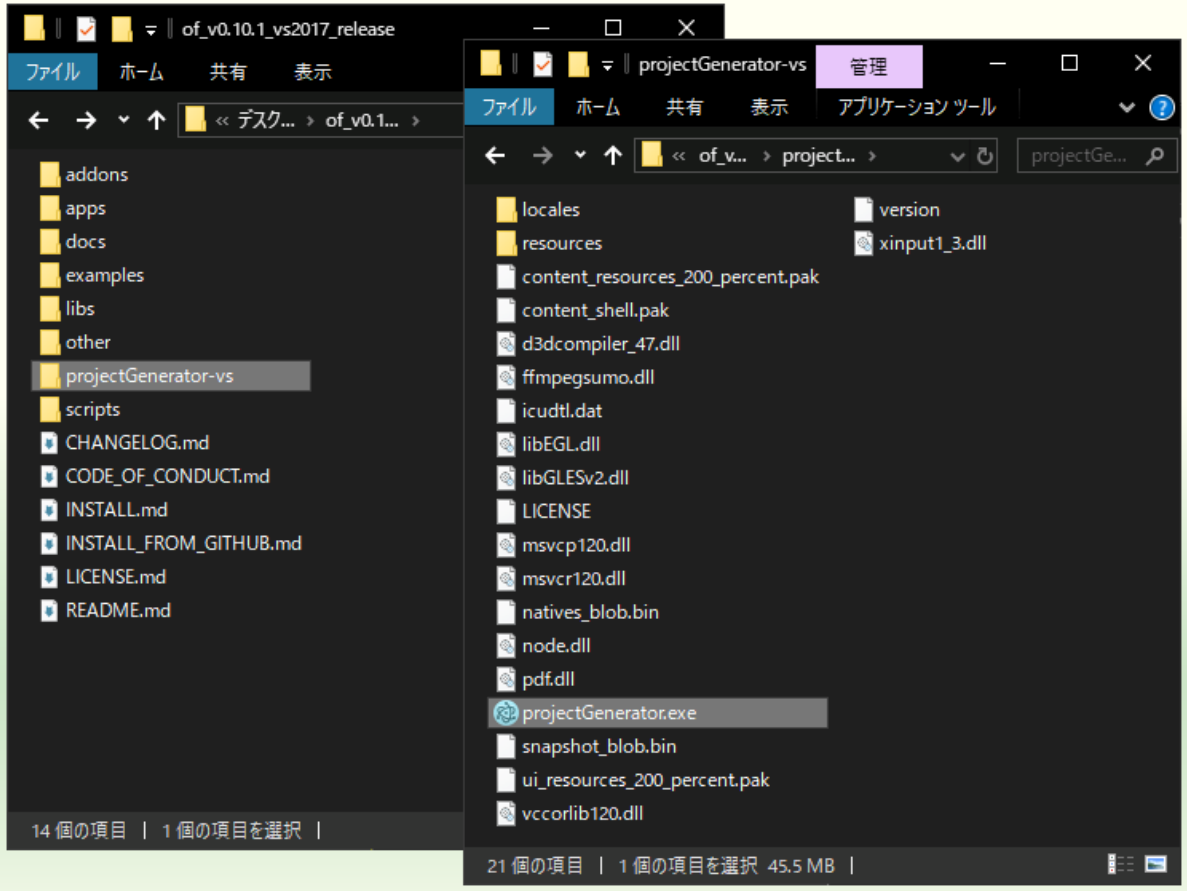




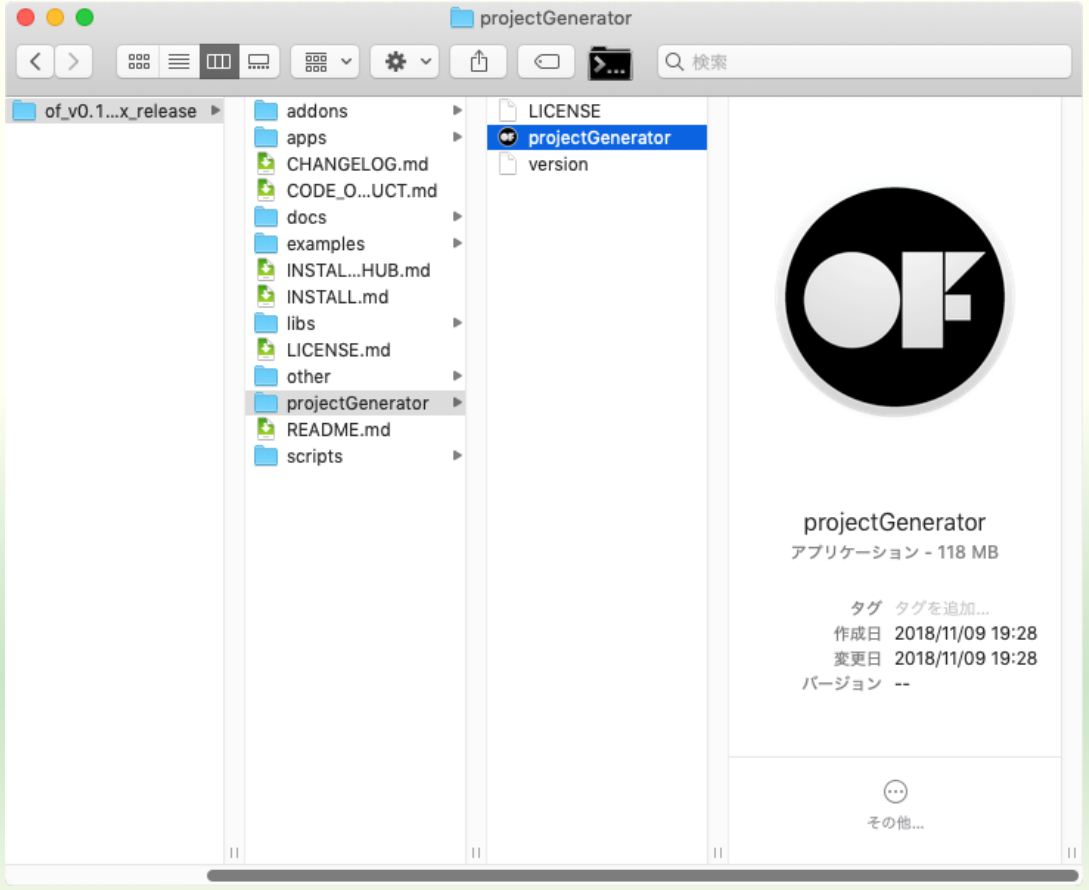
openFrameworks のパッケージの中身

パッケージの内容

windows 版のパッケージ



OSX 版のパッケージ



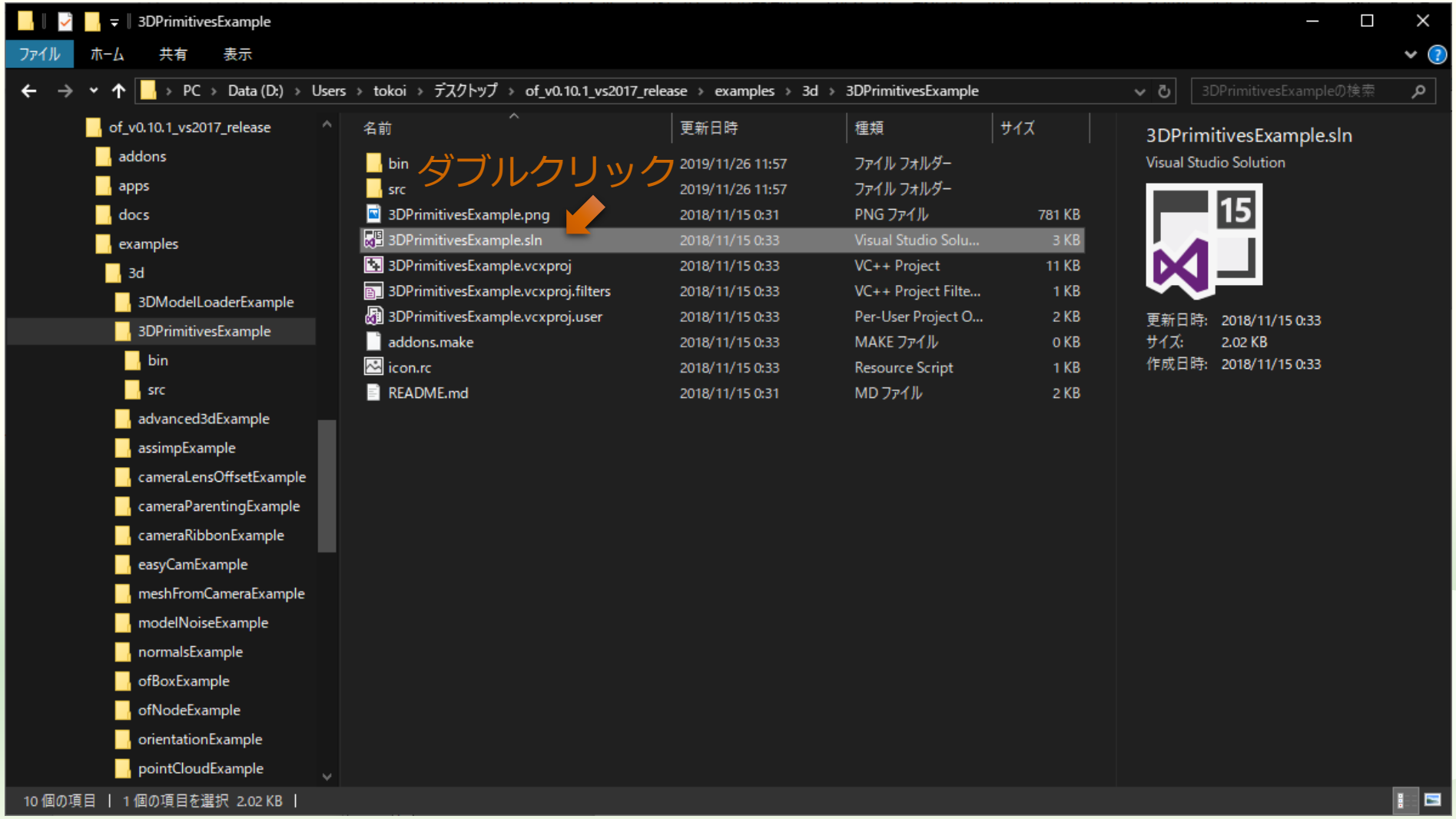
配置したパッケージはむやみに移動しない

- openFrameworks を使って作ったプログラム（プロジェクト）は openFrameworks のフォルダ内にある
  - 何かのプロジェクトをビルド後にパッケージ自体を別のところに移動すると以降ビルドに失敗するようになる
    - 最初にビルドしたときに作られる openFrameworks のライブラリファイルの場所を絶対パスで記録している

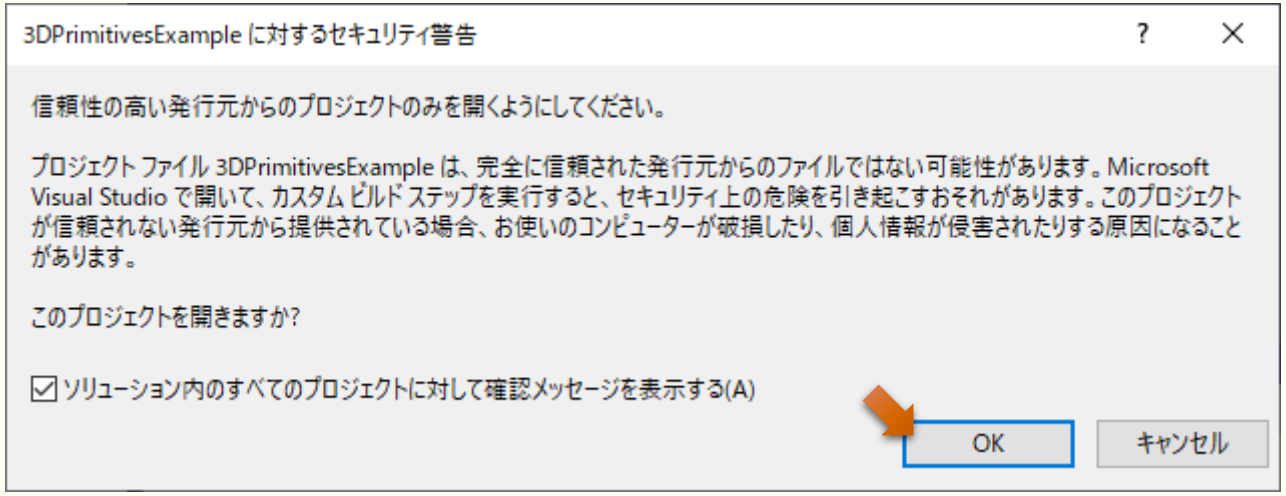
課題１－２

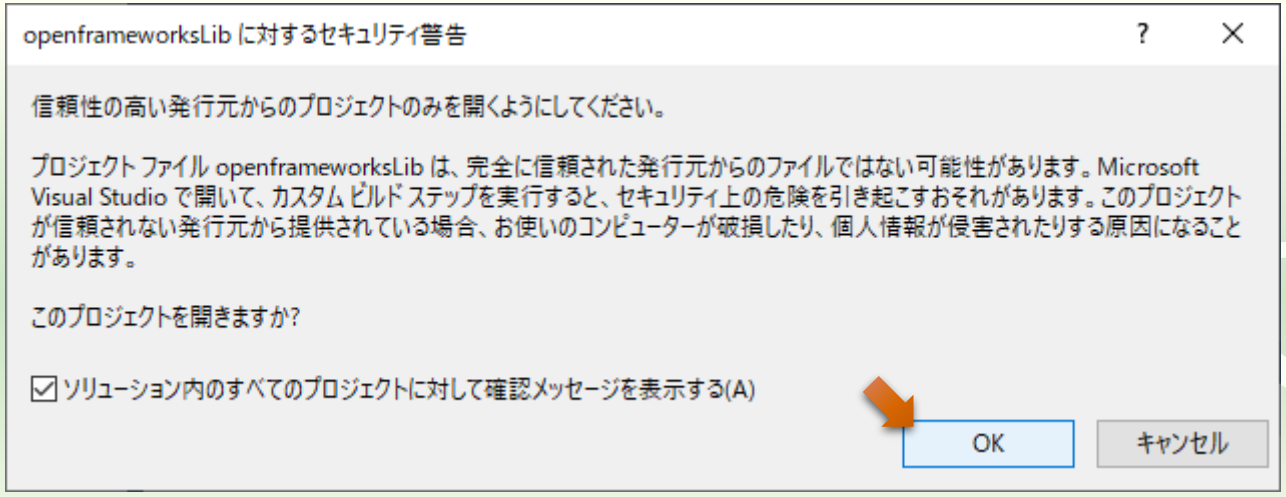
サンプルプロジェクトのビルド

(windows) 3DPrimitivesExample.sln

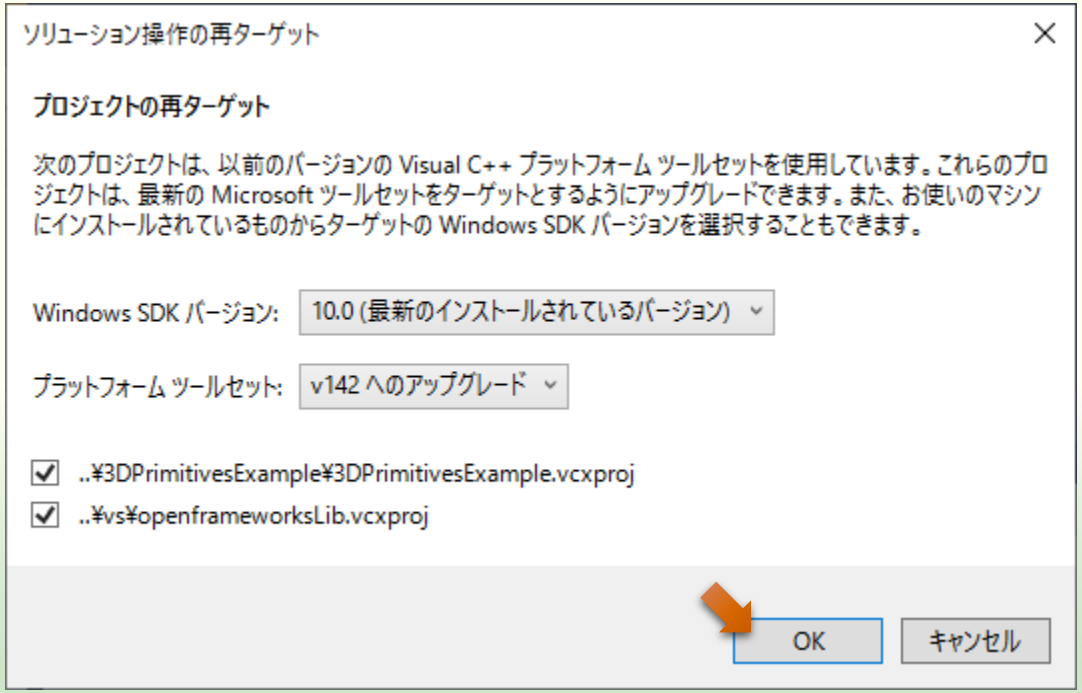


セキュリティ警告が出ても「OK」



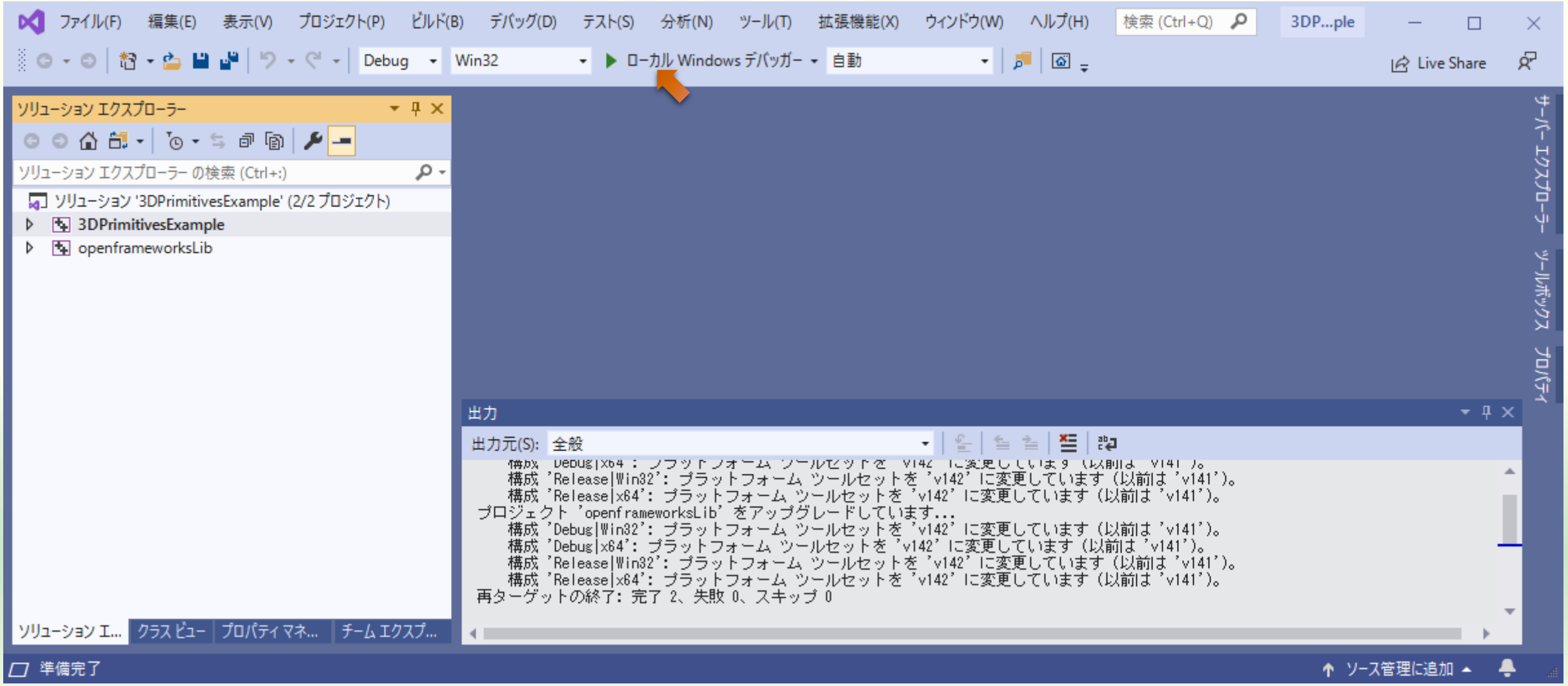


ソリューションの再ターゲット

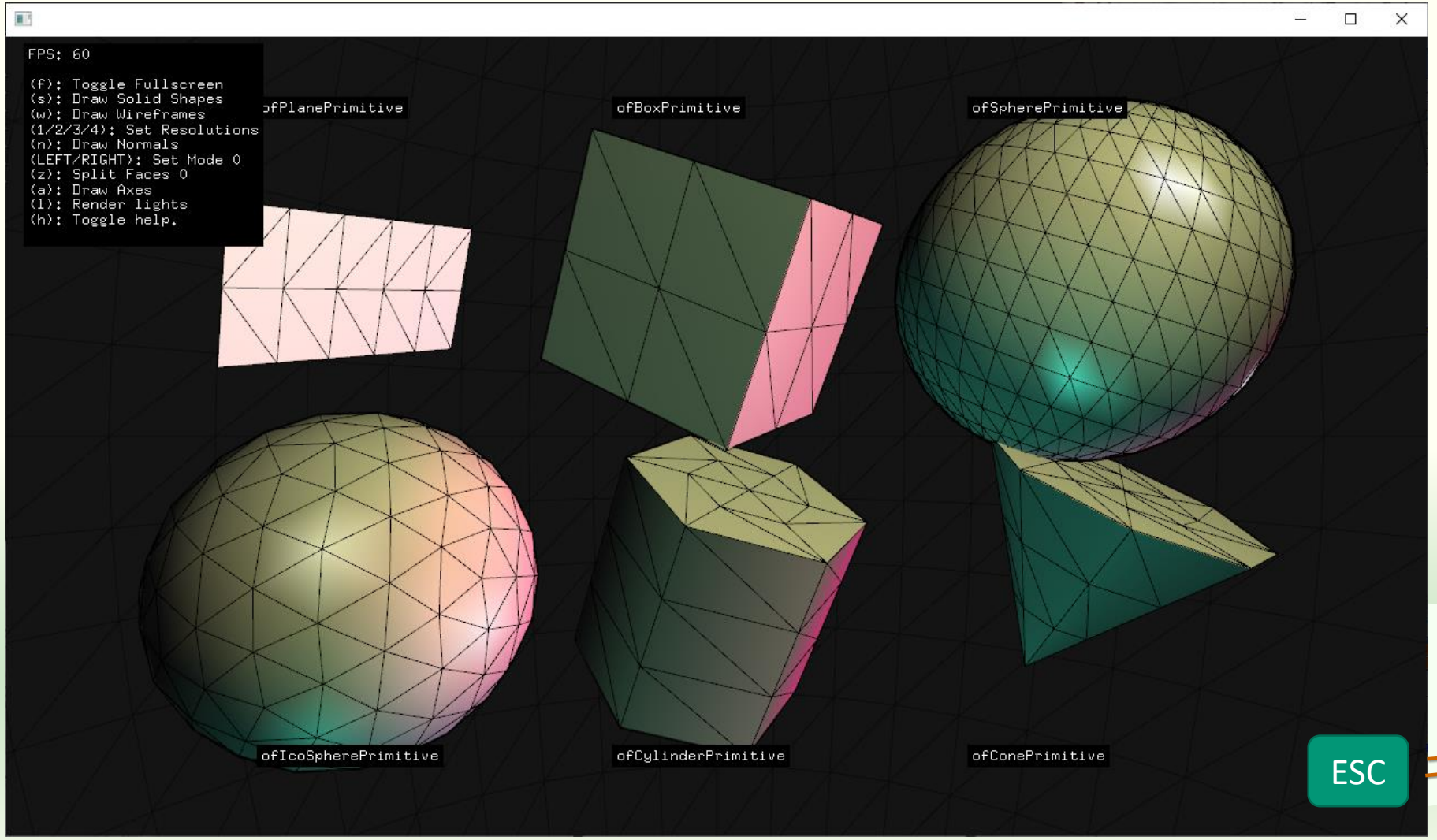


Visual Studio は頻繁に更新しているので皆さんがお使いの Visual Studio SDK のバージョンと合わない場合がある

(windows) デバッグを開始する

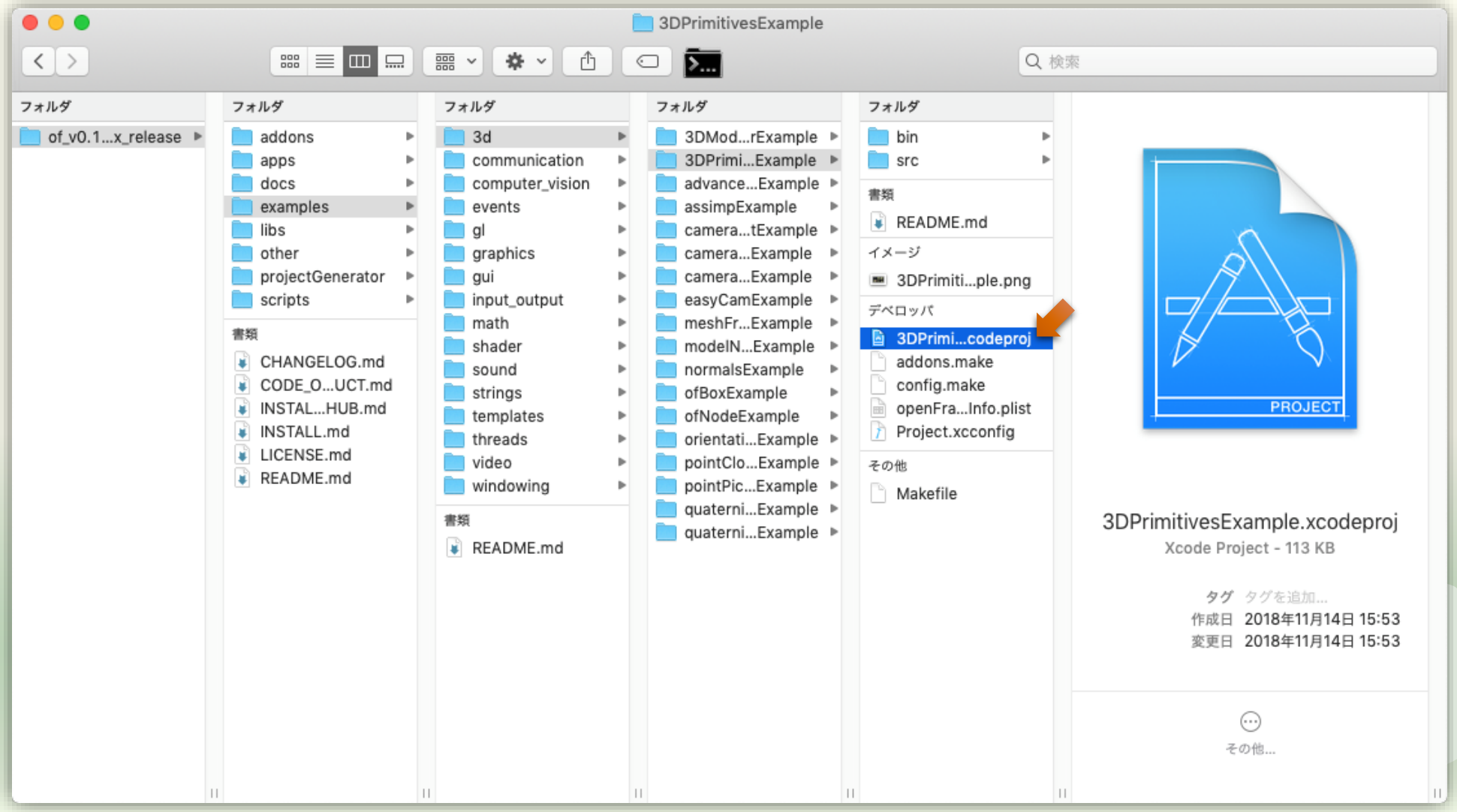


(windows) 実行中



ESC キーで終了

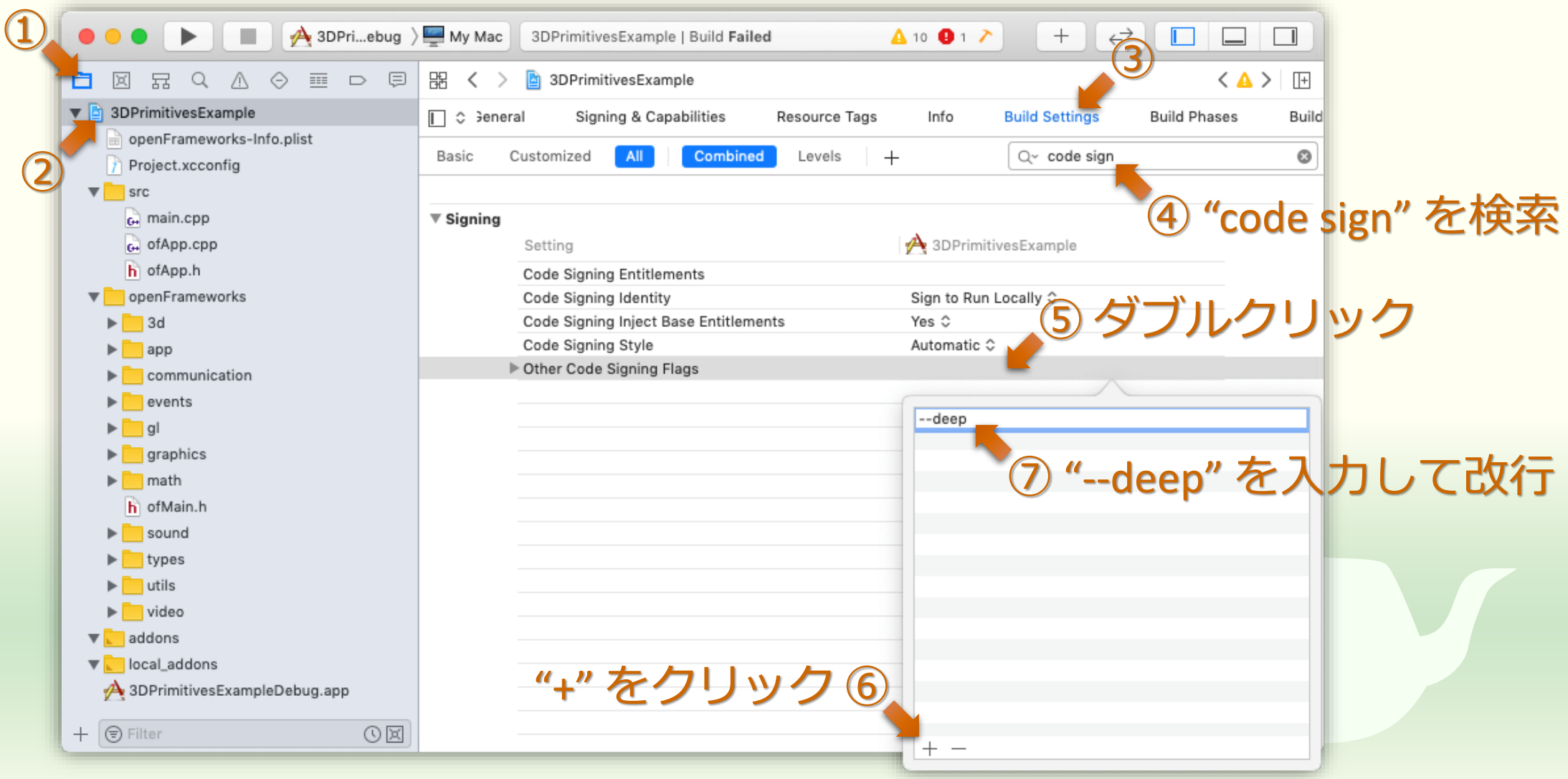
(macOS) 3DPrimitivesExample.xcodeproj



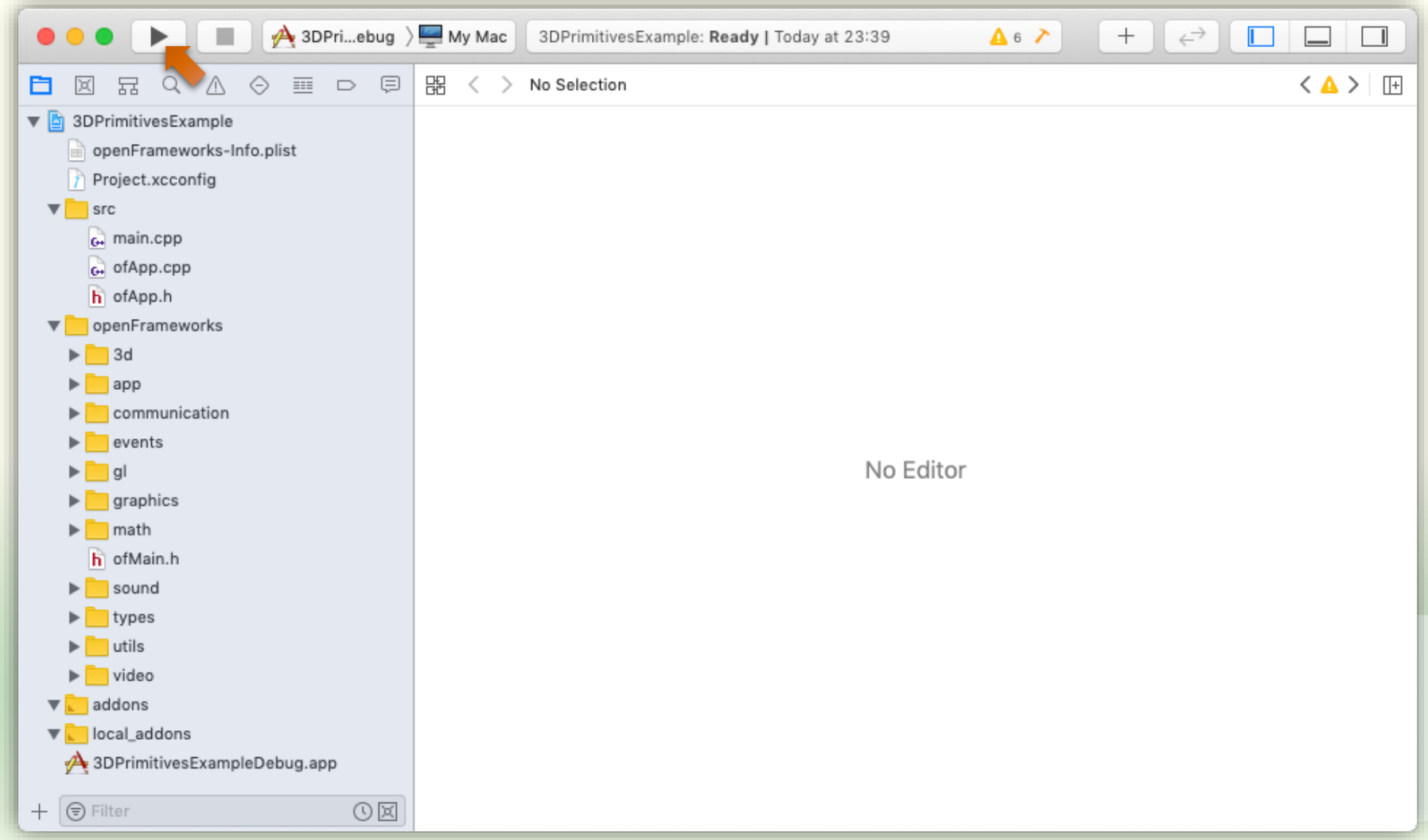
(macOS) “Open” する

- 「インターネットからダウンロードしてきたものだけど本当に開いてよいか」って聞かれるので “Open” をクリック

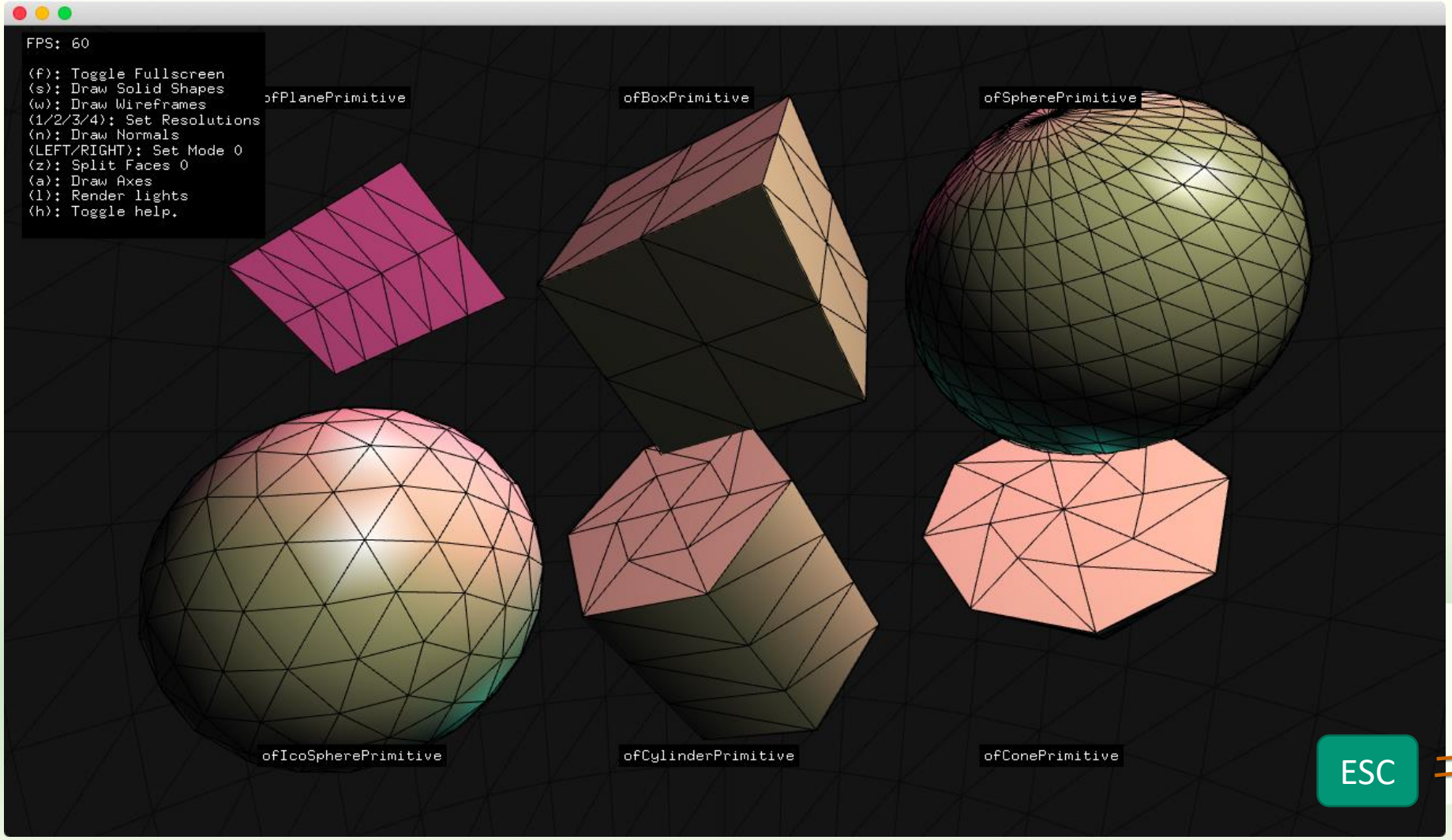
(macOS) Other Code Signing Flags に “--deep”



(macOS) “Run” する

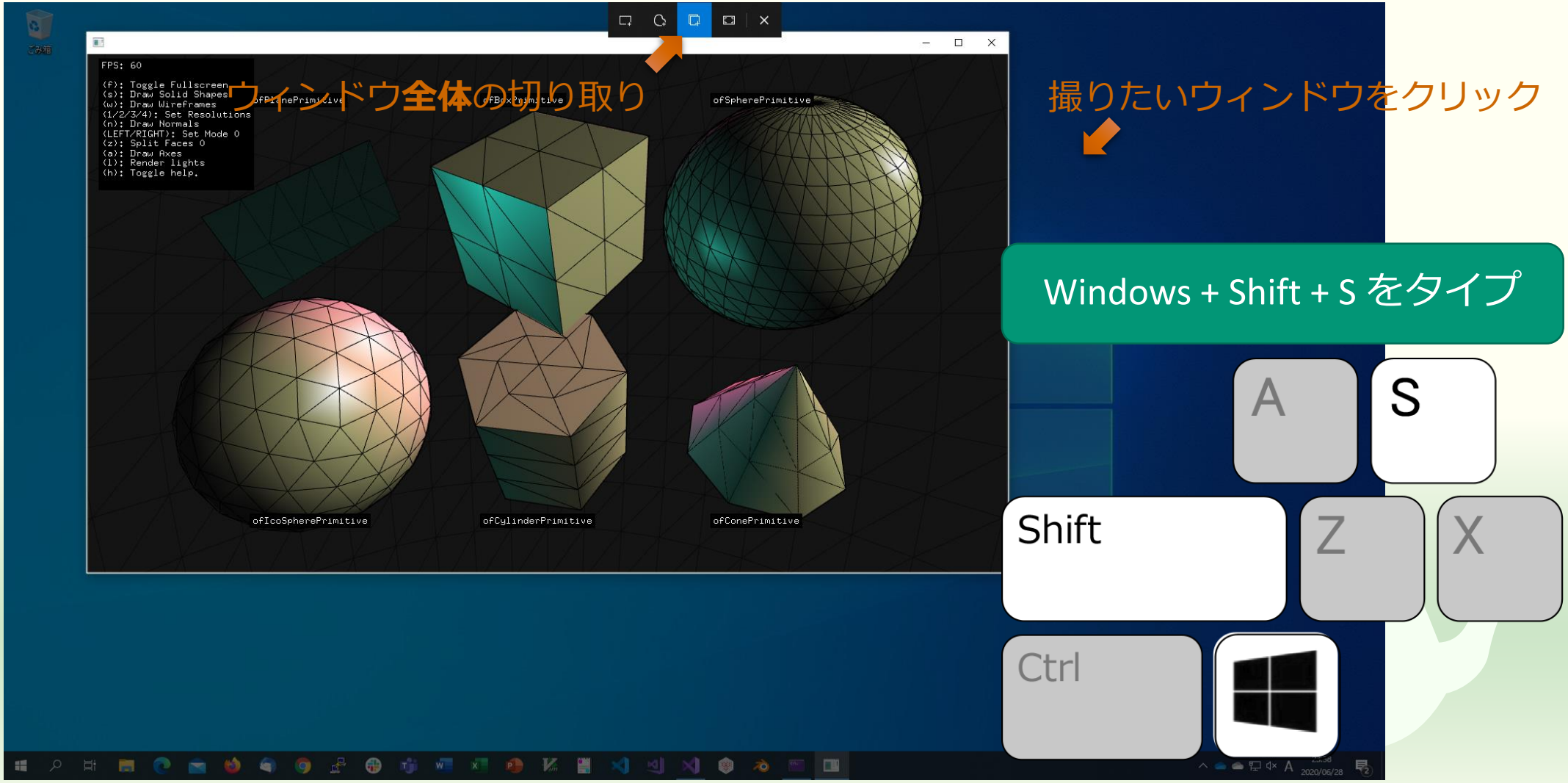

3DPri...ebug
My Mac
3DPrimitivesExample: Ready | Today at 23:39
No Selection
3DPrimitivesExample
openFrameworks-Info.plist
Project.xcconfig
src
main.cpp
ofApp.cpp
ofApp.h
openFrameworks
3d
app
communication
events
gl
graphics
math
ofMain.h
sound
types
utils
video
addons
local_addons
3DPrimitivesExampleDebug.app
Filter
No Editor

(macOS) 実行中



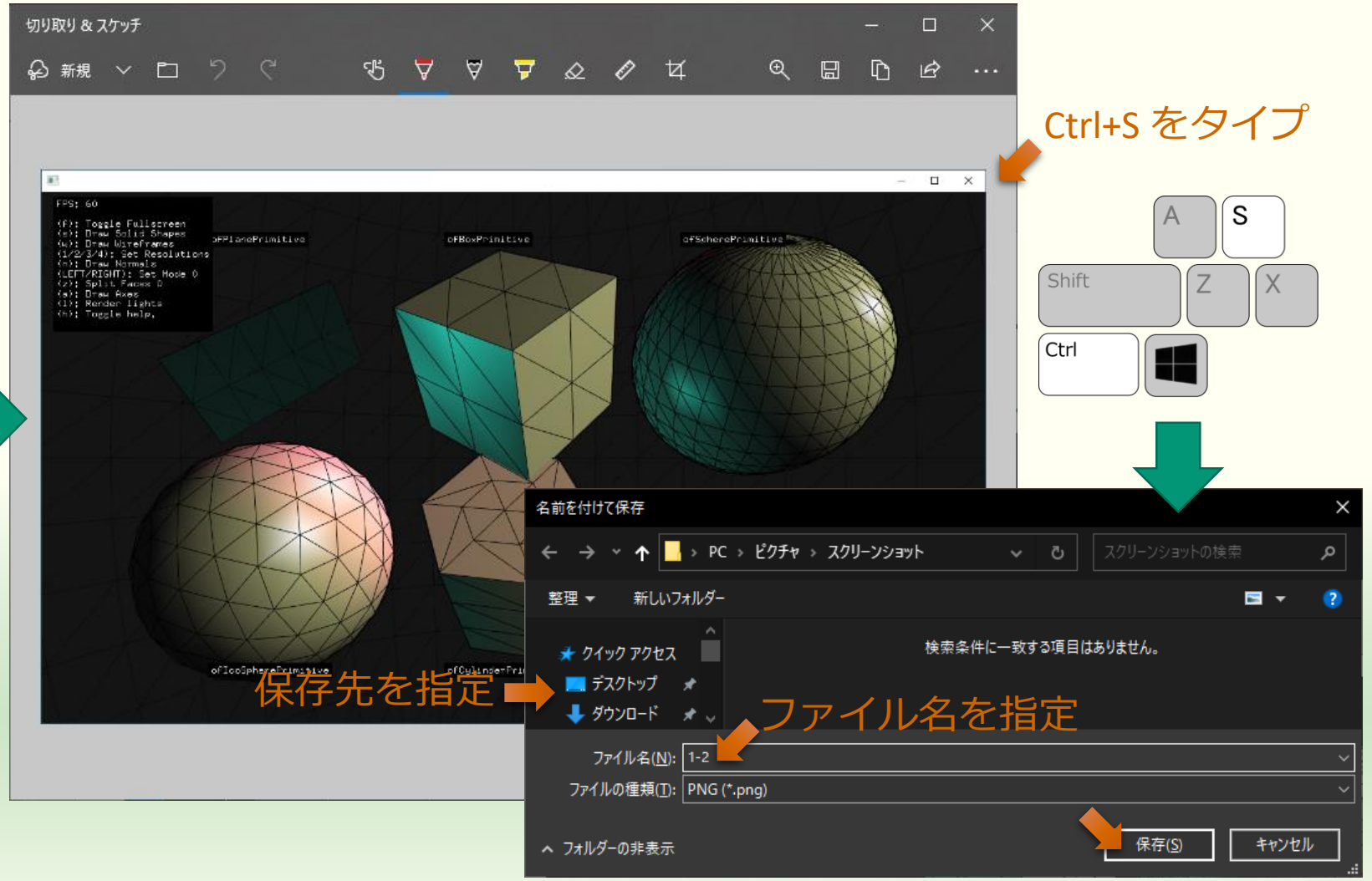
ESC キーで終了

実行結果のスクリーンショットを撮る



“1-2.png” というファイル名で保存

通知をクリック



スクリーンショットのアップロード

- 実行結果のスクリーンショットを **1-2.png** というファイル名で Moodle の第 1 回課題にアップロードしてください

課題１－３

他のサンプルプロジェクトのビルド

他のサンプルプロジェクトをビルドする

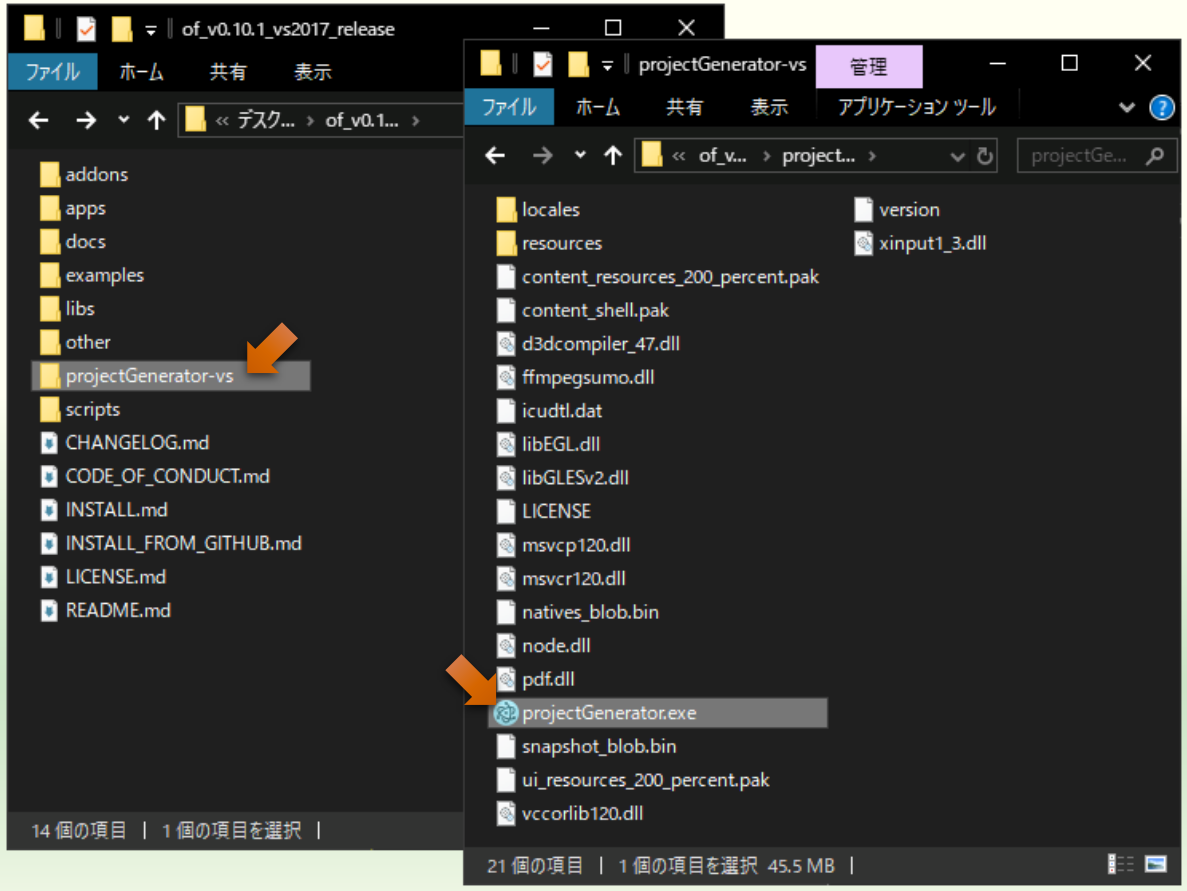
- example のサンプルプロジェクトは openFrameworks で何か作ろうとしたときに必ず参考になります
- 他のサンプルプロジェクトのプロジェクト名を一通り見てください
  - 何をするものか考えてください
- これらの中から 3 つ以上のサンプルプロジェクトをビルド・実行してみてください
  - カメラやマイクが必要なものもあります

課題１－４

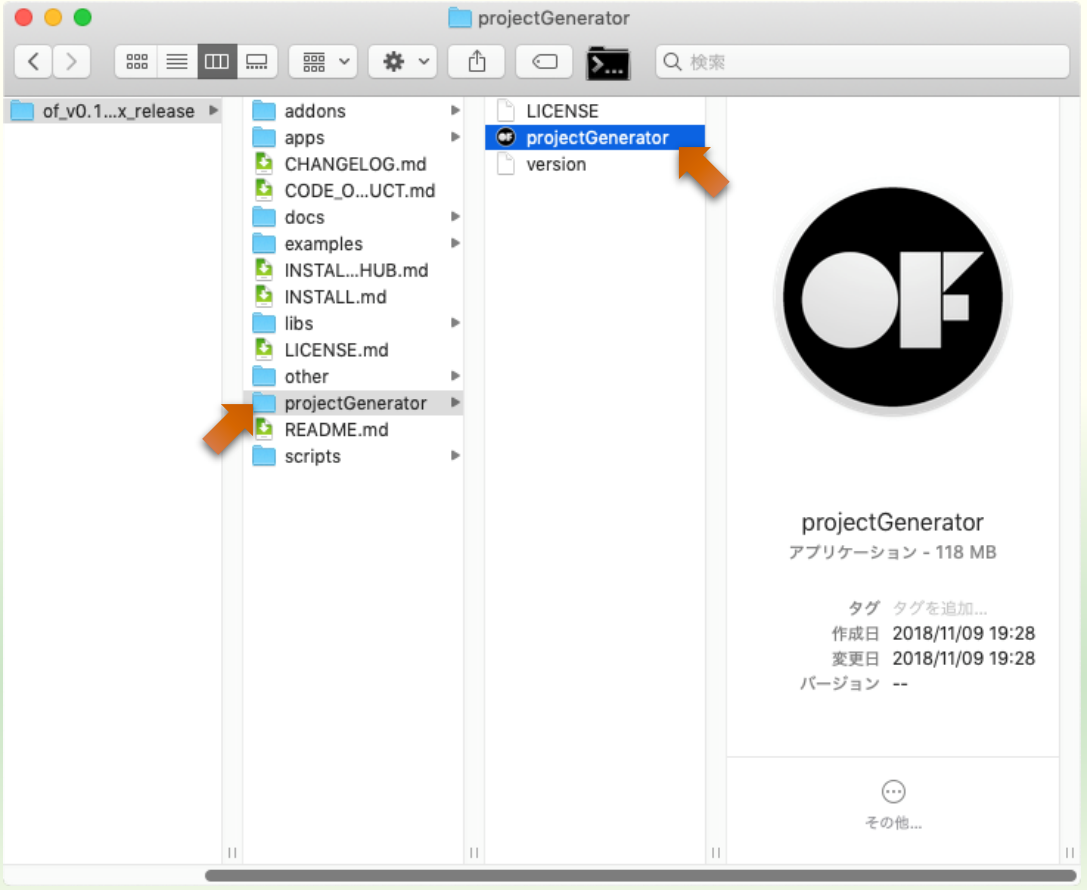
空のプロジェクトを作成してビルド

projectGenerator を起動する

windows 版のパッケージ



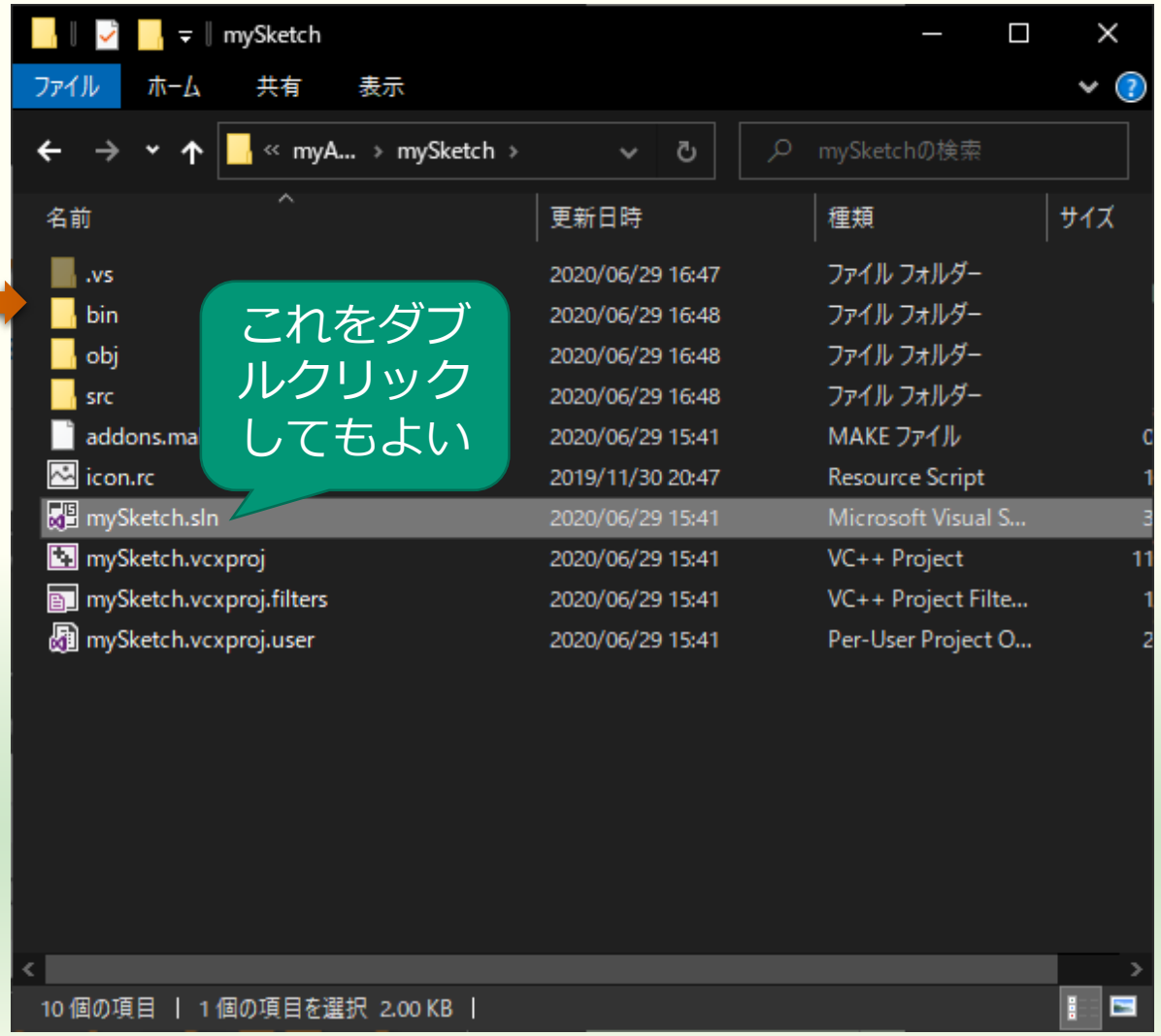
macOS 版のパッケージ



空のプロジェクトの作成

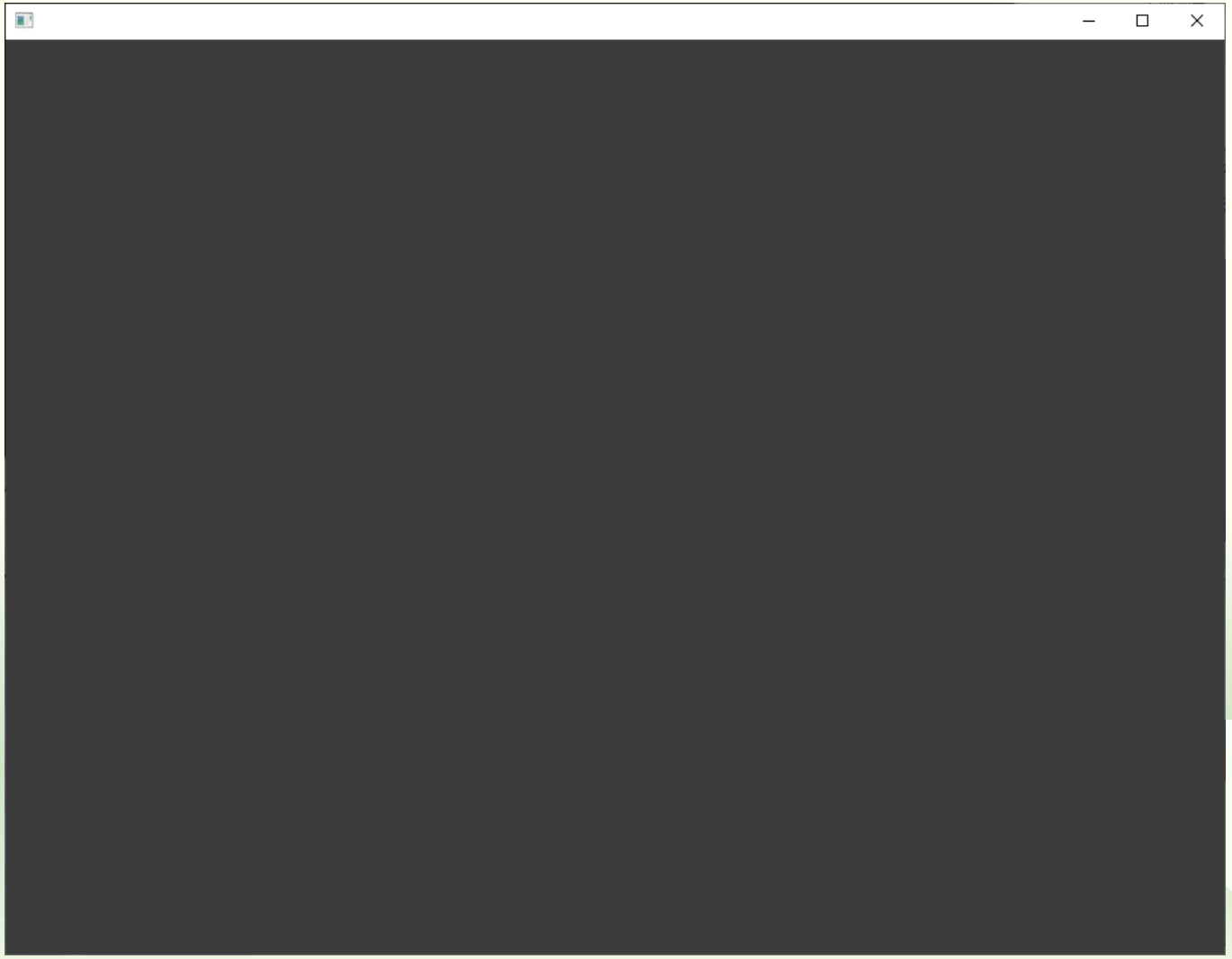
- Project name:
  - 作成するプロジェクト（プログラム）の名前
- Project path:
  - 作成するプロジェクトのファイルを置く場所
  - openFrameworks のパッケージを展開した場所の中の apps¥myApps

プロジェクトの作成成功



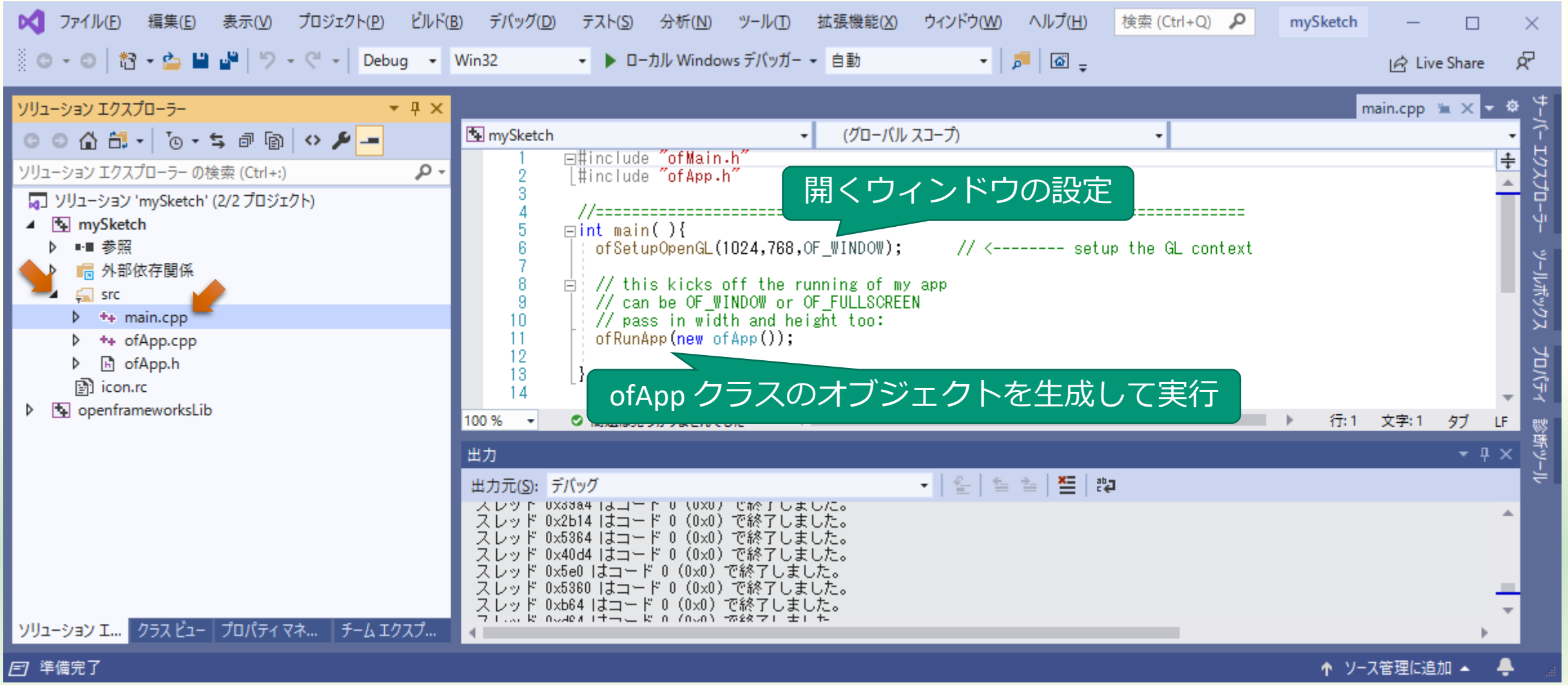
IDE (Visual Studio, Xcode) で開く

実行中のウィンドウ

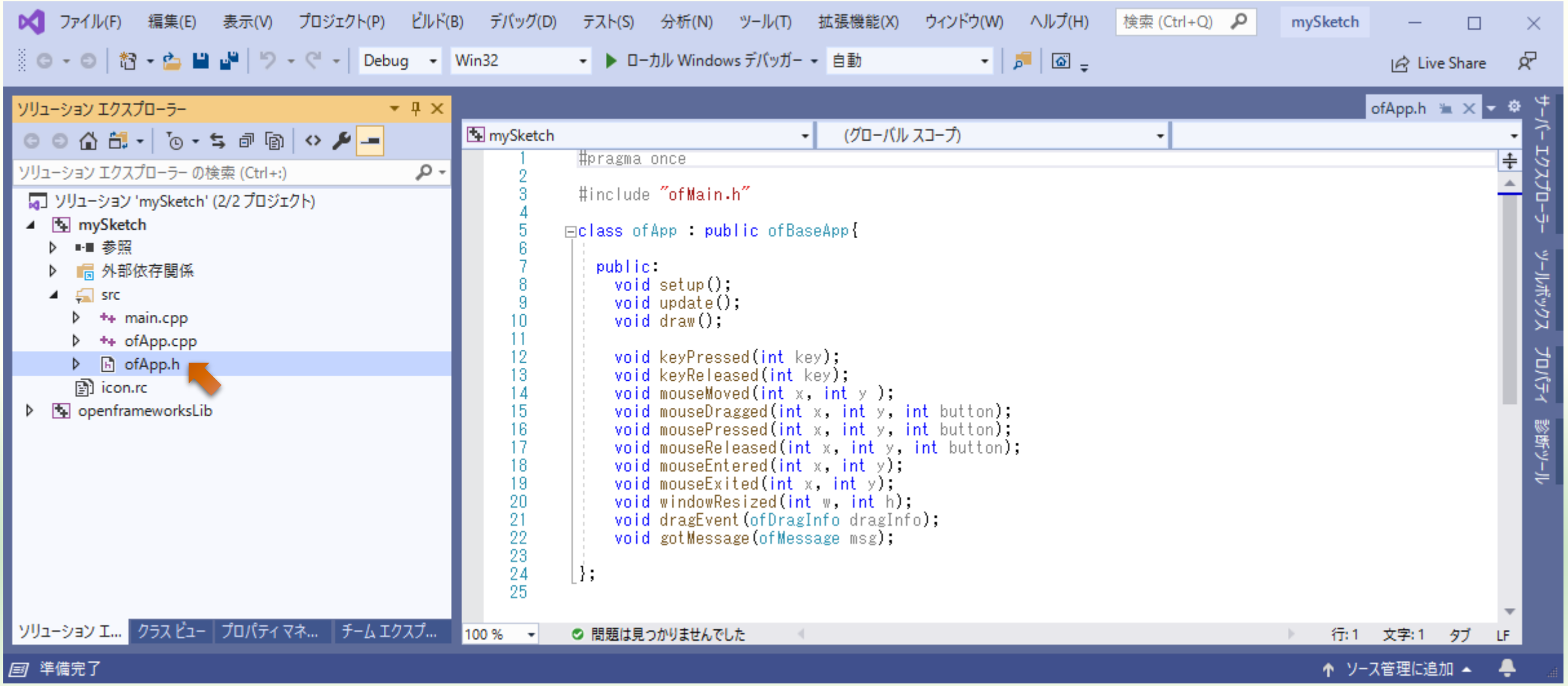


ESC キーで終了

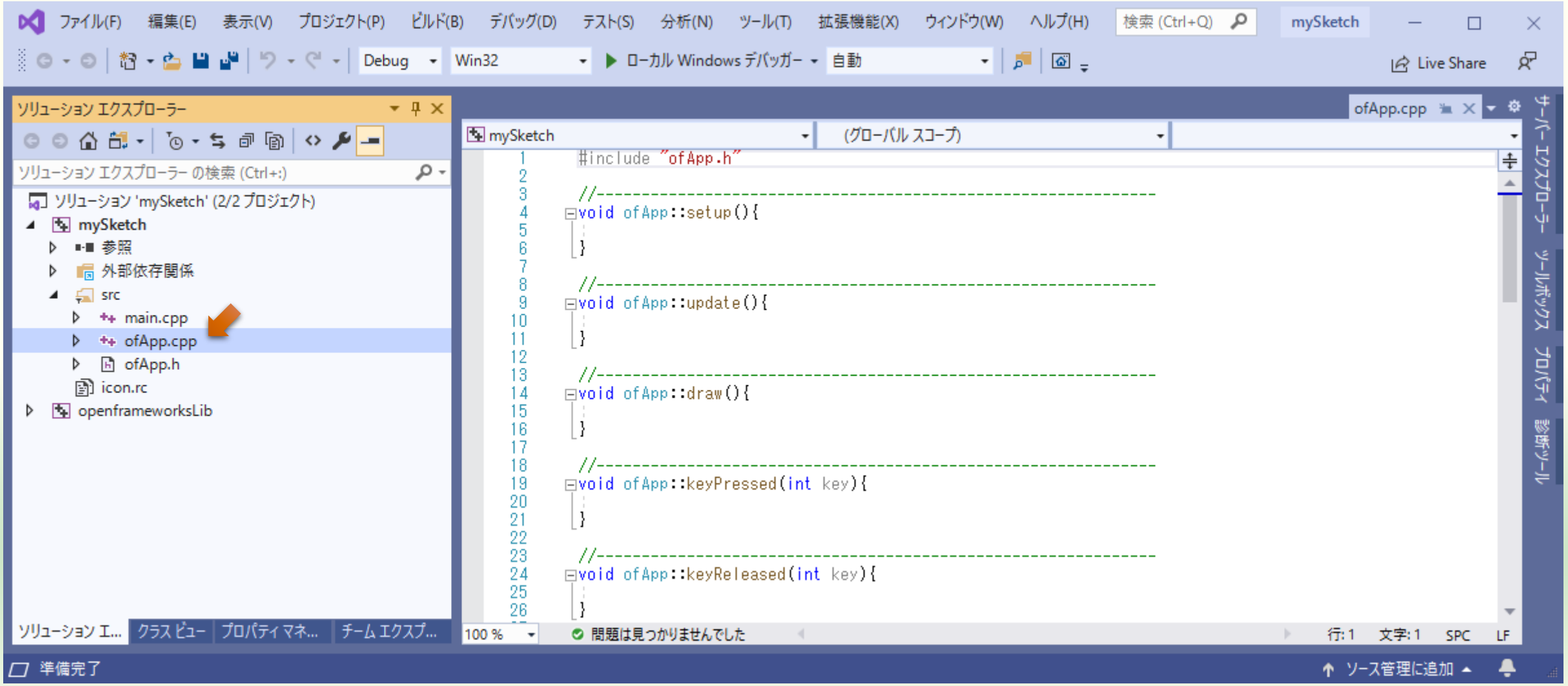
main() 関数は（今は）いじらない



ofApp クラス（“クラス” の話は後日）



ofApp クラスのメンバの実装はまだ空



課題１－５

openFrameworks を使ったプログラムの動作を確認してみよう

ofApp.cpp の内容（先頭部分）

```
#include "ofApp.h"

//--------------------------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------------------------
void ofApp::update(){

}

//--------------------------------------------------------------
void ofApp::draw(){

}

//--------------------------------------------------------------
void ofApp::keyPressed(int key){

}
```

メンバ関数 (画面表示関連)

- void ofApp::setup()
  - アプリケーションを起動したときに**一度だけ**実行される
- void ofApp::update()
  - 画面の表示を行う前に**繰り返し**実行される
- void ofApp::draw()
  - 画面の表示を**繰り返し**行う

“void” は戻り値を返さないことを表す
これらの関数は return を使わない

setup() で文字を出力してみる

```
#include "ofApp.h“

//--------------------------------------------
void ofApp::setup(){
  std::cout << "setup()¥n";
}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){

}
```

```
setup()
```

setup() は
最初に一度だけ実行される

update() で文字を出力してみる

```
#include "ofApp.h“

//--------------------------------------------
void ofApp::setup(){
  std::cout << "setup()¥n";
}

//--------------------------------------------
void ofApp::update(){
  std::cout << "update()¥n";
}

//--------------------------------------------
void ofApp::draw(){

}
```

```
setup()
update()
update()
update()
update()
update()
update()
update()
update()
update()
update()
update()
update()
update()
update()
update()
update()
update()
update()
update()
…
```

update() は
繰り返し何度も実行されている

draw() で文字を出力してみる

```
#include "ofApp.h“

//--------------------------------------------------
void ofApp::setup(){
  std::cout << "setup()¥n";
}

//--------------------------------------------------
void ofApp::update(){
  std::cout << "update()¥n";
}

//--------------------------------------------------
void ofApp::draw(){
  std::cout << "draw()¥n";
}
```

```
setup()
update()
draw()
update()
draw()
update()
draw()
update()
draw()
update()
draw()
update()
draw()
update()
draw()
update()
draw()
update()
draw()
update()
draw()
update()
draw()
update()
draw()
…
```

draw() は update() の後に実行される

メンバ関数 (キー操作関連)

- keyPressed(int key)
  - キーを押したときに実行
  - key は押したキー
- keyReleased(int key)
  - キーを離したときに実行
  - key は押していたキー

キーは文字が表示されている**黒い**ウィンドウ（コンソール）ではなく**グレー**のウィンドウでタイプしてください

keyPressed(int key) で文字を出力してみる

```
//--------------------------------------------
void ofApp::keyPressed(int key){
  std::cout << static_cast<char>(key)
            << " was  pressed¥n";
}

//--------------------------------------------
void ofApp::keyReleased(int key){

}
```

```
…
update()
draw()
update()
draw()
c was pressed
update()
draw()
update()
draw()
update()
draw()
d was pressed
update()
draw()
update()
draw()
update()
draw()
…
```

このウィンドウ（コンソール）ではなくアプリケーションのウィンドウで

キーを押したときに draw() の後で実行される

keyReleased(int key) で文字を出力してみる

```
//--------------------------------------------
void ofApp::keyPressed(int key){
  std::cout << static_cast<char>(key)
            << " was  pressed¥n";
}

//--------------------------------------------
void ofApp::keyReleased(int key){
 std::cout << static_cast<char>(key)
           << " was  released¥n";
}
```

```
…
update()
draw()
f was pressed
update()
draw()
update()
draw()
f was released
update()
draw()
g was pressed
update()
draw()
update()
draw()
g was released
update()
draw()
…
```

キーを離したときに
draw() の後で実行される

課題１－６

簡単な図形を描いてみよう

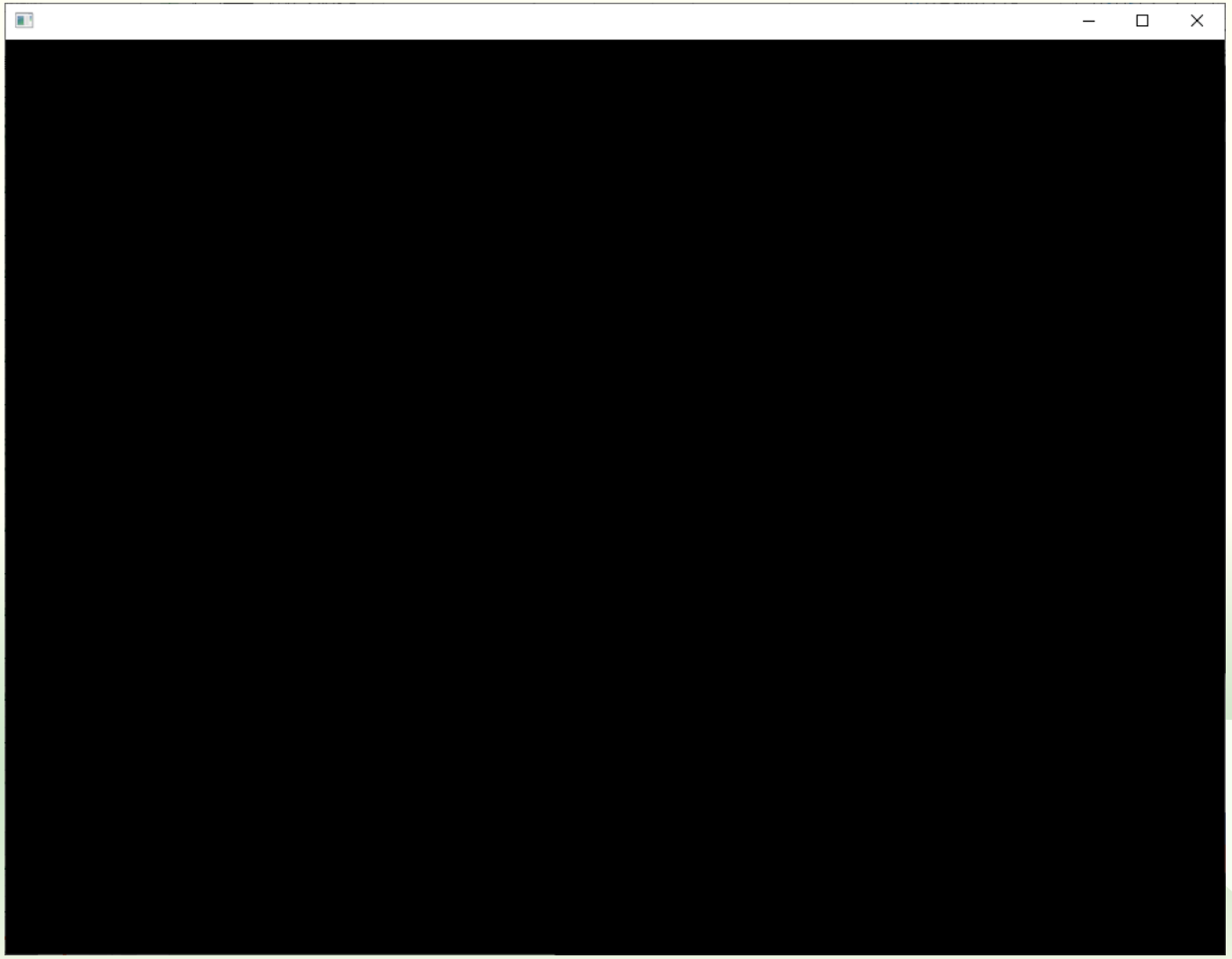
setup() で背景色を指定する

```
#include "ofApp.h“

//--------------------------------------------
void ofApp::setup(){
  ofBackground(0, 0, 0);
}
//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){

}
```

(r = 0, g = 0, b = 0) ⇒ 黒

- void ofBackground(int r, int g, int b, int a=255)
  - 背景色を指定する
    - r: 背景色の赤成分の強さ、0～255
    - g: 背景色の緑成分の強さ、0～255
    - b: 背景色の青成分の強さ、0～255
    - a: 背景色の不透明度、 0 （透明）～ 255（不透明）、省略時は 255
  - ofBackGround(255, 255, 255) なら白
  - マニュアル
    - https://openframeworks.cc/documentation/graphics/ofGraphics/#show_ofBackground

背景色が黒になる



マニュアルの見方

global functions

- ofBackground()
- ofBackgroundGradient()
- ofBackgroundHex()
- ofBeginSaveScreenAsPDF()
- ofBeginSaveScreenAsSVG()
- ofBeginShape()
- ofBezierVertex()
- ofClear()
- ofClearAlpha()
- ofCurveVertex()
- ofCurveVertices()
- ofDisableAlphaBlending()
- ofDisableAntiAliasing()
- ofDisableBlendMode()
- ofDisableDepthTest()
- ofDisablePointSprites()
- ofDisableSmoothing()

ofBackground(...)

```
void ofBackground(const ofColor &c)
```

ofBackground(...)

```
void ofBackground(int brightness, int alpha=255)
```

ofBackground(...)

```
void ofBackground(int r, int g, int b, int a=255)
```

引数の指定方法は
3 種類ある

今回使用したもの

Documentation from code comments

Sets the background color.

It takes as input r,g,b (0-255). The background is cleared automatically, just before the draw() command, so if the background color is not changing, you could call this inside of setup() (once, at the start of the application). If the background color is changing, you can call this inside of update().

```
void ofApp::setup(){
    ofBackground(255,0,0);   // Sets the background color to red
}
```

使用例

関数の定義の見方

```
void ofBackground(int r, int g, int b, int a=255)
```

- `void`：戻り値のデータ型 void は値を返さない場合
- `int r`：第１引数は int 型
- `int g`：第２引数は int 型
- `int b`：第３引数は int 型
- `int a=255`：第４引数は int 型 省略したときは 255

使用例

```
ofBackground(0, 0, 0)
```

背景色は不透明の黒

```
ofBackground(255, 0, 0, 51)
```

背景色は不透明度20%の赤

draw() で円を描く

```
#include "ofApp.h“

//--------------------------------------------------
void ofApp::setup(){
  ofBackground(0, 0, 0);
}

//--------------------------------------------------
void ofApp::update(){

}

//--------------------------------------------------
void ofApp::draw(){
  ofDrawCircle(0.0f, 0.0f, 100.0f);
}
```

中心 (0, 0), 半径 100

- void ofDrawCircle(float x, float y, float radius)
  - 円を描く
    - x: 円の中心の x 座標値
    - y: 円の中心の y 座標値
    - radius: 円の半径
  - ofDrawCircle(30.0f, 40.0f, 100.0f) なら (30, 40) を中心に半径 100 の円
  - マニュアル
    - https://openframeworks.cc//documentation/graphics/ofGraphics/#!show_ofDrawCircle

左上に円が描かれる

左上が原点 (0, 0)

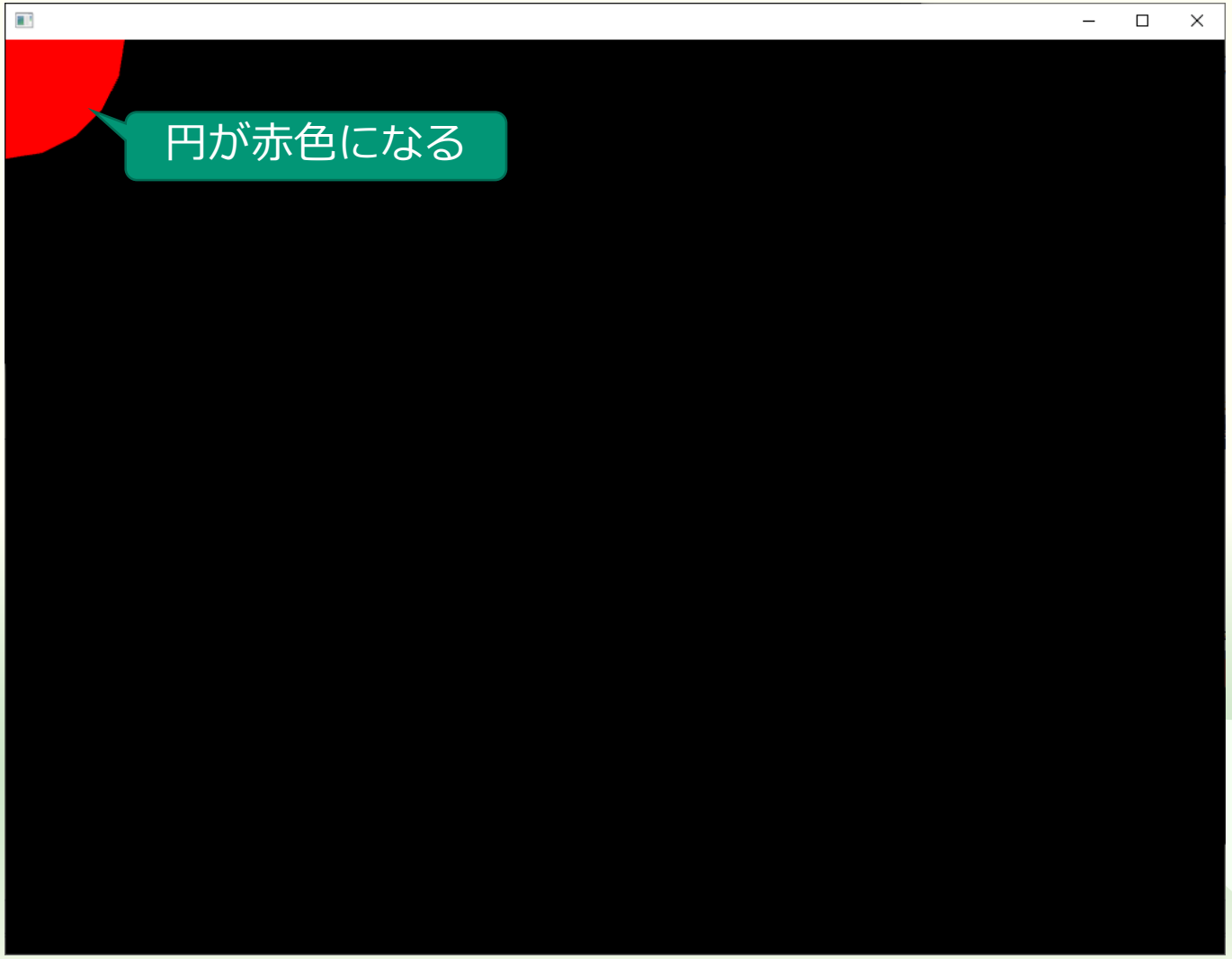
円の色を変える

```
#include "ofApp.h“

//--------------------------------------------------
void ofApp::setup(){
  ofBackground(0, 0, 0);
}

//--------------------------------------------------
void ofApp::update(){

}

//--------------------------------------------------
void ofApp::draw(){
  ofSetColor(255, 0, 0);
  ofDrawCircle(0.0f, 0.0f, 100.0f);
}
```

(r = 255, g = 0, b = 0) ⇒ 赤

- void ofSetColor(int r, int g, int b)
- void ofSetColor(int r, int g, int b, int a)
  - 以後描画するものの色を指定する
    - r: 背景色の赤成分の強さ、0～255
    - g: 背景色の緑成分の強さ、0～255
    - b: 背景色の青成分の強さ、0～255
    - a: 背景色の不透明度、０（透明）～255（不透明）、省略時は 255
  - マニュアル
    - https://openframeworks.cc//documentation/graphics/ofGraphics/#!show_ofSetColor

円の色が変わる



draw() で矩形を描く

```
#include "ofApp.h“

//--------------------------------------------------
void ofApp::setup(){
  ofBackground(0, 0, 0);
}

//--------------------------------------------------
void ofApp::update(){

}

//--------------------------------------------------
void ofApp::draw(){
  ofSetColor(255, 0, 0);
  ofDrawCircle(0.0f, 0.0f, 100.0f);
  ofDrawRectangle(300.0f, 200.0f, 90.0f, 60.0f);
}
```

- void ofDrawRectangle(float x1, float y1, float w, float h)
  - 矩形を描く
    - x1: 矩形の左端の x 座標
    - y1: 矩形の上端の y 座標
    - w: 矩形の幅
    - h: 矩形の高さ
  - ofDrawRectangle(10.0f, 20.0f, 100.0f, 200.0f) なら (10, 20) を左上にして幅 100 高さ 200 の矩形
  - マニュアル
    - https://openframeworks.cc//documentation/graphics/ofGraphics/#!show_ofDrawRectangle

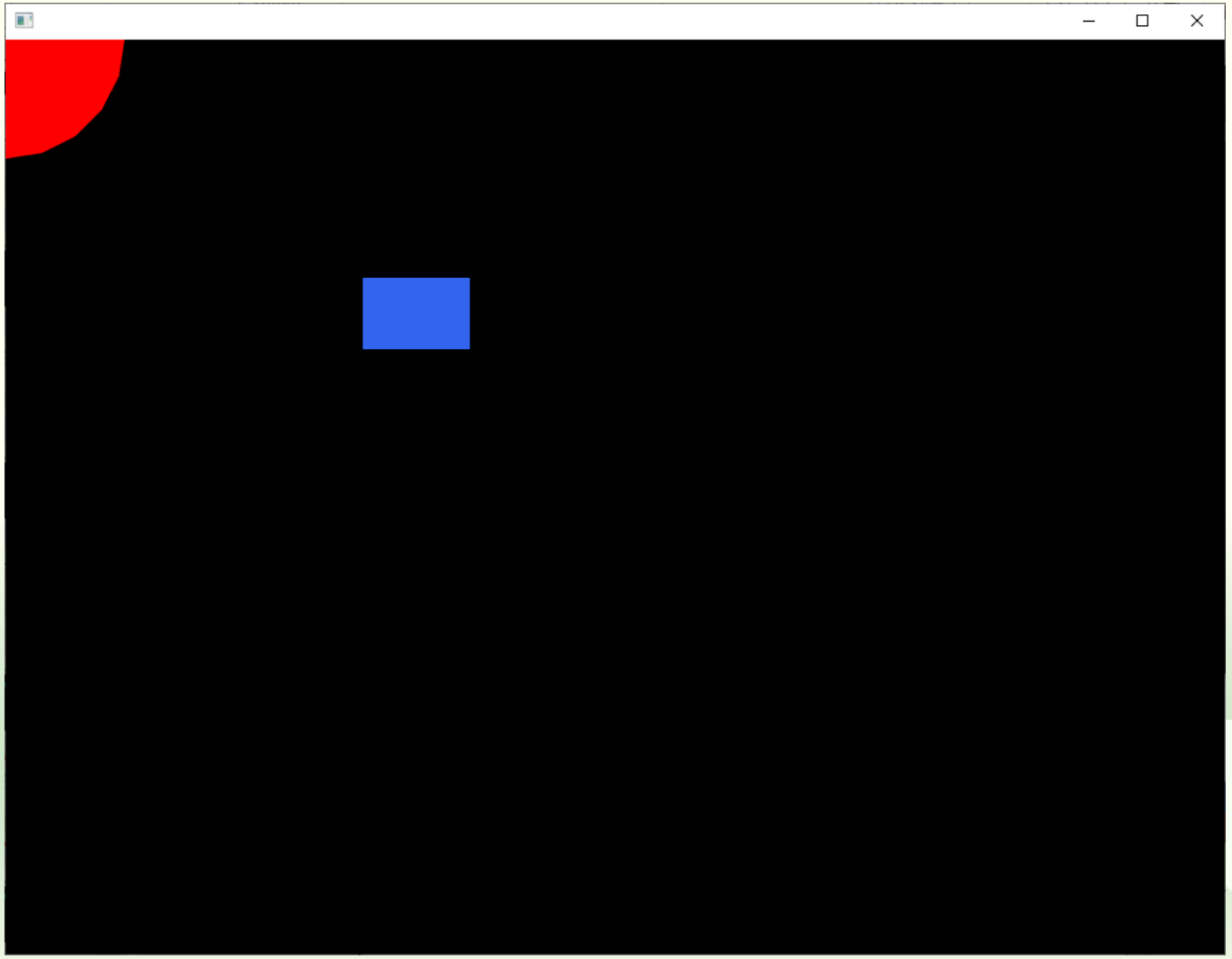
矩形が追加される

矩形の色を変える

```
#include "ofApp.h“

//--------------------------------------------------
void ofApp::setup(){
  ofBackground(0, 0, 0);
}

//--------------------------------------------------
void ofApp::update(){

}

//--------------------------------------------------
void ofApp::draw(){
  ofSetColor(255, 0, 0);
  ofDrawCircle(0.0f, 0.0f, 100.0f);
  ofSetColor(50, 100, 240);
  ofDrawRectangle(300.0f, 200.0f, 90.0f, 60.0f);
}
```

- ofSetColor() で指定した色は ofSetColor() の呼び出し以降に描画する図形に適用される

矩形の色が変わる



課題１－７

自分で図形を作ってみよう

二次元図形を描く

- 以下の openFrameworks の関数を使って何か二次元の図形を描いてください
  - ofDrawCircle()
    - https://openframeworks.cc//documentation/graphics/ofGraphics/#!show_ofDrawCircle
  - ofDrawCurve()
    - https://openframeworks.cc//documentation/graphics/ofGraphics/#!show_ofDrawCurve
  - ofDrawEllipse()
    - https://openframeworks.cc/documentation/graphics/ofGraphics/#show_ofDrawEllipse
  - ofDrawLine()
    - https://openframeworks.cc/documentation/graphics/ofGraphics/#show_ofDrawLine
  - ofDrawRectRounded()
    - https://openframeworks.cc/documentation/graphics/ofGraphics/#show_ofDrawRectRounded
  - ofDrawRectangle()
    - https://openframeworks.cc/documentation/graphics/ofGraphics/#show_ofDrawRectangle
  - ofDrawTriangle()
    - https://openframeworks.cc/documentation/graphics/ofGraphics/#show_ofDrawTriangle

課題のアップロード

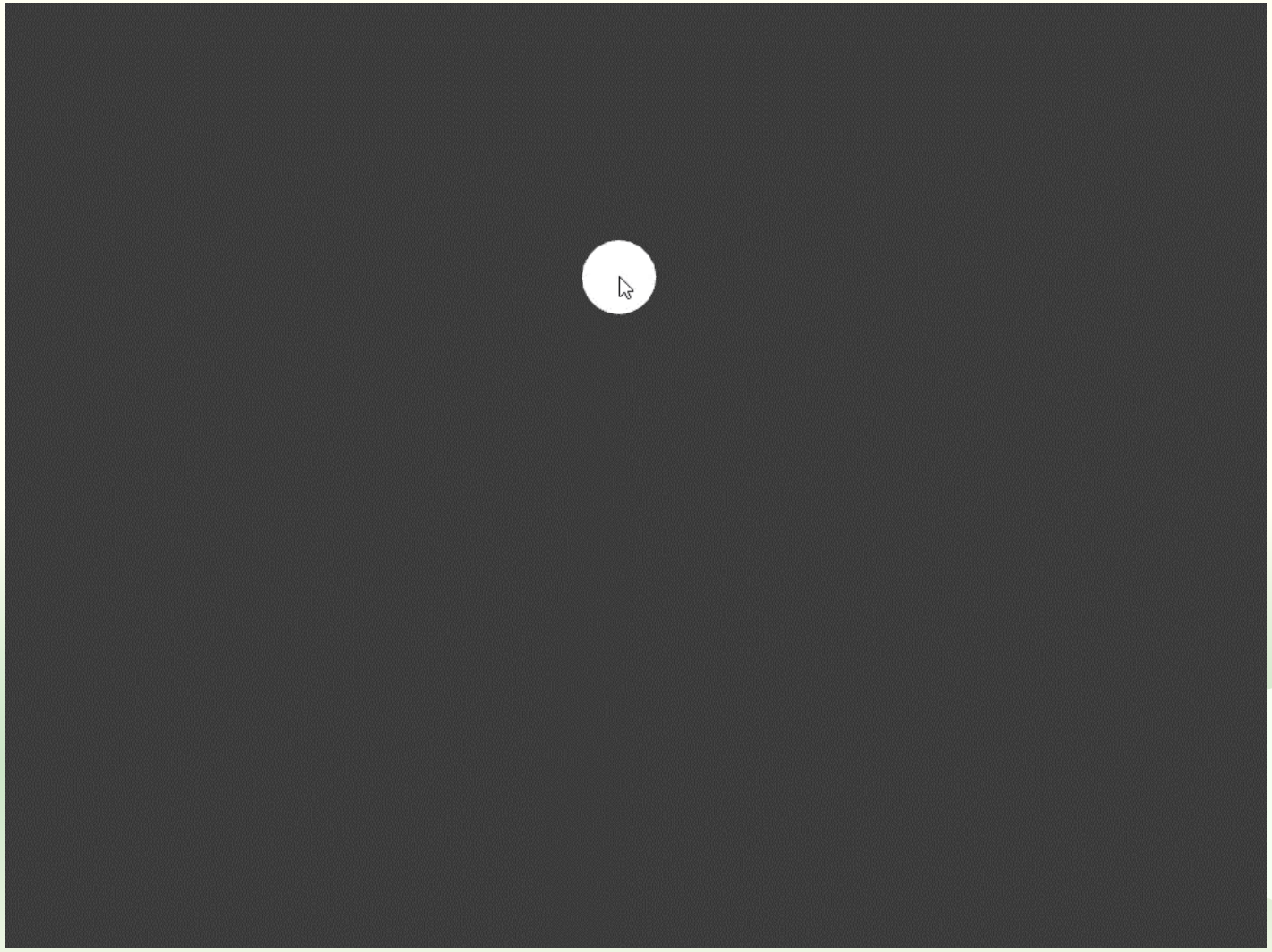
- 作成したプログラムの実行結果のスクリーンショットを撮って **1-7.png** というファイル名で保存し、Moodle の第 1 回課題にアップロードしてください
- ソースプログラム **ofApp.h** と **ofApp.cpp** を Moodle の第 1 回課題にアップロードしてください

ofApp.cpp の保存場所

メディアプログラミング演習

第2回

本日は円を放り投げるだけのアプリの作成

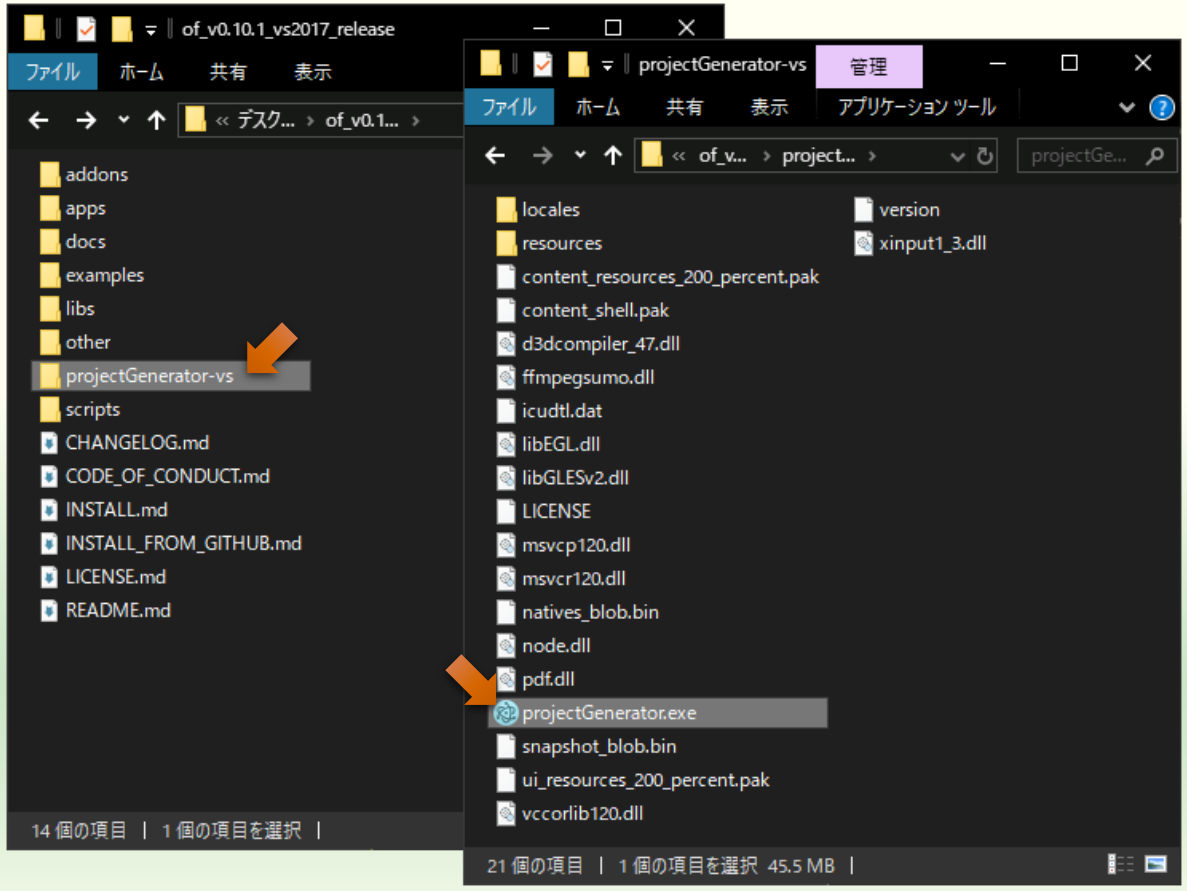


準備

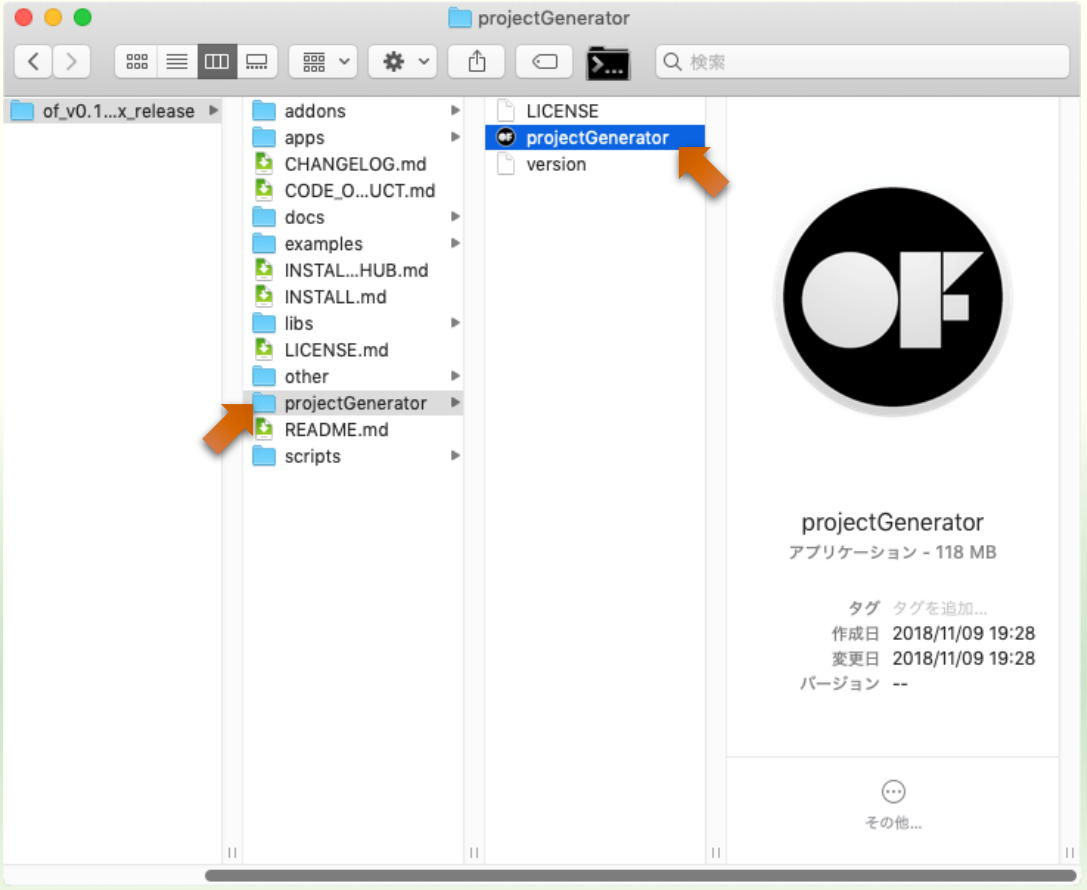
プロジェクトの作成

projectGenerator を起動する

windows 版のパッケージ



macOS 版のパッケージ

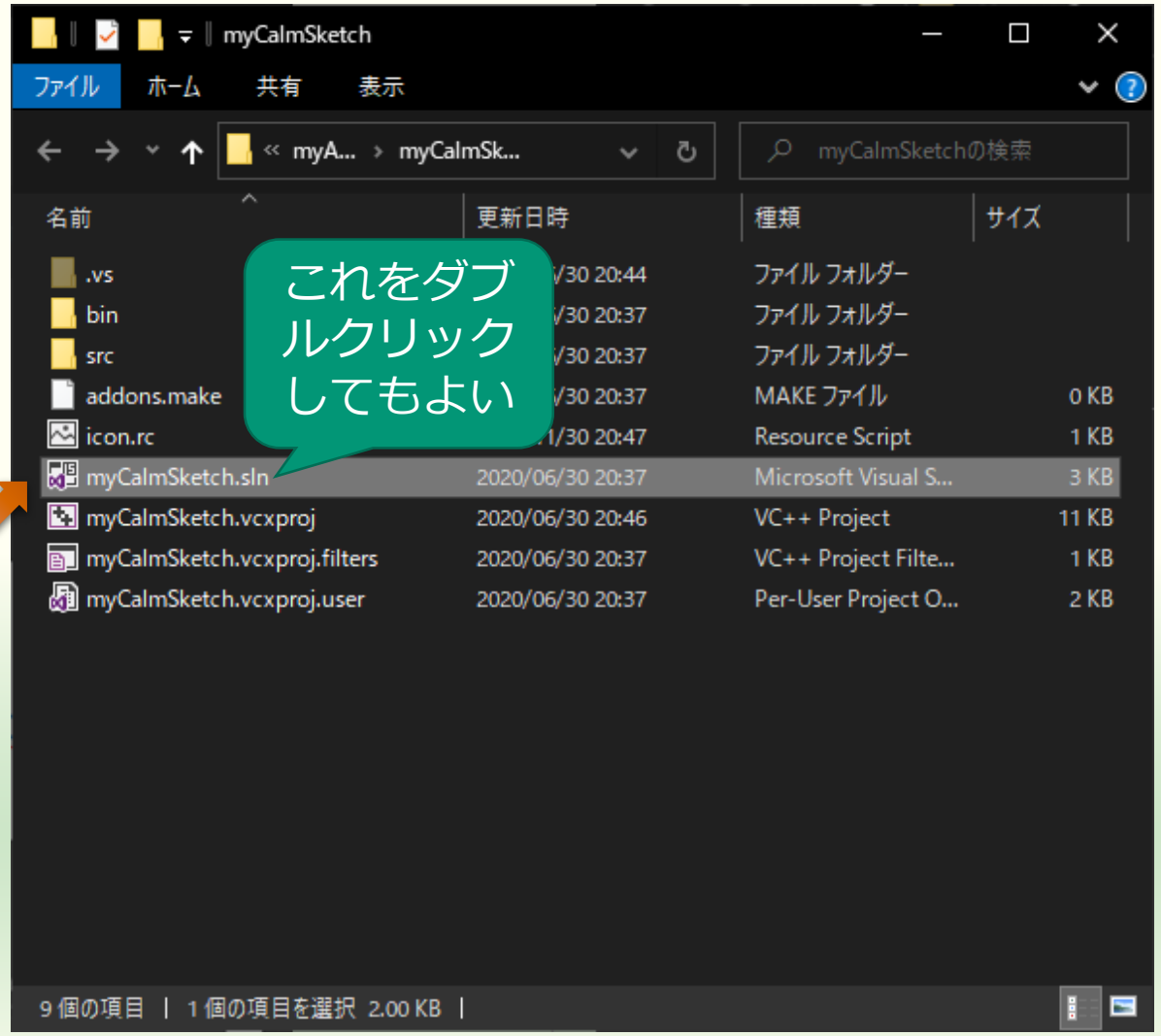


空のプロジェクトの作成

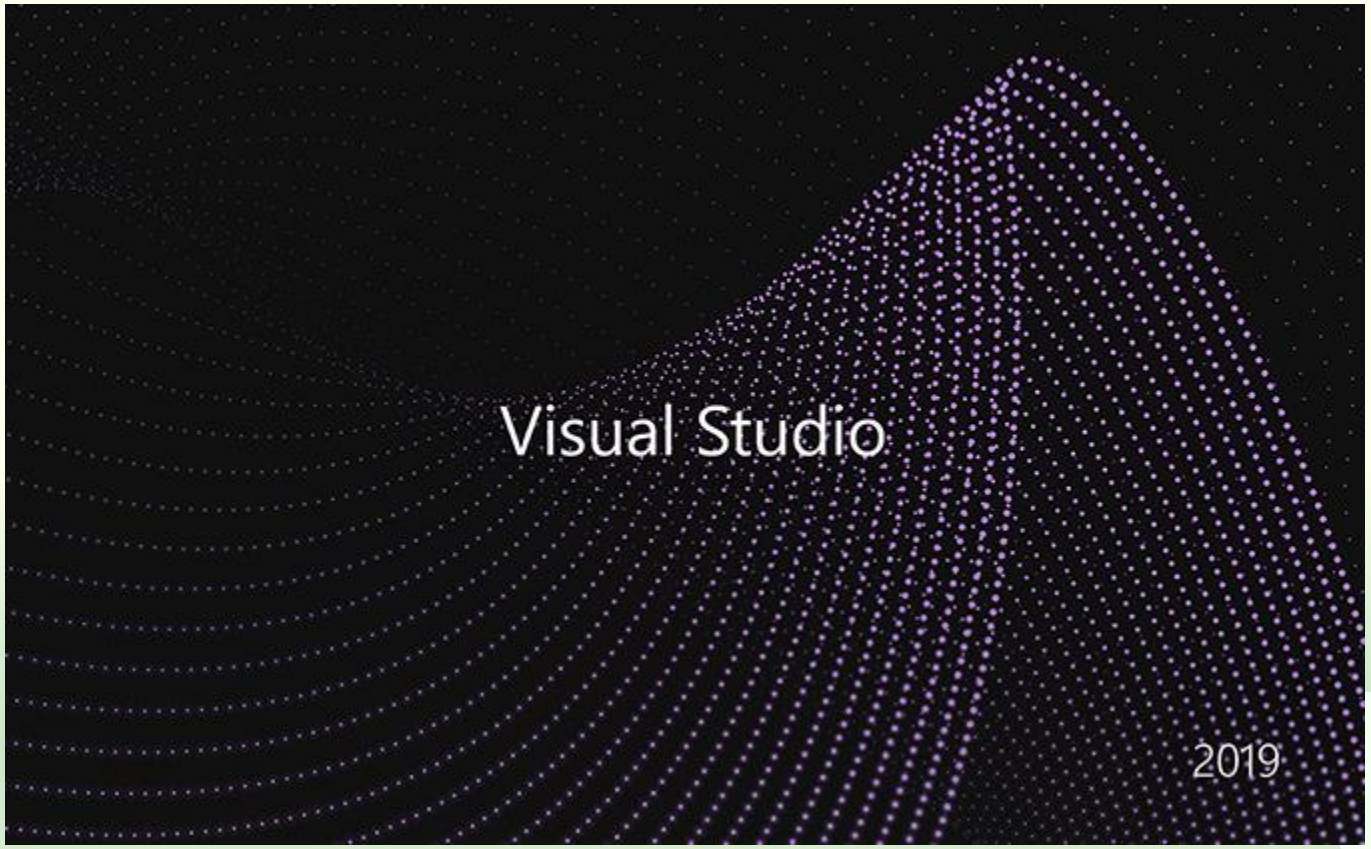
- Project name:
  - 作成するプロジェクト（プログラム）の名前
- Project path:
  - 作成するプロジェクトのファイルを置く場所
  - openFrameworks のパッケージを展開した場所の中の apps¥myApps

プロジェクトの作成成功

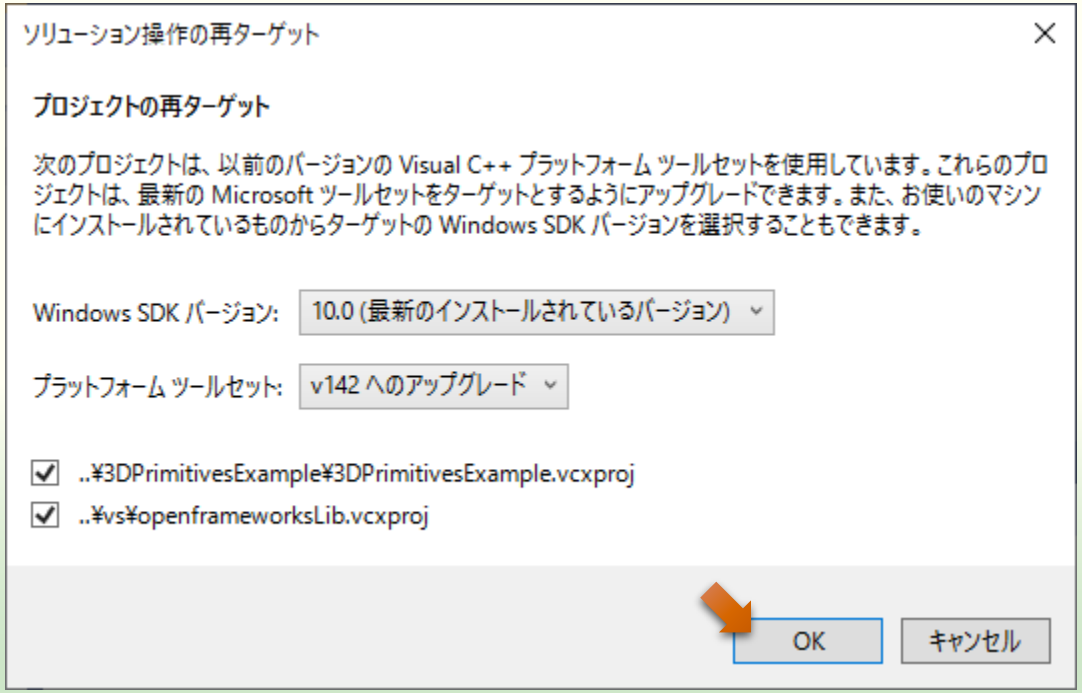
クリックすると開く



Visual Studio 2019 が起動する

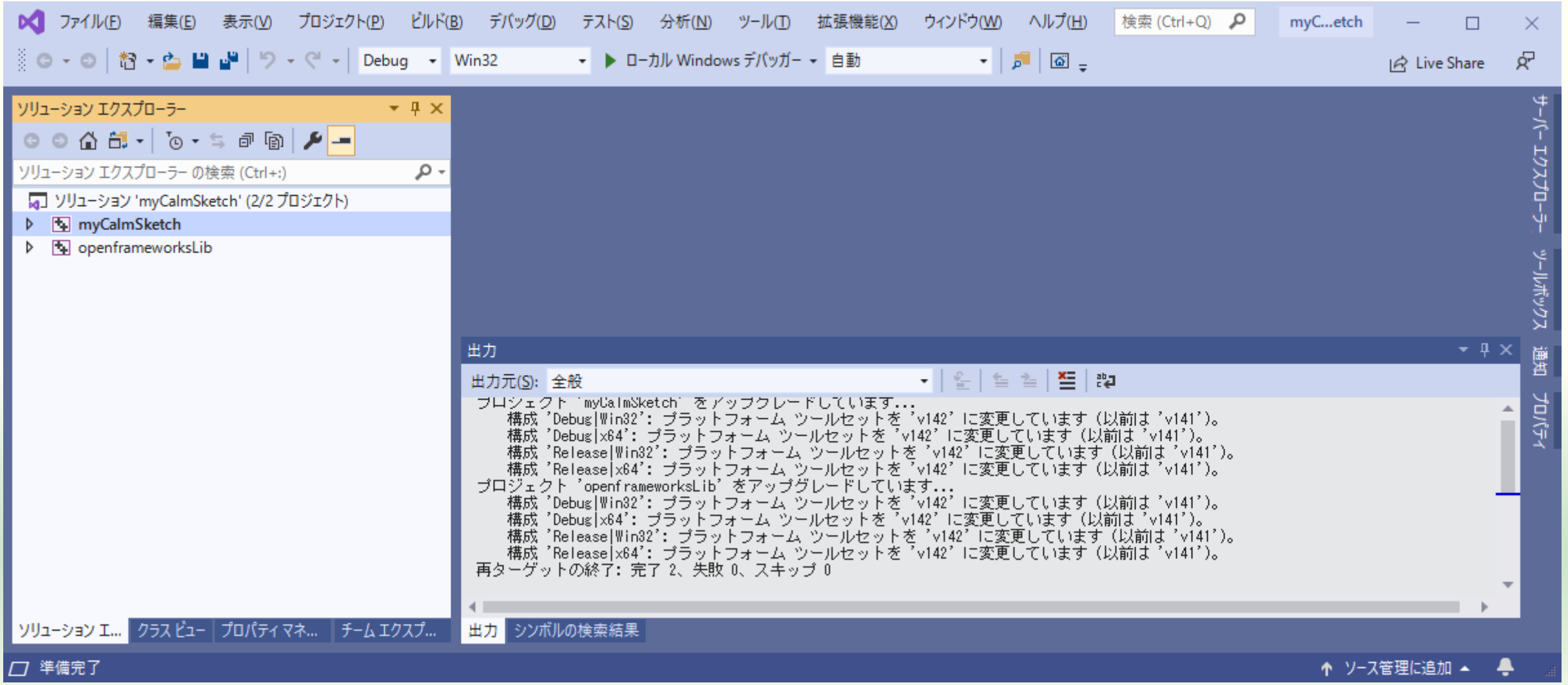


ソリューションの再ターゲット



Visual Studio は頻繁に更新しているので皆さんがお使いの Visual Studio SDK のバージョンと合わない場合がある

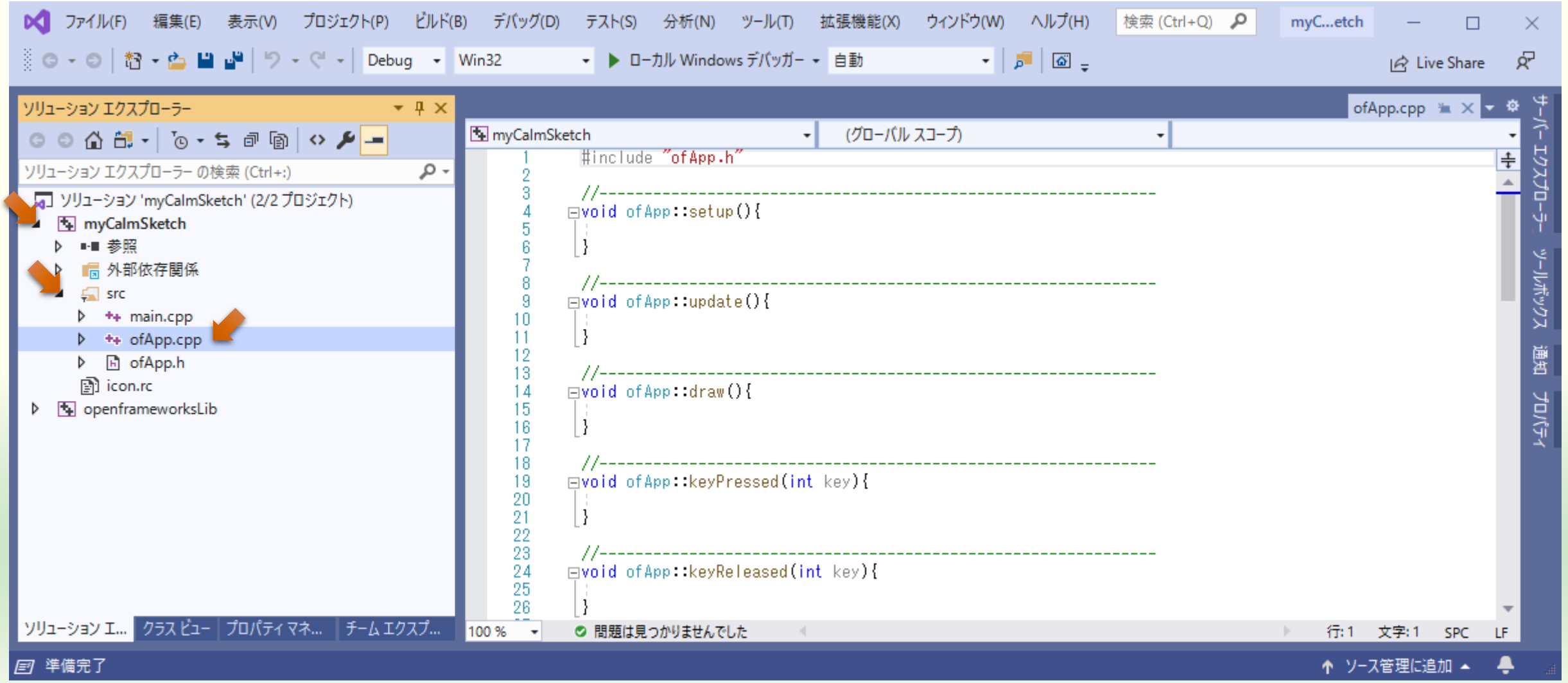
Visual Studio 起動



変数・定数とデータ型

データを記憶する

ofApp.cpp を開く



ofDrawCircle() で円を描く

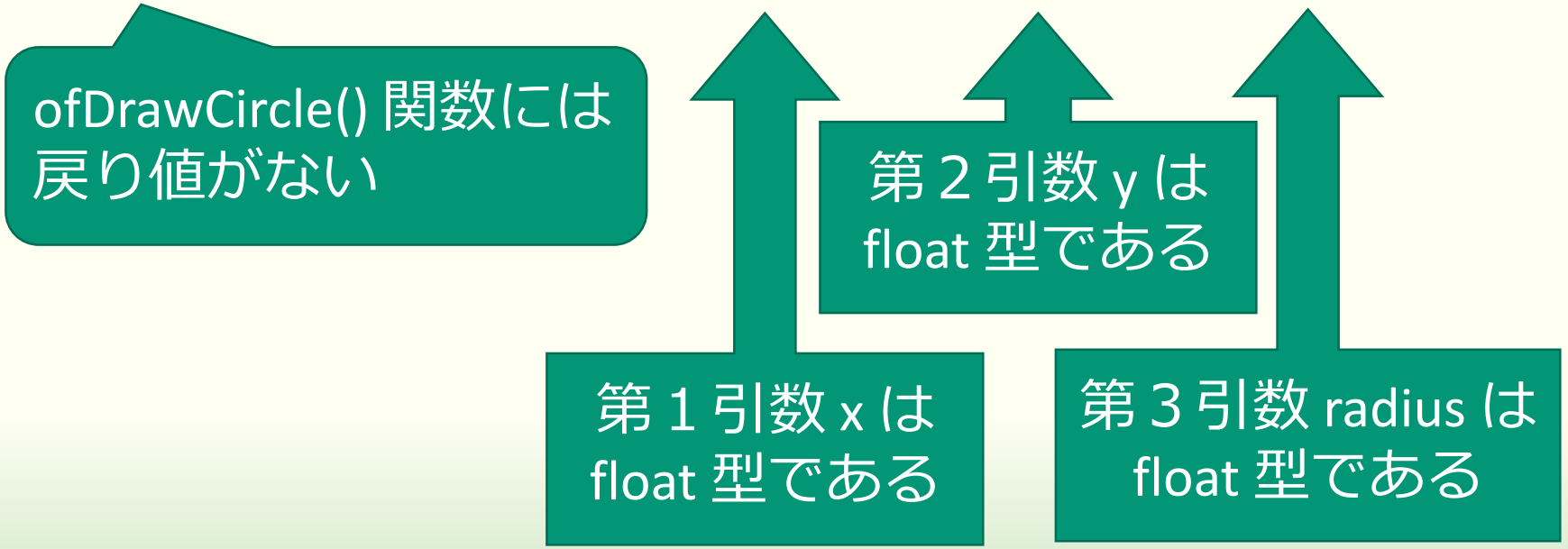
```
//--------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  ofDrawCircle(0.0f, 0.0f, 30.0f);
}

//--------------------------------------------
void ofApp::keyPressed(int key){

}
```

中心位置　半径

- (0, 0) を中心に半径 30 の円を ofDrawCircle() 関数で描く

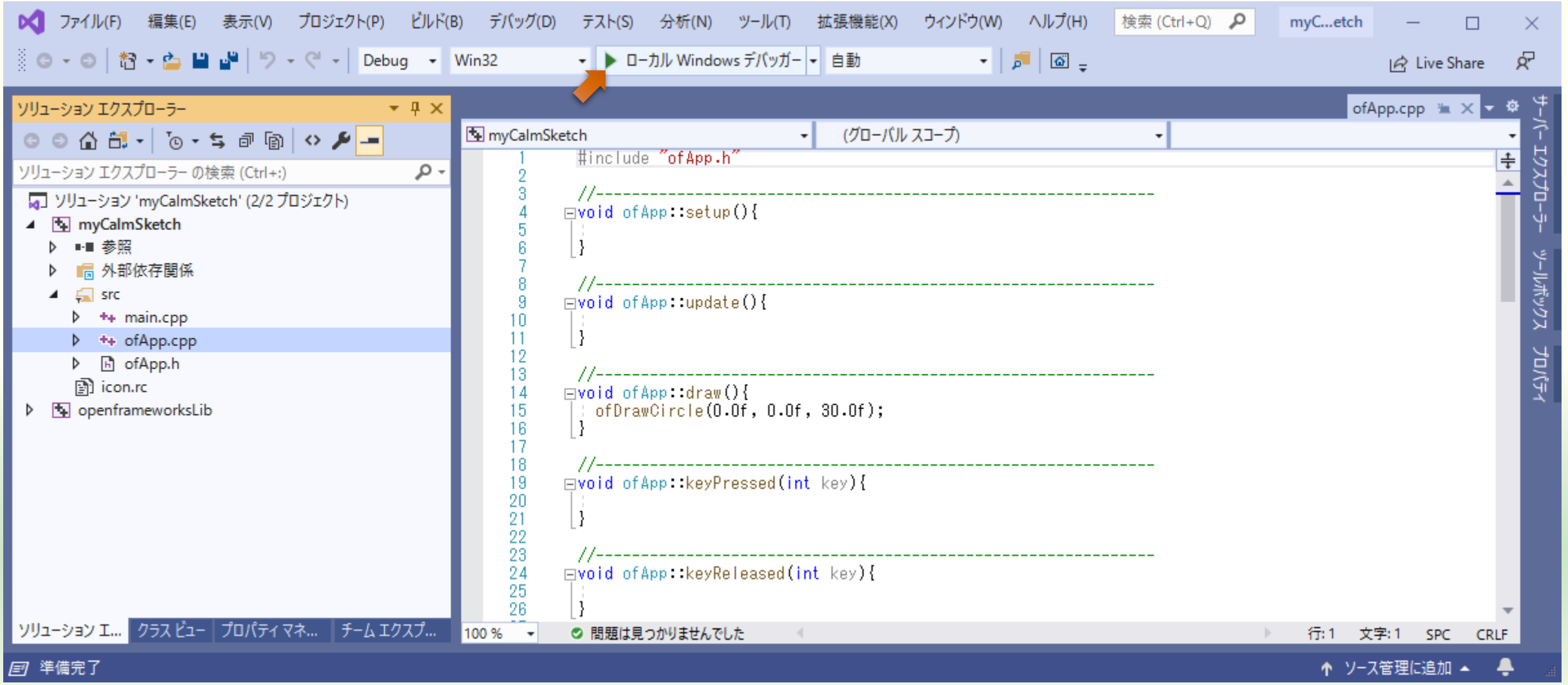
ofDrawCircle() の宣言

- **void ofDrawCircle(float x, float y, float radius)**

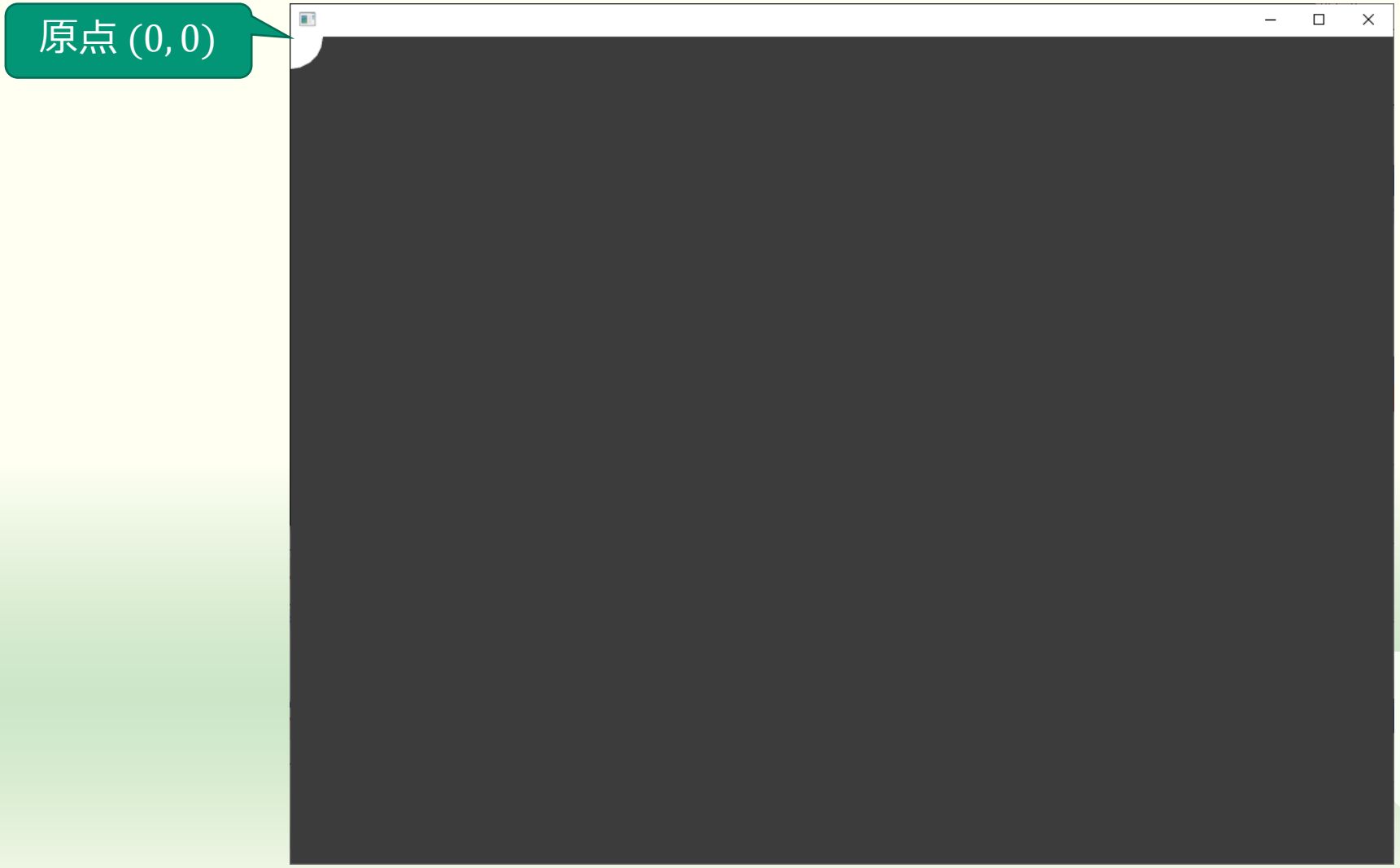


- 仮引数 x, y, radius が何を表すのかは示されていない
  - マニュアルやコメントで説明されている（はず）
  - 宣言では引数名 x, y, radius が省略されている場合がある

ビルドと実行


ファイル(F)
編集(E)
表示(V)
プロジェクト(P)
ビルド(B)
デバッグ(D)
テスト(S)
分析(N)
ツール(T)
拡張機能(X)
ウィンドウ(W)
ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myCalmSketch' (2/2 プロジェクト)
myCalmSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.cpp
myCalmSketch
(グローバル スコープ)
#include "ofApp.h"
//--
void ofApp::setup(){
}
//--
void ofApp::update(){
}
//--
void ofApp::draw(){
ofDrawCircle(0.0f, 0.0f, 30.0f);
}
//--
void ofApp::keyPressed(int key){
}
//--
void ofApp::keyReleased(int key){
}
100 %
問題は見つかりませんでした
行: 1
文字: 1
SPC
CRLF
ソリューション エ...
クラス ビュー
プロパティ マネ...
チーム エクスプ...
サーバー エクスプローラー
ツールボックス
プロパティ
準備完了
ソース管理に追加

左上の原点を中心に円が描かれる



定数

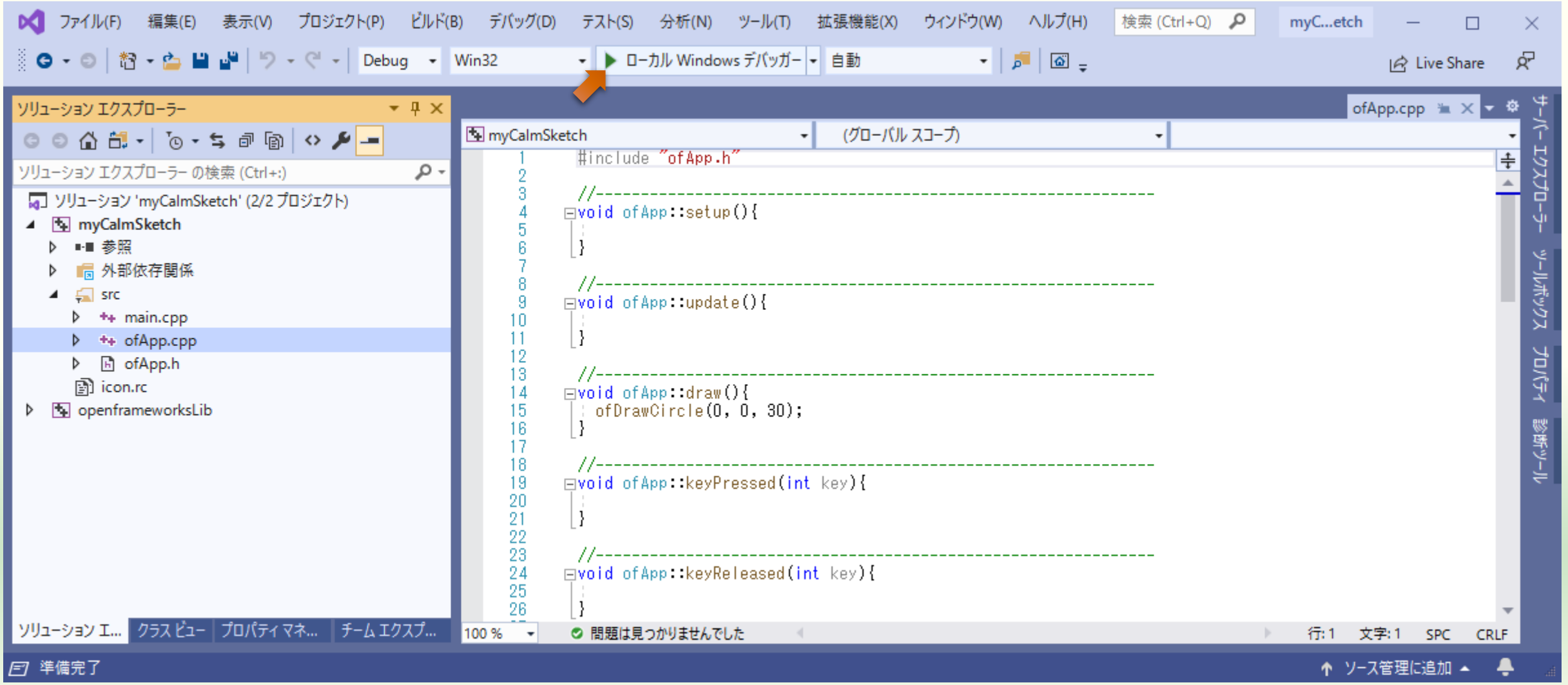
- 100
  - int 型の定数 (10 進数)
- 100.0
  - double 型の定数
- 100.0f
  - float 型の定数
- 1.0e2f
  - float 型の定数
    - $1.0 \times 10^2 = 100$
- 0100
  - int 型の定数 (8 進数)
    - 10 進数の 64
- 0b100
  - int 型の定数 (2進数)
    - 10進数の 4
- 0x100
  - int 型の定数 (16進数)
    - 10進数の 256

引数を整数にしてみる

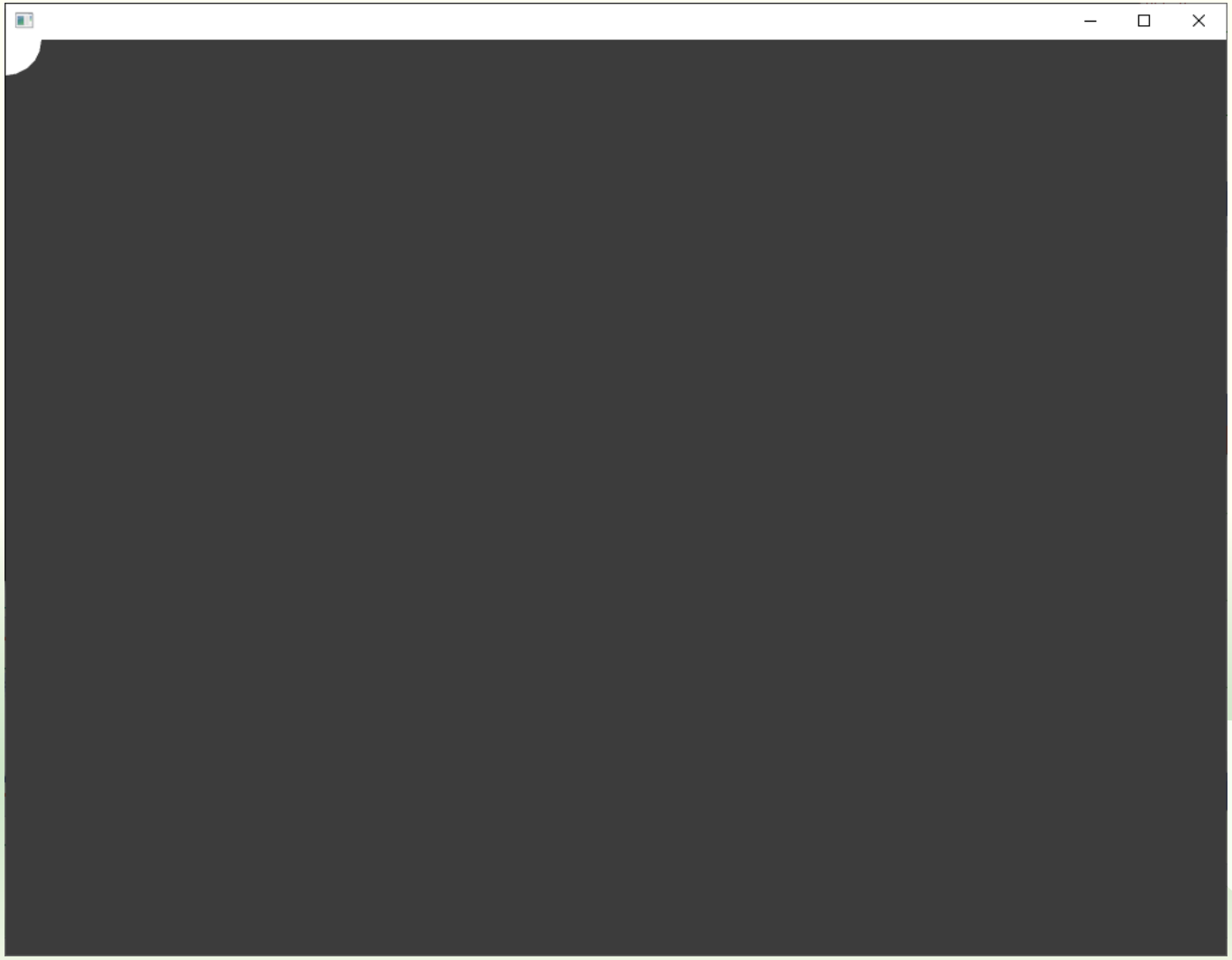
```
//--------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  ofDrawCircle(0, 0, 30);
}

//--------------------------------------------
void ofApp::keyPressed(int key){

}
```

- int から float への型変換が自動的に行われる
  - **暗黙の型変換**

ビルドと実行


ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) テスト(S) 分析(N) ツール(T) 拡張機能(X) ウィンドウ(W) ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myCalmSketch' (2/2 プロジェクト)
myCalmSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.cpp
myCalmSketch
(グローバル スコープ)
#include "ofApp.h"
//--
void ofApp::setup(){
}
//--
void ofApp::update(){
}
//--
void ofApp::draw(){
ofDrawCircle(0, 0, 30);
}
//--
void ofApp::keyPressed(int key){
}
//--
void ofApp::keyReleased(int key){
}
100 %
問題は見つかりませんでした
行: 1 文字: 1 SPC CRLF
ソリューション エ... クラス ビュー プロパティ マネ... チーム エクスプ...
サーバー エクスプローラー ツールボックス プロパティ 診断ツール
準備完了
ソース管理に追加

特に変わらない



引数を変数で与える

```
//--------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  float x, y;
  float radius;

  x = 0.0f;
  y = 0.0f;
  radius = 30.0f;

  ofDrawCircle(x, y, radius);
}
```

変数宣言

代入

- 変数は**宣言**すれば使えるようになる
  - データの記憶場所がメモリ上に確保される
  - コンマ (,) で区切って複数の変数を宣言できる
- **代入**により変数に値を格納する

変数宣言に const を付けると代入できない

```
//--------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  float x, y;
  const float radius;

  x = 0.0f;
  y = 0.0f;
  radius = 30.0f;

  ofDrawCircle(x, y, radius);
}
```

- **const** を付けて宣言した変数は値を**変更できない**
  - 値を変更（代入）できない

初期化は変数宣言のときに値を設定する

```
//--------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  float x, y;
  const float radius = 30.0f;

  x = 0.0f;
  y = 0.0f;

  ofDrawCircle(x, y, radius);
}
```

初期化

- **初期化**は変数宣言の時に確保されたメモリに値を設定する

変数は宣言された { } 内でしか使えない

```
//--------------------------------------------
void ofApp::setup(){
  const float radius = 30.0f;
}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  float x, y;

  x = 0.0f;
  y = 0.0f;

  ofDrawCircle(x, y, radius);
}
```

エラー

- { } 内で宣言された変数は同じ {} 内でしか使えない
  - **局所変数（ローカル変数）**
- 変数が使える範囲のことを変数の**スコープ**という

変数宣言を関数の外に置くと共有できる

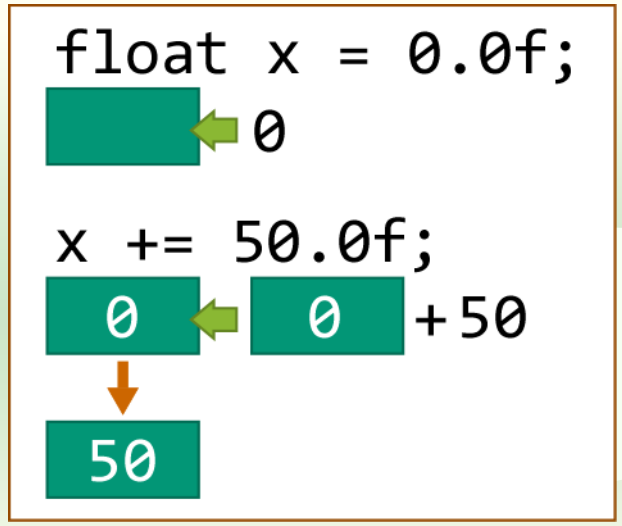
```
const float radius = 30.0f;

//--------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  float x, y;

  x = 0.0f;
  y = 0.0f;

  ofDrawCircle(x, y, radius);
}
```

- 変数宣言を関数の外に置くと**後に続く関数**で共通に使用できる
- **const** float radius = 100.0f;
  - const を付けて関数の外で宣言した変数は ofApp.cpp 内でのみ使用できる

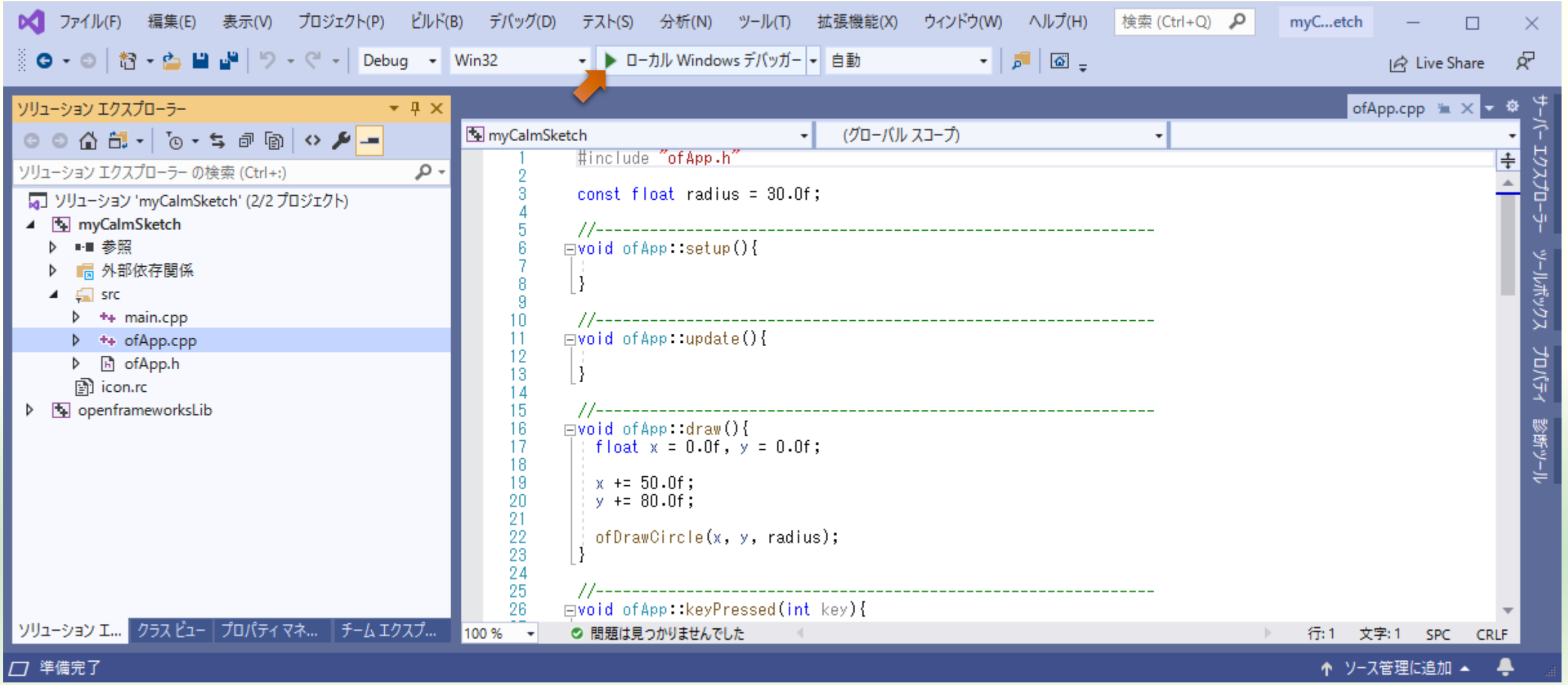
const を付けない変数の初期化は初期値

```
const float radius = 30.0f;

//--------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  float x = 0.0f, y = 0.0f;

  x += 50.0f;
  y += 80.0f;

  ofDrawCircle(x, y, radius);
}
```

- x, y の初期値を 0 にする
- **x += 50.0f;**
  - x に 50 を加える
  - x = x + 50.0f; と同じ



ビルドと実行

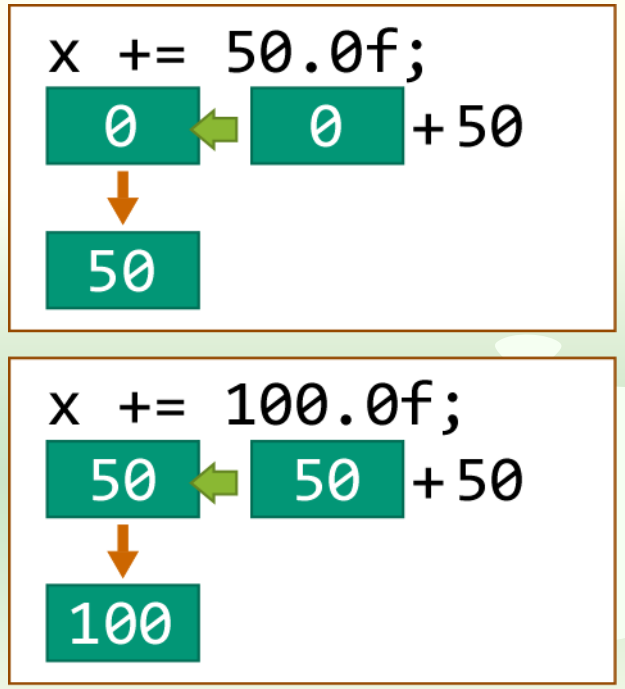

ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) テスト(S) 分析(N) ツール(T) 拡張機能(X) ウィンドウ(W) ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myCalmSketch' (2/2 プロジェクト)
myCalmSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.cpp
myCalmSketch
(グローバル スコープ)
#include "ofApp.h"
const float radius = 30.0f;
//--
void ofApp::setup(){
}
//--
void ofApp::update(){
}
//--
void ofApp::draw(){
float x = 0.0f, y = 0.0f;
x += 50.0f;
y += 80.0f;
ofDrawCircle(x, y, radius);
}
//--
void ofApp::keyPressed(int key){
100 %
問題は見つかりませんでした
行: 1 文字: 1 SPC CRLF
準備完了
ソース管理に追加
サーバー エクスプローラー ツールボックス プロパティ 診断ツール

中心が (50, 80) に移動する

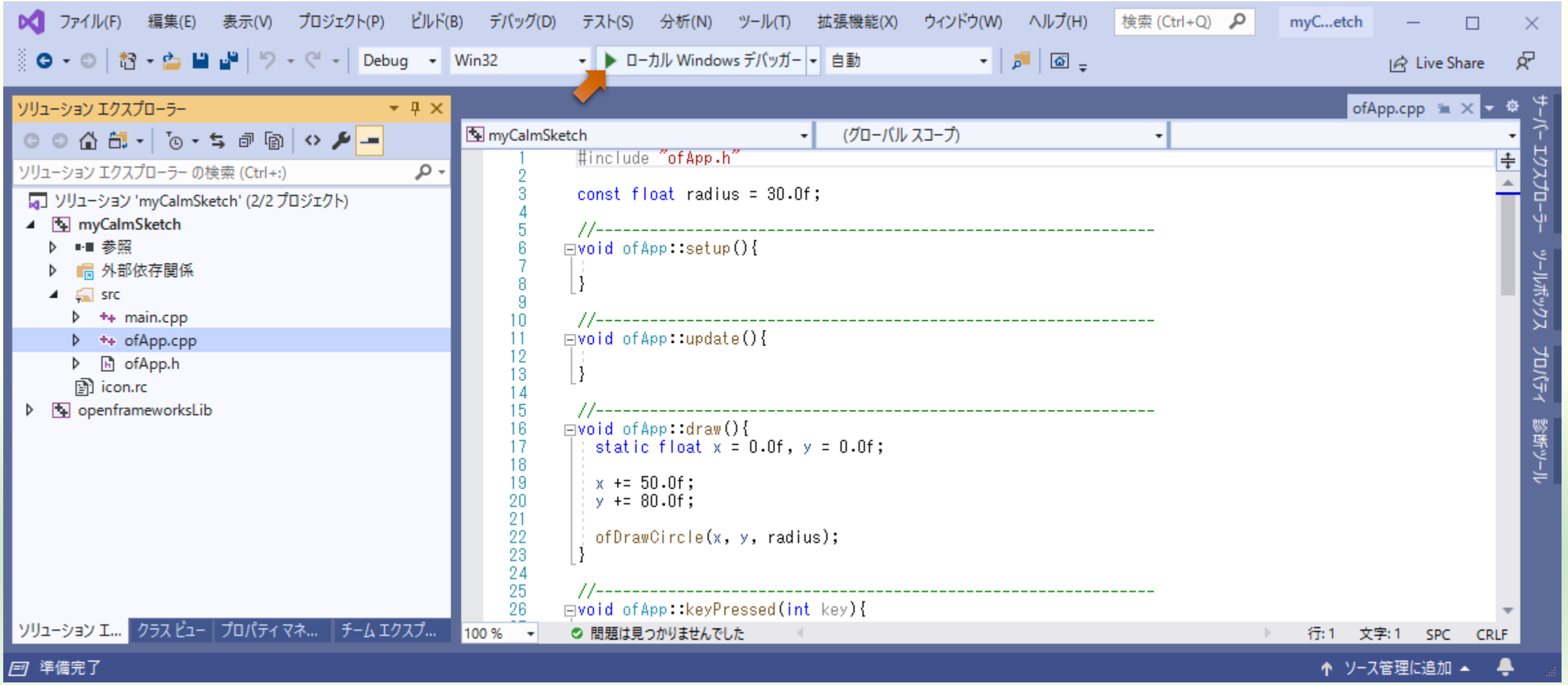
変数宣言に static を付けると値を保持する

```
const float radius = 30.0f;

//--------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  static float x = 0.0f, y = 0.0f;

  x += 50.0f;
  y += 80.0f;

  ofDrawCircle(x, y, radius);
}
```

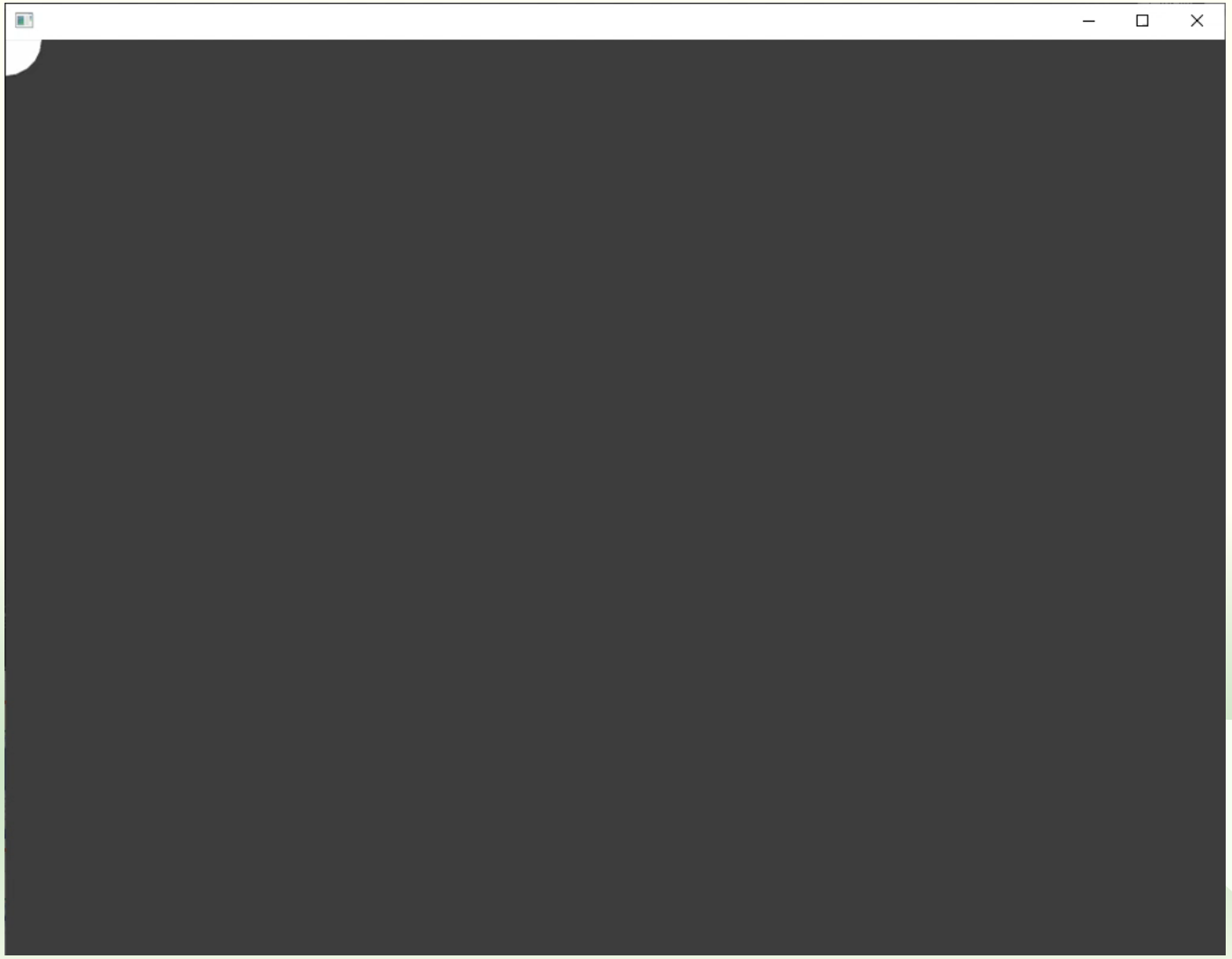
- draw() は繰り返し何度も実行されている
  - x += 50.0f も繰り返す



ビルドと実行


ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) テスト(S) 分析(N) ツール(T) 拡張機能(X) ウィンドウ(W) ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myCalmSketch' (2/2 プロジェクト)
myCalmSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.cpp
myCalmSketch
(グローバル スコープ)
#include "ofApp.h"
const float radius = 30.0f;
//--
void ofApp::setup(){
}
//--
void ofApp::update(){
}
//--
void ofApp::draw(){
static float x = 0.0f, y = 0.0f;
x += 50.0f;
y += 80.0f;
ofDrawCircle(x, y, radius);
}
//--
void ofApp::keyPressed(int key){
100 %
問題は見つかりませんでした
行: 1 文字: 1 SPC CRLF
ソリューション エ... クラス ビュー プロパティ マネ... チーム エクスプ...
サーバー エクスプローラー ツールボックス プロパティ 診断ツール
準備完了
ソース管理に追加

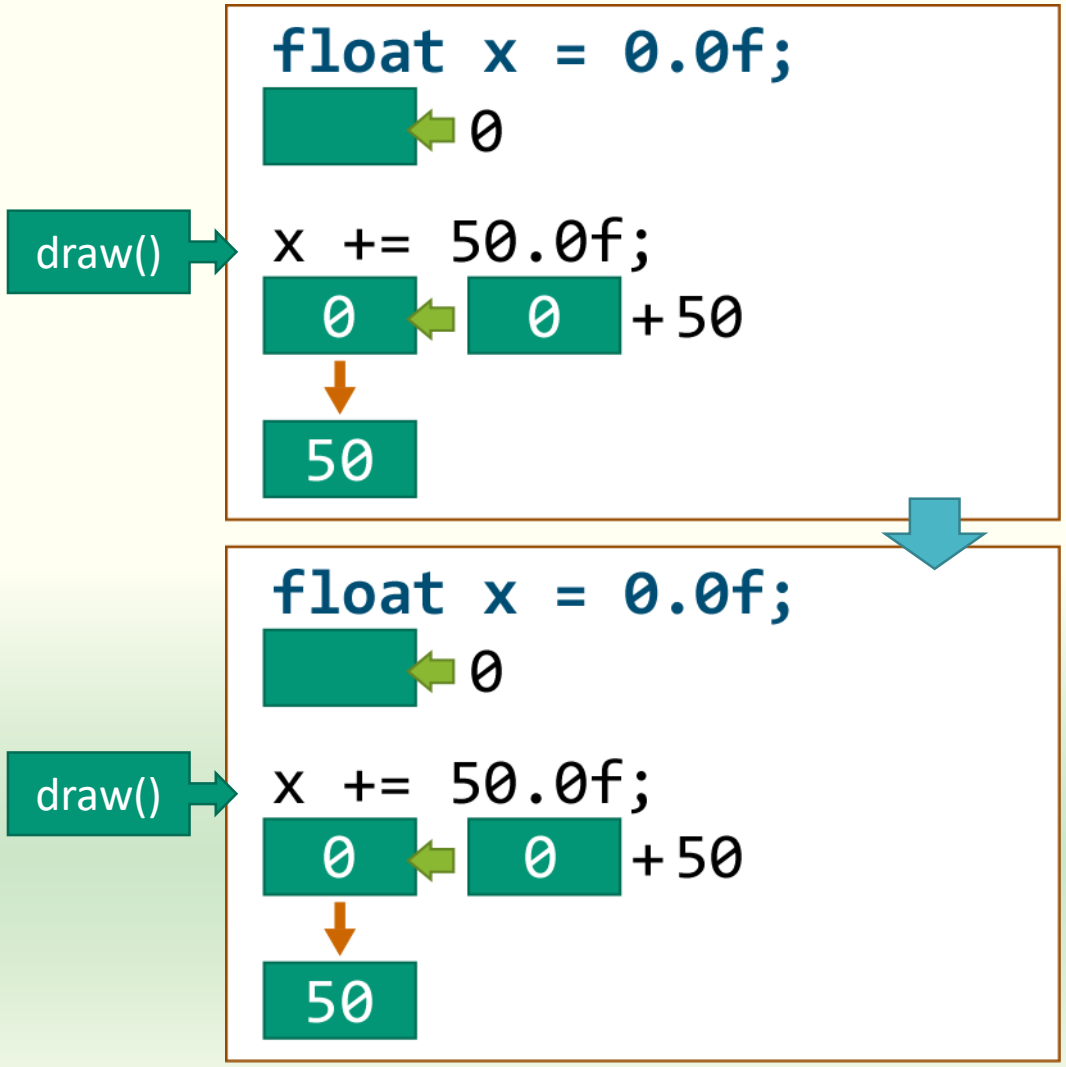
一瞬で消える

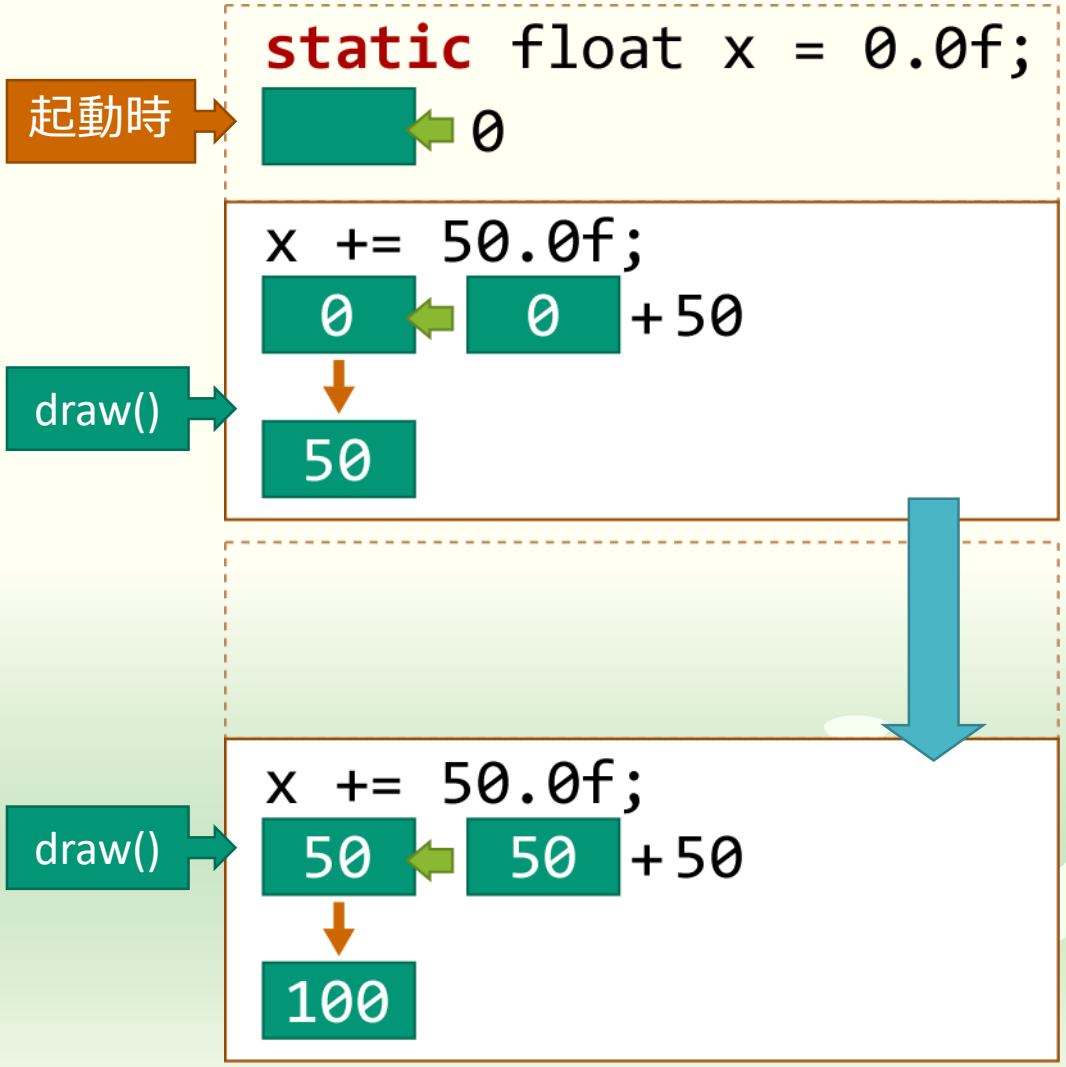


変数のメモリが確保されるタイミング

	static を付けない変数	static を付けた変数
メモリの確保	プログラムの実行が 変数のスコープに**入った**とき	プログラムの**起動**時
メモリの解放	プログラムの実行が 変数のスコープから**出た**とき	プログラムの**終了**時
初期化	変数宣言のところで **毎回**初期化される	起動時に**一度だけ** 初期化される
	自動変数	**静的変数**

静的変数の初期化は起動時に行われる





x, y の値の変更を update() で行う

```
const float radius = 30.0f;
static float x = 0.0f, y = 0.0f;

//--------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------
void ofApp::update(){
  x += 50.0f;
  y += 80.0f;
}

//--------------------------------------------
void ofApp::draw(){

  ofDrawCircle(x, y, radius);
}
```

- x, y の変数宣言を関数の外の update() と draw() の両方より前に置く
- **static** float x = 0.0f, y = 0.0f;
  - static を付けて関数の外で宣言した変数は ofApp.cpp 内でのみ使用できる

static を付けずに関数の外で変数宣言する

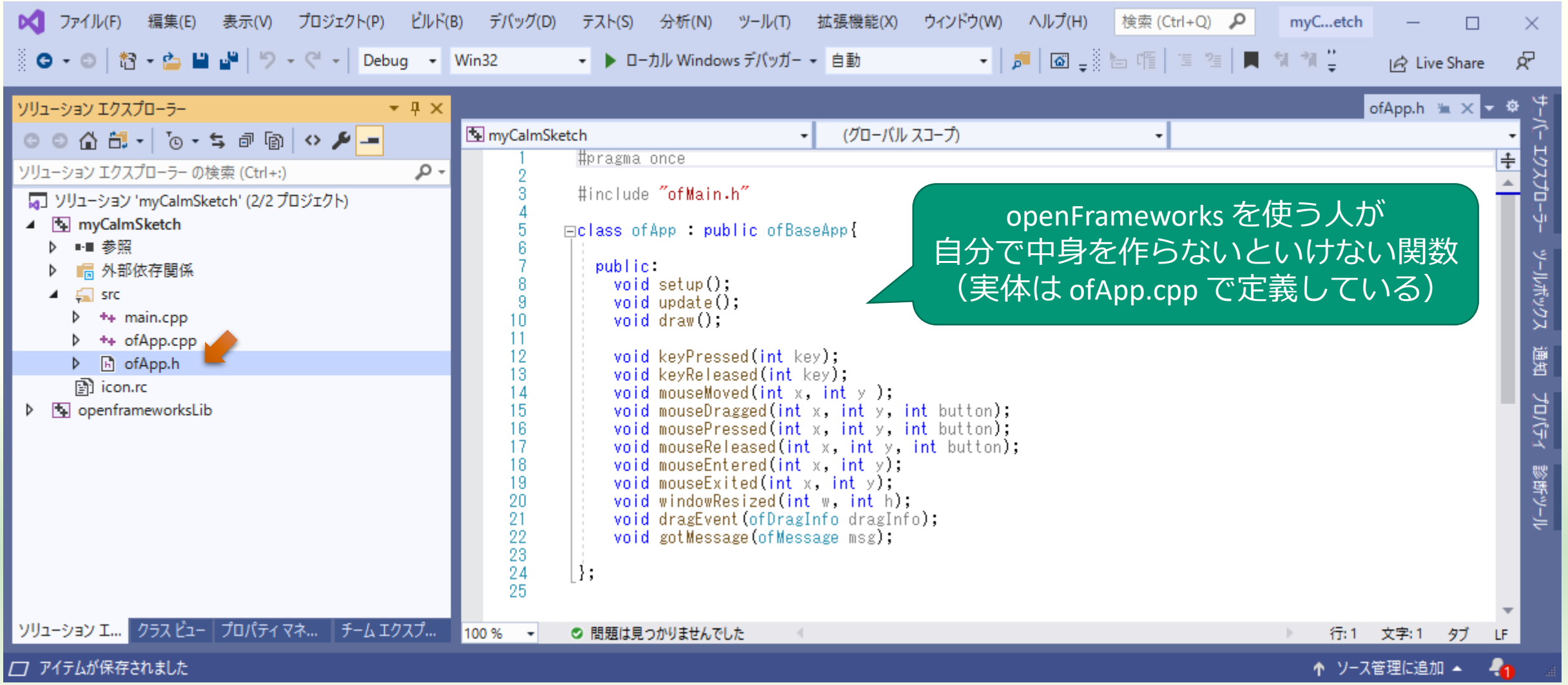
```
const float radius = 30.0f;
float x = 0.0f, y = 0.0f;

//--------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------
void ofApp::update(){
  x += 50.0f;
  y += 80.0f;
}

//--------------------------------------------
void ofApp::draw(){
  ofDrawCircle(x, y, radius);
}
```

- static を付けないで関数の外で宣言した変数は ofApp.cpp 以外にあるどの関数からも使える
  - **大域変数 (グローバル変数)**
  - メモリは静的（プログラム起動時）に確保される

大域変数はできるだけ使用しない

- 大域変数はプログラム全体のどこからも利用できる
  - 意図しないところで変数の内容を変えてしまうことがある
  - 大域変数と同じ名前の変数を使ってしまうことがある
  - 関数同士の依存性が高まってプログラムが複雑になる
    - 異なる関数がどこかにある同じ大域変数を使っていると、一方を変更したときの影響がもう一方に出ることがある
- どうしても大域変数を使わざるを得ない場合はある
  - それ以外は使用を避ける

そこで ofApp.h を開く

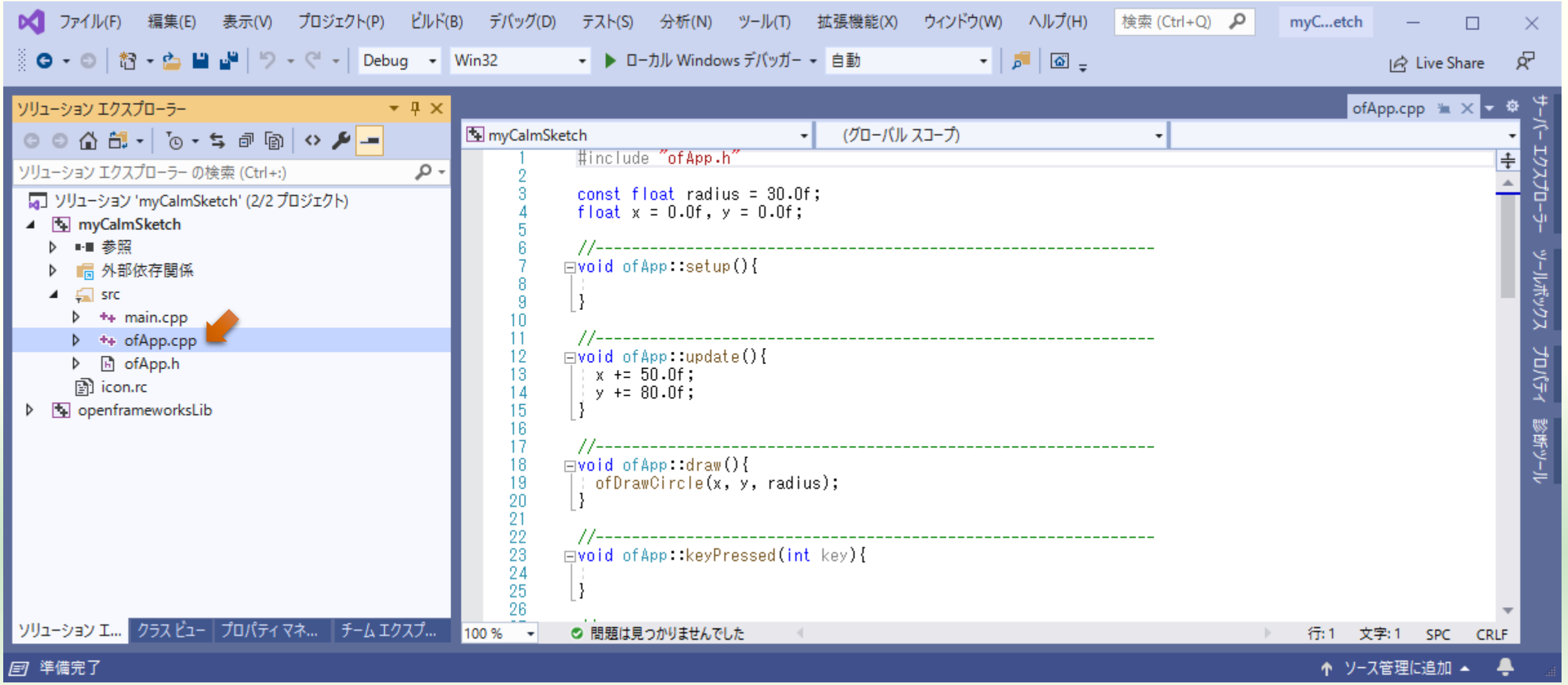


ofApp.h の ofApp クラスで x, y を変数宣言する

```
class ofApp : public ofBaseApp{
  float x, y;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  void mouseDragged(int x, int y, int button);
  void mousePressed(int x, int y, int button);
  void mouseReleased(int x, int y, int button);
  void mouseEntered(int x, int y);
  void mouseExited(int x, int y);
  void windowResized(int w, int h);
  void dragEvent(ofDragInfo dragInfo);
  void gotMessage(ofMessage msg);

};
```

- ofApp.h は **ofApp** という**クラス**を宣言している
  - クラスは自分で定義するデータ型
- x, y を変数宣言する
  - **メンバ変数**
  - 下の setup() 以降の**メンバ関数**で共通に使える
    - これら以外からは使えない

再び ofApp.cpp を開く



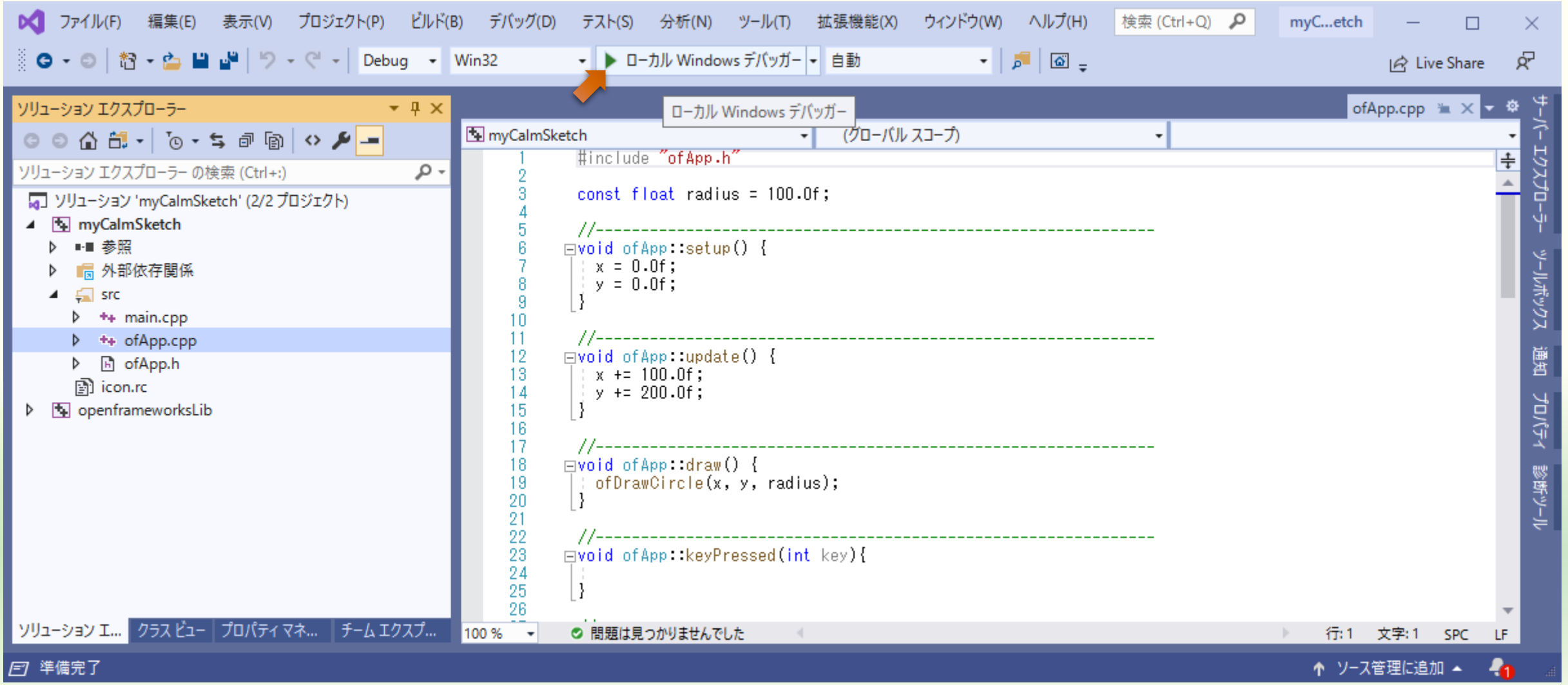
x, y の変数宣言を ofApp.cpp から削除する

```
const float radius = 30.0f;
float x = 0.0f, y = 0.0f;

//--------------------------------------------
void ofApp::setup(){
  x = 0.0f;
  y = 0.0f;
}

//--------------------------------------------
void ofApp::update(){
  x += 50.0f;
  y += 80.0f;
}

//--------------------------------------------
void ofApp::draw(){
  ofDrawCircle(x, y, radius);
}
```

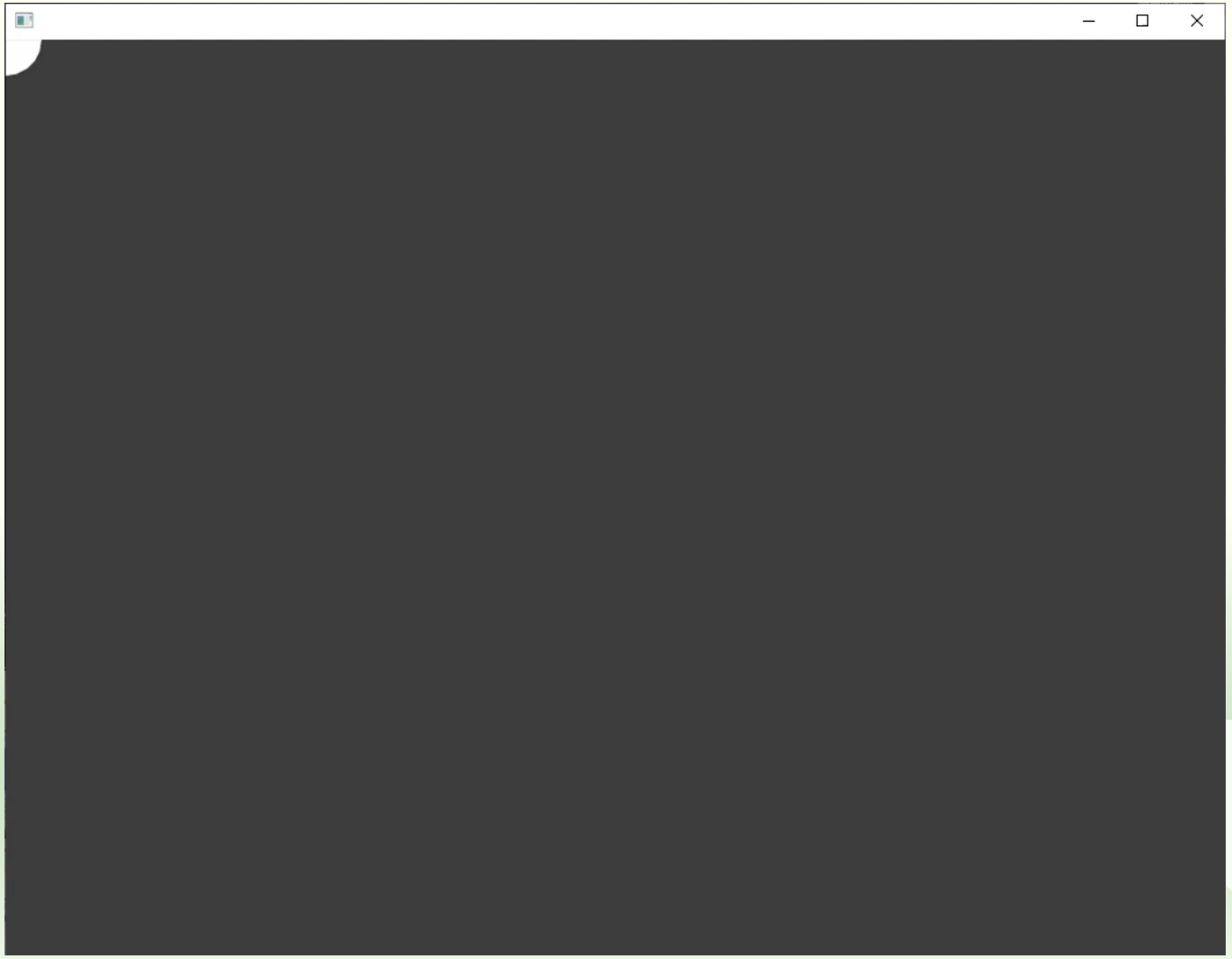
削除

- ofApp.h で初期化は行っていないので setup() で初期値を設定する

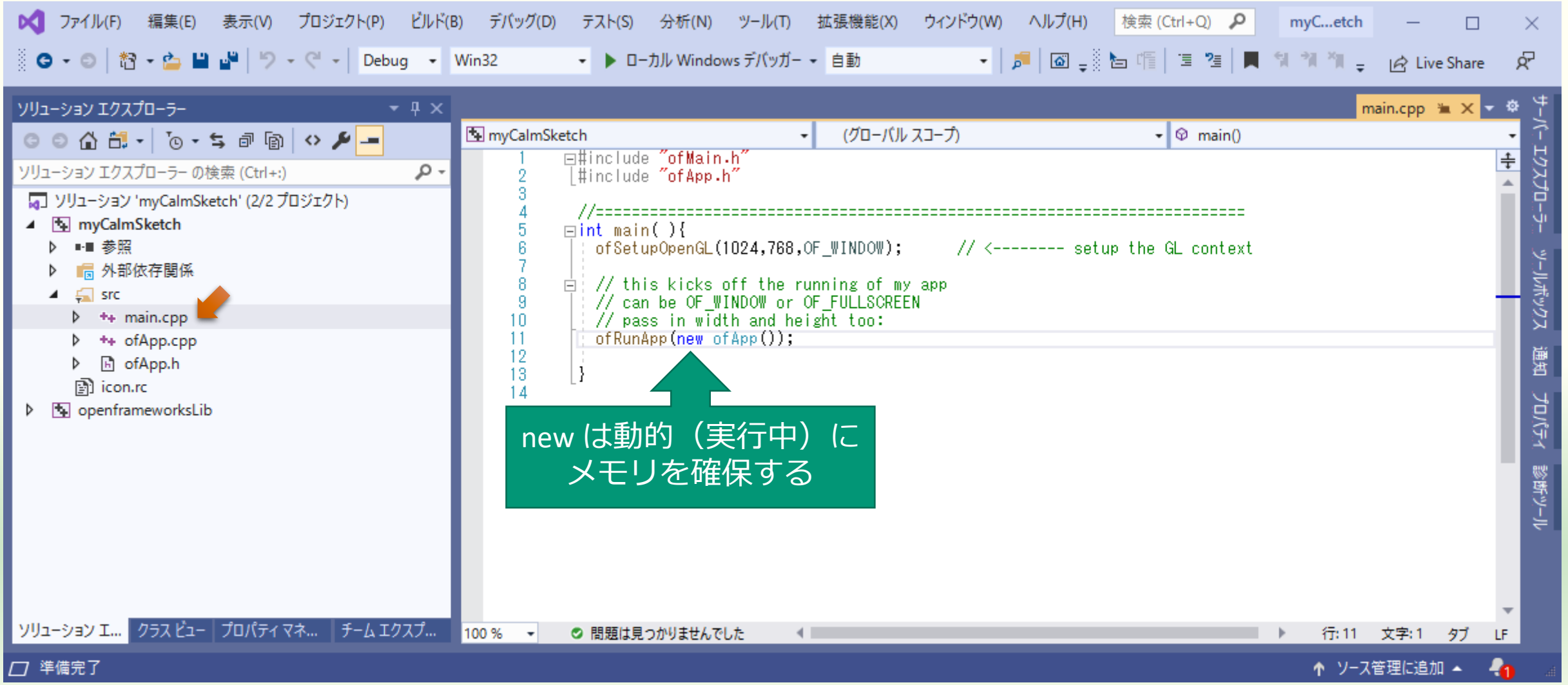
ビルドと実行


ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) テスト(S) 分析(N) ツール(T) 拡張機能(X) ウィンドウ(W) ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myCalmSketch' (2/2 プロジェクト)
myCalmSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.cpp
myCalmSketch
(グローバル スコープ)
#include "ofApp.h"
const float radius = 100.0f;
//--
void ofApp::setup() {
x = 0.0f;
y = 0.0f;
}
//--
void ofApp::update() {
x += 100.0f;
y += 200.0f;
}
//--
void ofApp::draw() {
ofDrawCircle(x, y, radius);
}
//--
void ofApp::keyPressed(int key){
}
100 %
問題は見つかりませんでした
行: 1 文字: 1 SPC LF
準備完了
ソース管理に追加
サーバー エクスプローラー ツールボックス 通知 プロパティ 診断ツール

特に変わらない（やっぱり一瞬で消える）



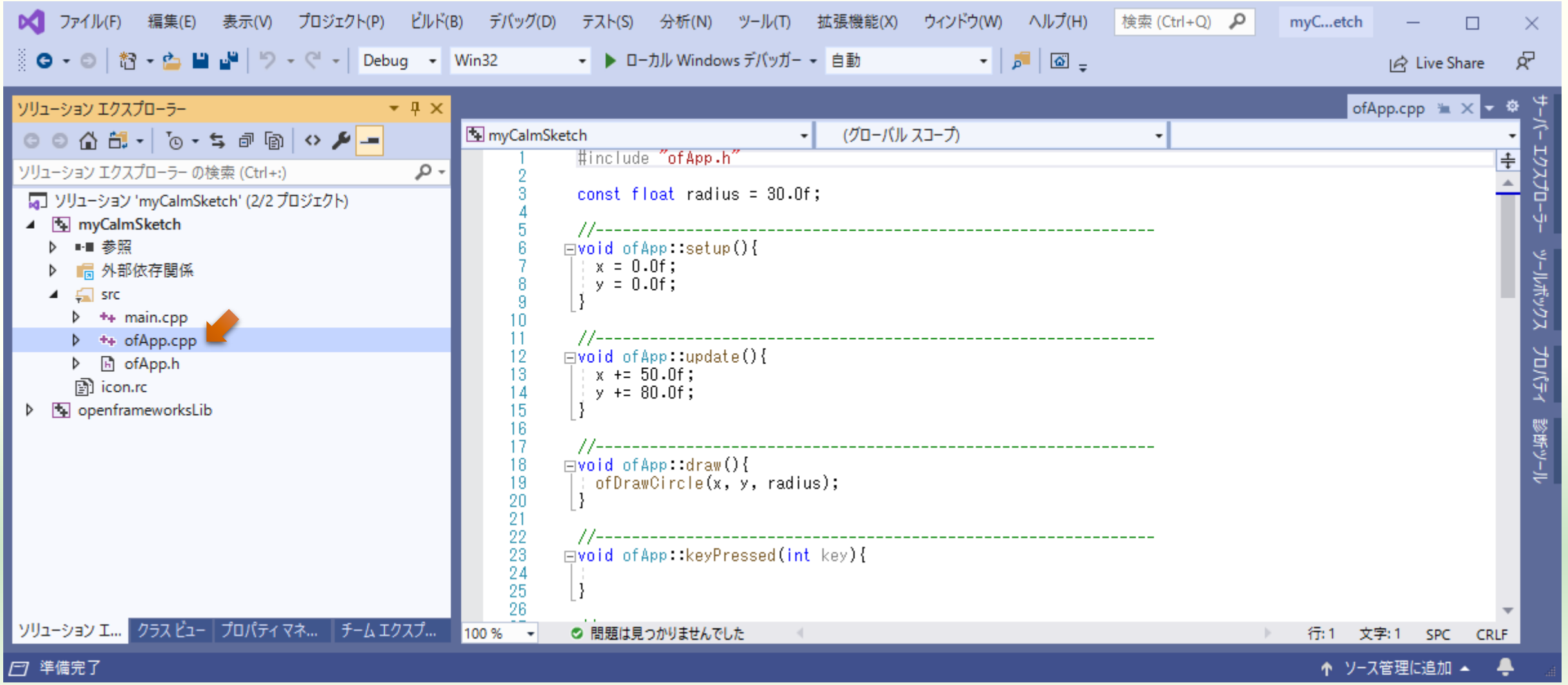
x, y のメモリは main() で確保している



クラスとオブジェクト

- **クラス**
  - ユーザ（プログラムを書く人）が自分で定義するデータ型
  - データを保持する**メンバ変数**と、それを操作する**メンバ関数**（データを操作する**メソッド**）を内包している
- **オブジェクト**
  - クラスによって定義された構造を持つメモリ上のデータ
    - クラスをデータ型とする変数に確保されたメモリ上のデータ
    - クラスをデータ型として動的に確保されたメモリ上のデータ
  - そのクラスの**インスタンス**とも呼ばれる

ofApp.cpp を開く



{ } の内側にある変数宣言が優先される

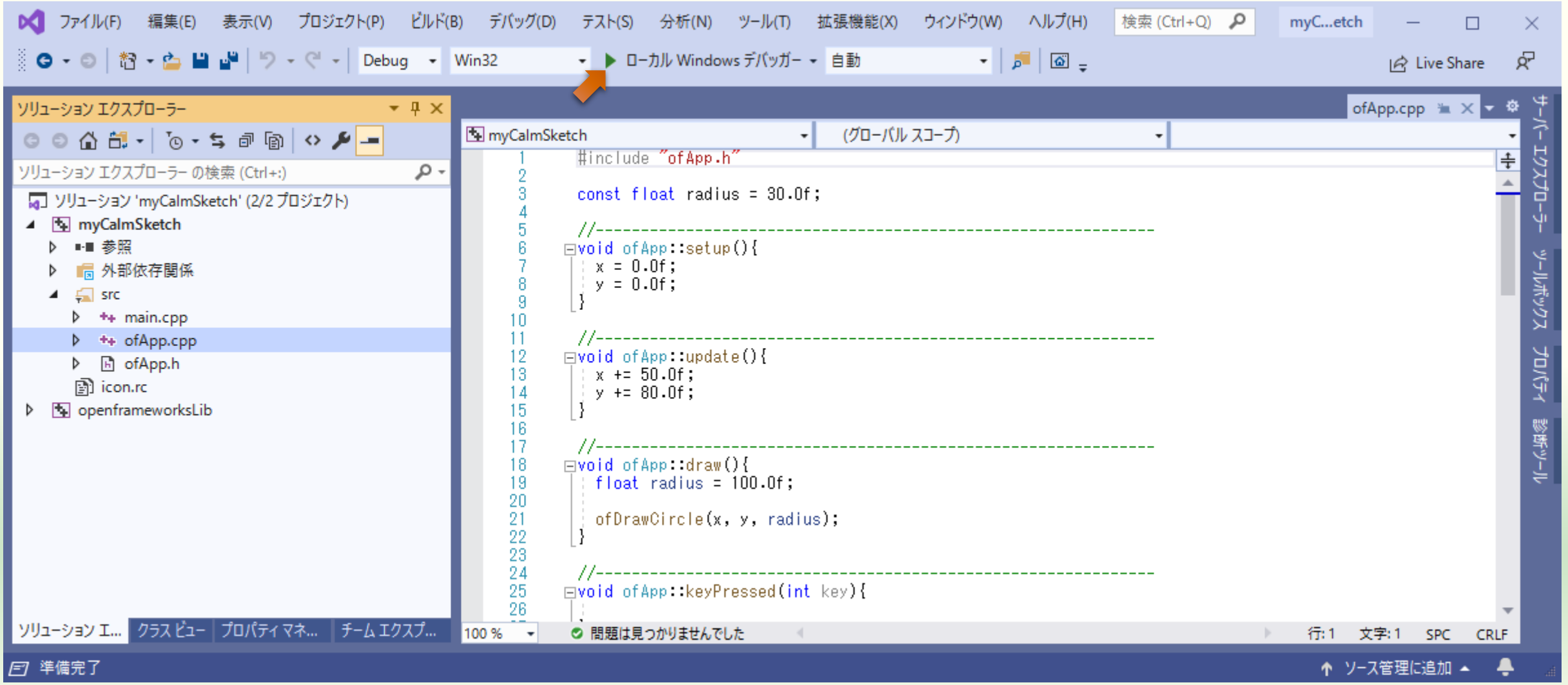
```
const float radius = 30.0f;

//--------------------------------------------
void ofApp::setup(){
  x = 0.0f;
  y = 0.0f;
}

//--------------------------------------------
void ofApp::update(){
  x += 100.0f;
  y += 200.0f;
}

//--------------------------------------------
void ofApp::draw(){
  float radius = 100.0f;

  ofDrawCircle(x, y, radius);
}
```

別物

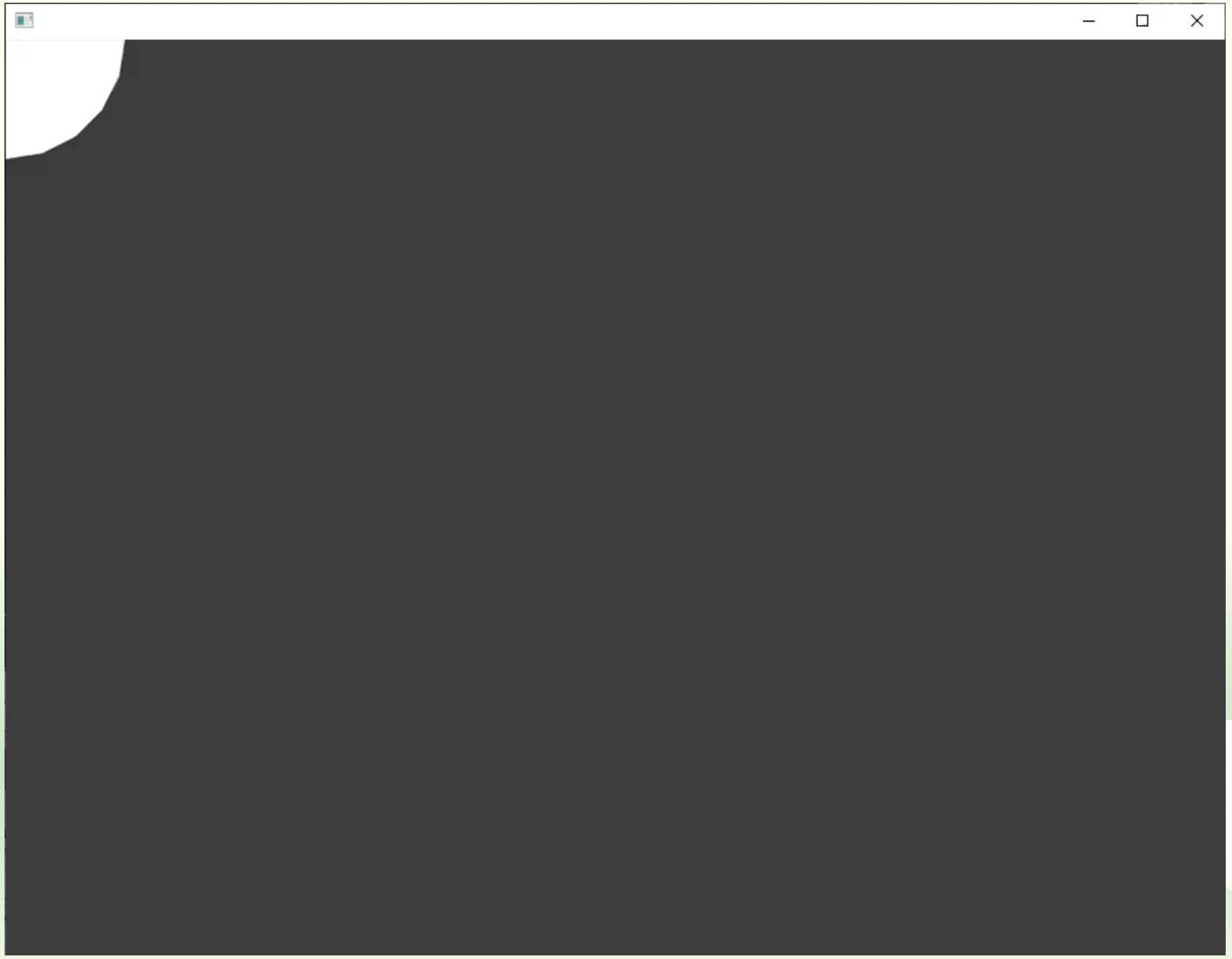
同じ

- { } の外にある変数と同じ変数名の変数を { } 内で変数宣言する
  - 外にある変数が使えなくなる
  - 別の変数になる（データを保存するメモリの場所が異なる）ので互いに影響は与えない

ビルドと実行


ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) テスト(S) 分析(N) ツール(T) 拡張機能(X) ウィンドウ(W) ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myCalmSketch' (2/2 プロジェクト)
myCalmSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.cpp
myCalmSketch
(グローバル スコープ)
#include "ofApp.h"
const float radius = 30.0f;
//--
void ofApp::setup(){
x = 0.0f;
y = 0.0f;
}
//--
void ofApp::update(){
x += 50.0f;
y += 80.0f;
}
//--
void ofApp::draw(){
float radius = 100.0f;
ofDrawCircle(x, y, radius);
}
//--
void ofApp::keyPressed(int key){
100 %
問題は見つかりませんでした
行: 1 文字: 1 SPC CRLF
準備完了
ソース管理に追加

円が大きくなる（しかし一瞬で消える）



変数に関して覚えておいて欲しいこと

- **データ型**
  - 整数型と実数型がある
  - それぞれ精度の異なるデータ型がある
- **代入**
  - 変数の値を変更する
- **初期化**
  - 変数の初期値を設定する
- **スコープ**
  - 局所（ローカル）変数
  - 大域（グローバル）変数
  - メンバ変数
- **記憶クラス**
  - 自動変数
  - 静的変数
- **動的なメモリ確保**

速度を考慮する

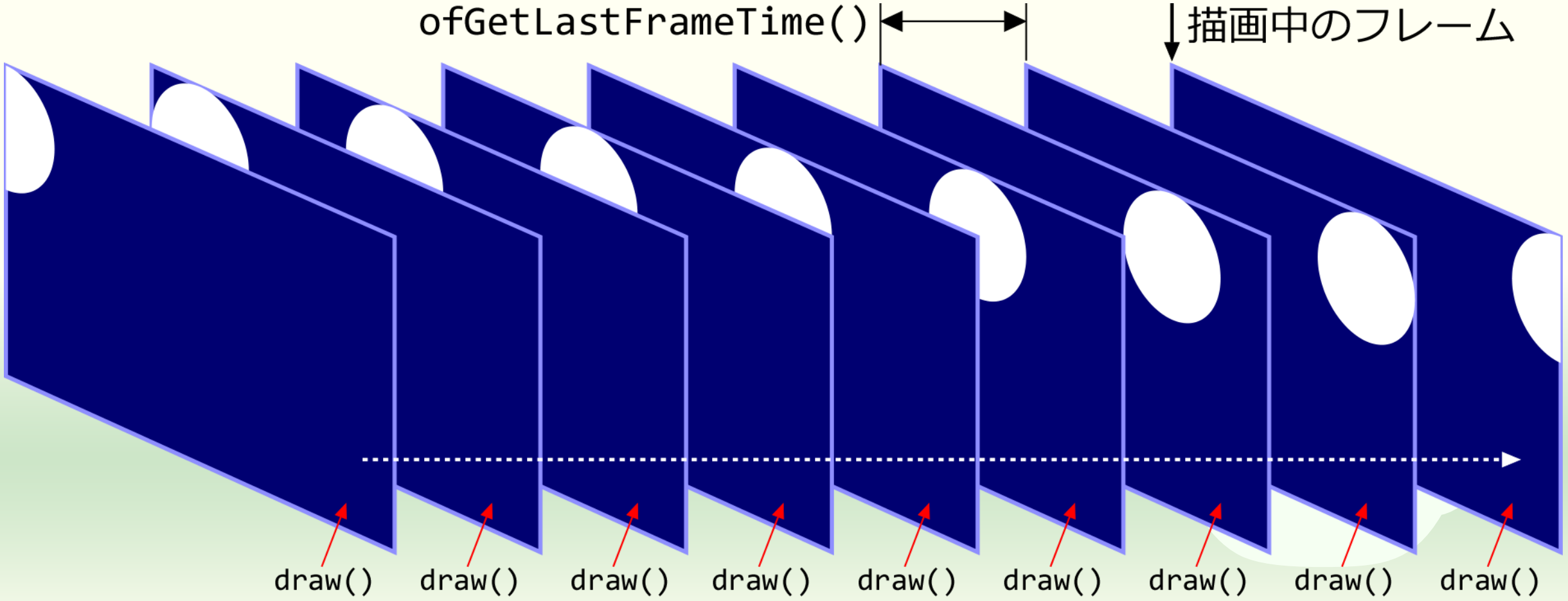
アニメーションの速度を描画時間に依存させない

アニメーション

- パソコンの画面は静止画を**繰り返し表示**している
  - openFrameworks ではそのタイミングで draw() を実行する
- パソコンの１枚の画面表示を**フレーム**という
  - 一般的なディスプレイは１秒間に６０フレーム表示可能
  - １フレームにかかる時間は $1/60 = 0.016666...$ 秒
  - それより速いディスプレイもある
- ofGetLastFrameTime()
  - 直前のフレームの描画にかかった時間を調べる

double ofGetLastFrameTime()

- 直前のフレームの描画にかかった時間を返す



位置に速度と時間の積を加えて更新する

```
const float radius = 30.0f;
const float velocity = 300.0f;

//--------------------------------------------
void ofApp::setup(){
  x = radius;
  y = radius;
}

//--------------------------------------------
void ofApp::update(){
  x += velocity * ofGetLastFrameTime();
  y += 0.0f;
}

//--------------------------------------------
void ofApp::draw(){
  float radius = 200.0f;

  ofDrawCircle(x, y, radius);
}
```

1秒間に300画素移動ということ

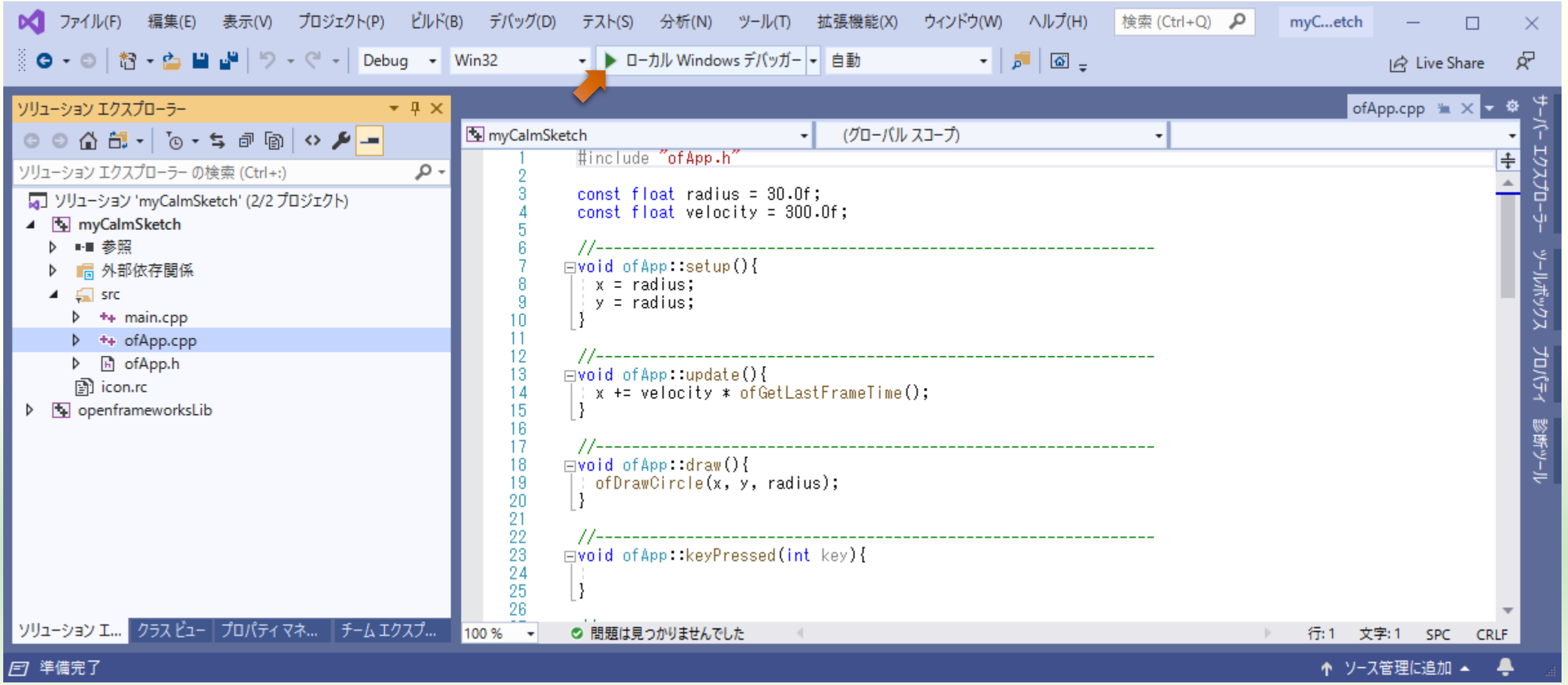
削除

- 現在位置x, 次の位置x', 速度v, 経過時間Δt

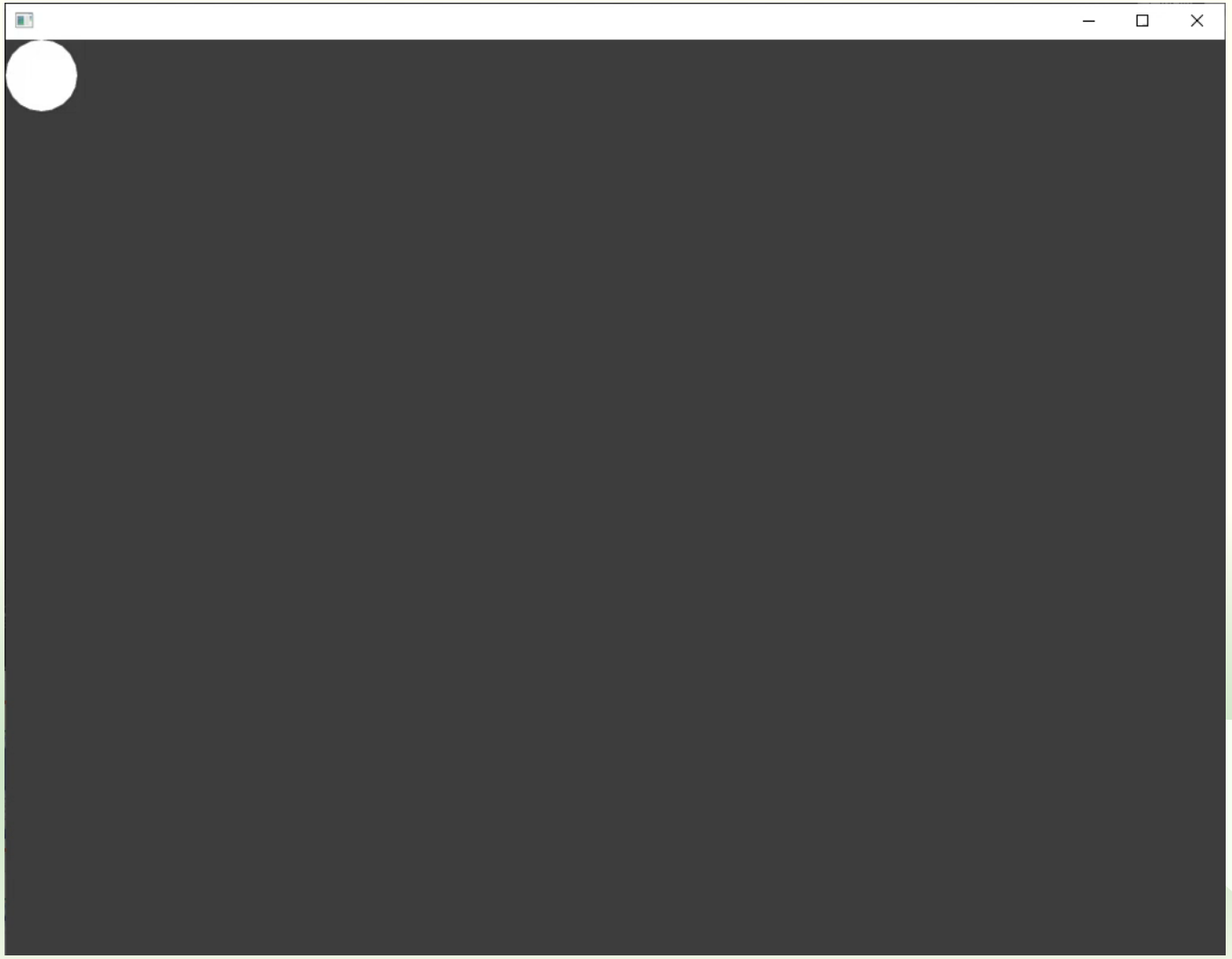
$$x' = x + v\Delta t$$

- Δt: ofGetLastFrameTime()
- y 方向には移動しない
- x, y の初期値を円の半径にしておけば起点でウィンドウからはみ出ない

ビルドと実行


ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) テスト(S) 分析(N) ツール(T) 拡張機能(X) ウィンドウ(W) ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myCalmSketch' (2/2 プロジェクト)
myCalmSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.cpp
myCalmSketch
(グローバル スコープ)
#include "ofApp.h"
const float radius = 30.0f;
const float velocity = 300.0f;
//--
void ofApp::setup(){
x = radius;
y = radius;
}
//--
void ofApp::update(){
x += velocity * ofGetLastFrameTime();
}
//--
void ofApp::draw(){
ofDrawCircle(x, y, radius);
}
//--
void ofApp::keyPressed(int key){
}
100 %
問題は見つかりませんでした
行: 1 文字: 1 SPC CRLF
ソリューション エ... クラス ビュー プロパティ マネ... チーム エクスプ...
サーバー エクスプローラー ツールボックス プロパティ 診断ツール
準備完了
ソース管理に追加

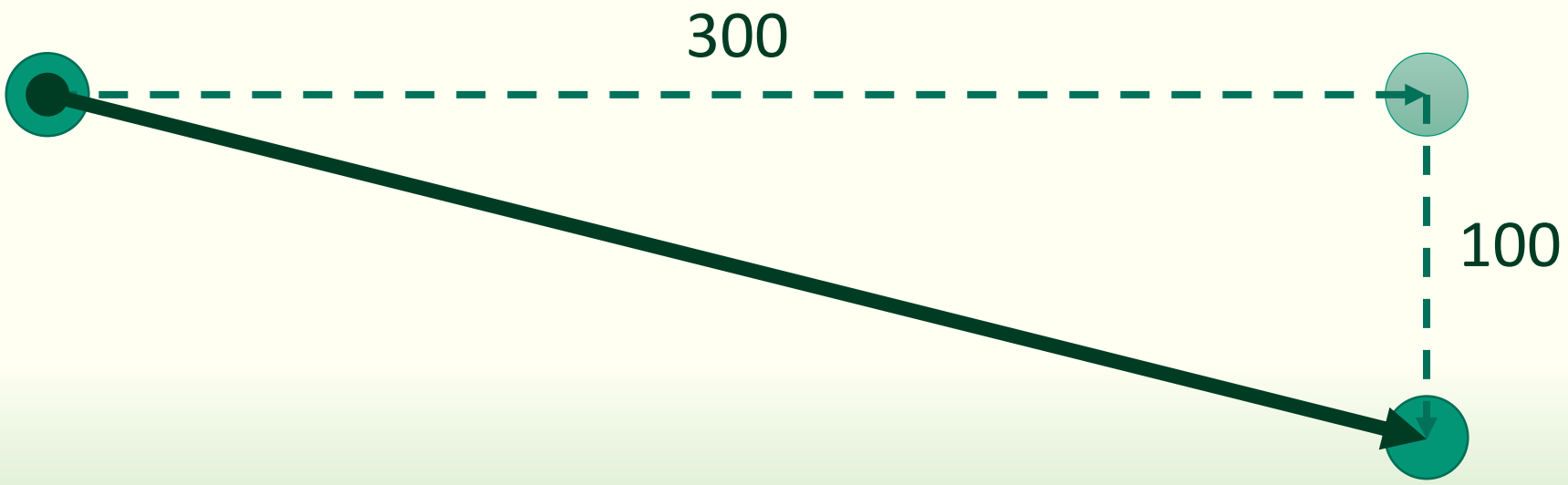
横に動く



課題２－１

斜めに動かしてみる

y 方向の速度を 100画素/秒に設定しなさい



ofApp.cpp を修正して「ビルドと実行」しなさい

ベクトルを使う

線形代数のアレ

ofDrawCircle() のマニュアル

global functions

- ofBackground()
- ofBackgroundGradient()
- ofBackgroundHex()
- ofBeginSaveScreenAsPDF()
- ofBeginSaveScreenAsSVG()
- ofBeginShape()
- ofBezierVertex()
- ofClear()
- ofClearAlpha()
- ofCurveVertex()
- ofCurveVertices()
- ofDisableAlphaBlending()
- ofDisableAntiAliasing()
- ofDisableBlendMode()
- ofDisableDepthTest()
- ofDisablePointSprites()
- ofDisableSmoothing()
- ofDrawBezier()
- ofDrawBitmapString()
- ofDrawBitmapStringHighlight()
- ofDrawCircle()

ofDrawCircle(...)

```
void ofDrawCircle(const glm::vec3 &p, float radius)
```

ofDrawCircle(...)

```
void ofDrawCircle(const glm::vec2 &p, float radius)
```

ofDrawCircle(...)

```
void ofDrawCircle(float x, float y, float radius)
```

Documentation from code comments

Draws a circle, centered at x,y, with a given radius.

```
void ofApp::draw(){
    ofDrawCircle(150,150,100);
}
```

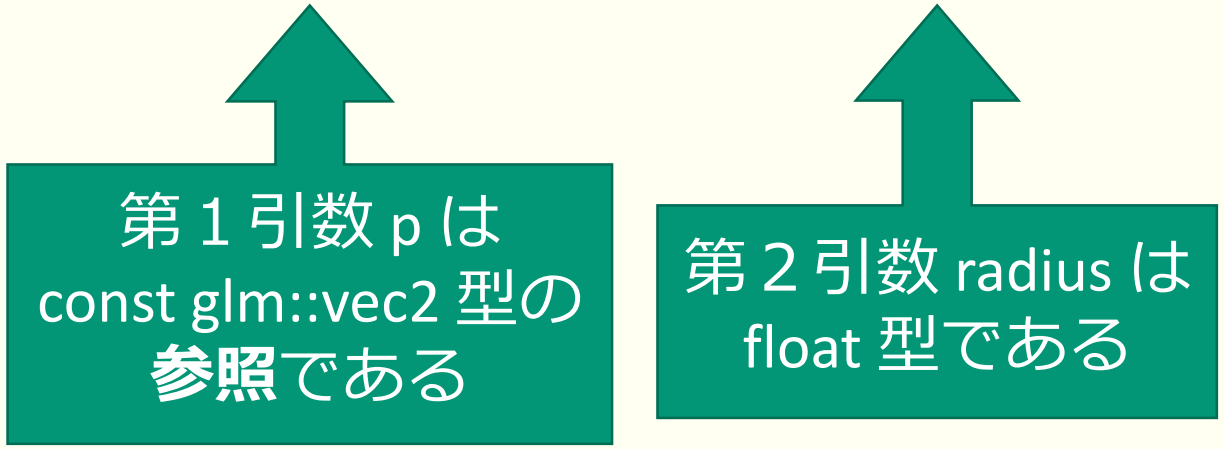
Please keep in mind that drawing circle with different outline color and fill requires calling ofNoFill and ofSetColor for drawing stroke and ofFill and again ofSetColor for filled solid color circle.

ofDrawCircle(...)

```
void ofDrawCircle(float x, float y, float z, float radius)
```

ofDrawCircle() のもう一つの宣言

- **void ofDrawCircle(const glm::vec2 &p, float radius)**



マニュアル

https://openframeworks.cc//documentation/graphics/ofGraphics/#!show_ofDrawCircle

スコープ解決演算子 ::

- プログラムはいろんなライブラリを組み合わせて作る
  - 他の誰かが作ったものを使う
  - 他の誰かが同じ変数名や関数名を使っているかもしれない
  - 変数名や関数名が同じだと使えない（**名前の衝突**）
- **名前空間**
  - 変数や関数のグループの外側に付けた名前
  - 名前空間を変えれば変数名や関数名が同じでも衝突しない
  - std:: や glm:: のようにして名前空間を指定する

glm

- **OpenGL Mathematics (GLM)** の名前空間
  - https://glm.g-truc.net/
  - OpenGL Shading Language (GLSL) というグラフィックス用のプログラミング言語の仕様に倣った数学ライブラリ
  - openFrameworks はグラフィックス表示に OpenGL というライブラリを使っている
- **std** は標準ライブラリの名前空間

glm::vec2

- ２次元のベクトルのクラス

```
glm::vec2 p;                          // vec2 型の変数 p の宣言
p.x = 10.0f;                          // p の x 要素への代入
p.y = 20.0f;                          // p の y 要素への代入
p = vec2{ 10.0f, 20.0f };             // p の x, y 要素への代入
p = vec2{ 0.0f };                     // p の x, y 要素への代入
p += vec2{ 10.0f, 20.0f };            // p の x, y 要素への加算
p += vec2{ 10.0f };                   // p の x, y 要素への加算
p += 10.0f;                           // p の x, y 要素への加算
glm::vec2 q{ 3.0f, 4.0f };            // q の x, y を初期化
glm::vec2 q{ 0.0f };                  // q の x, y を初期化
```

参照演算子 &

引数の値渡し

```
#include <iostream>

int sub(int x)
{
  x = 10;
}

int main()
{
  int x = 0;

  std::cout << x;   // 0 が出力される

  sub(x);

  std::cout << x;   // 0 が出力される
}
```

値 (0) がコピーされる
（使われるメモリが異なる）

引数の参照渡し

```
#include <iostream>

int sub(int &x)
{
  x = 10;
}

int main()
{
  int x = 0;

  std::cout << x;   // 0 が出力される

  sub(x);

  std::cout << x;   // 10 が出力される
}
```

同じものになる
（使われるメモリが同じ）

参照演算子 & と const

const 変数を参照渡しするとエラー

```
#include <iostream>

int sub(int &x)
{
  x = 10;
}

int main()
{
  const int x = 0;

  std::cout << x;   // 0 が出力される

  sub(x);

  (以下略)
}
```

参照渡しの仮引数を変更すると実引数が変更される

エラー

仮引数を const にした参照渡し

```
#include <iostream>

int sub(const int &x)
{
  std::cout << x;
}

int main()
{
  const int x = 0;

  std::cout << x;   // 0 が出力される

  sub(x);

  (以下略)
}
```

const を付けた仮引数は変更できない

この関数 sub() は実引数 x を変更しないことを保証している

ofApp クラスのメンバ x, y をベクトルにする

```
#pragma once

#include "ofMain.h"

using namespace glm;

class ofApp : public ofBaseApp{
    vec2 position;

public:
    void setup();
    void update();
    void draw();

   (以下略)
```

- float x, y; を vec2 position; に変更する
- **using namespace glm;** を先に書いておくと glm:: を省略できる

ofApp.cpp を変更する

```
const float radius = 30.0f;
const vec2 velocity{ 300.0f, 200.0f };

//--------------------------------------------
void ofApp::setup(){
  position = vec2{ radius };
}

//--------------------------------------------
void ofApp::update(){
  position += velocity * ofGetLastFrameTime();
}

//--------------------------------------------
void ofApp::draw(){
  ofDrawCircle(position, radius);
}
```

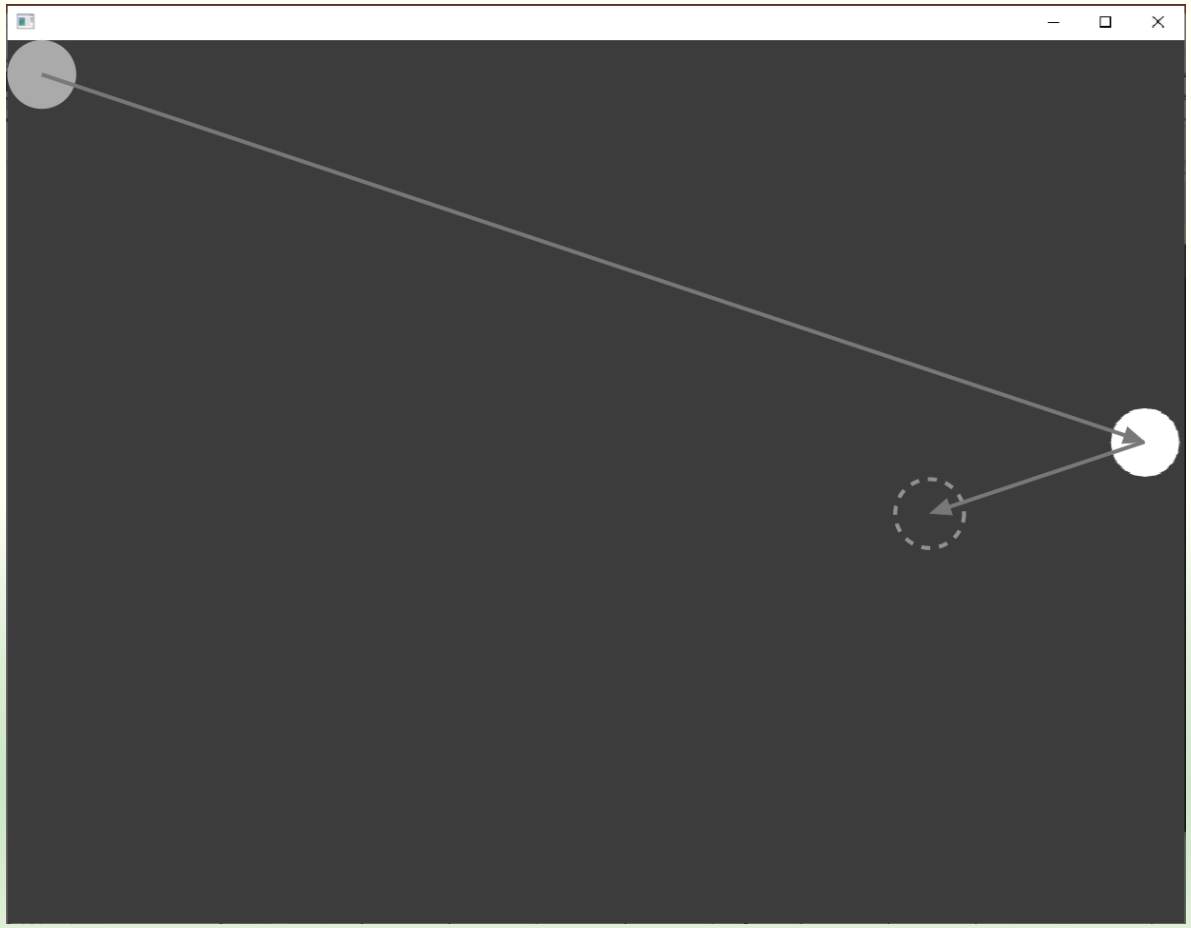
「ビルドと実行」

- velocity に速度を設定する
- position の x, y の両方の要素に radius を初期値として代入する
- position に velocity と経過時間の積を加えて円の位置を更新する
- position の位置に半径 radius の円を描く

課題２－２

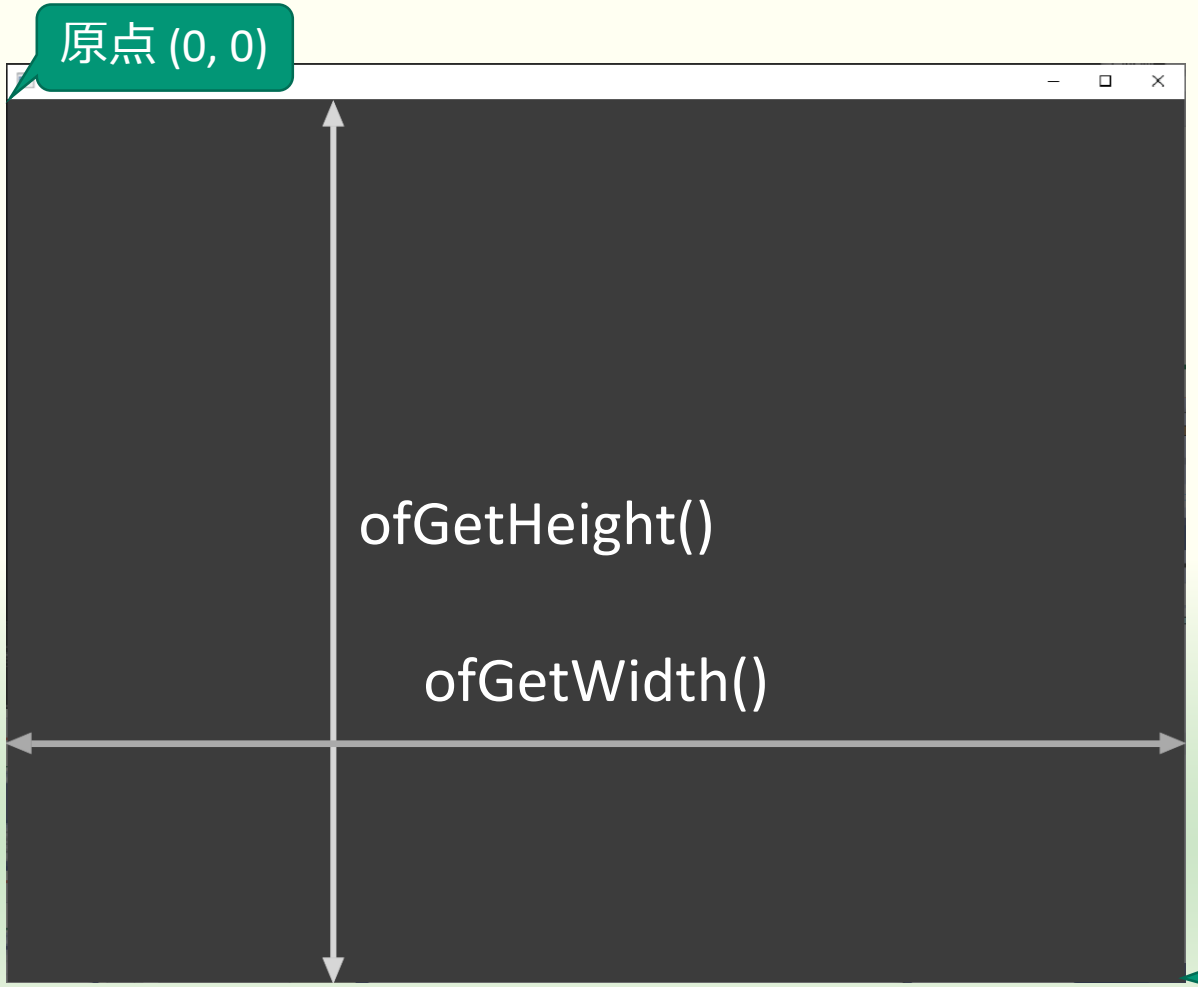
跳ね返らせてみる

円を壁（ウィンドウの端）で跳ね返らせる



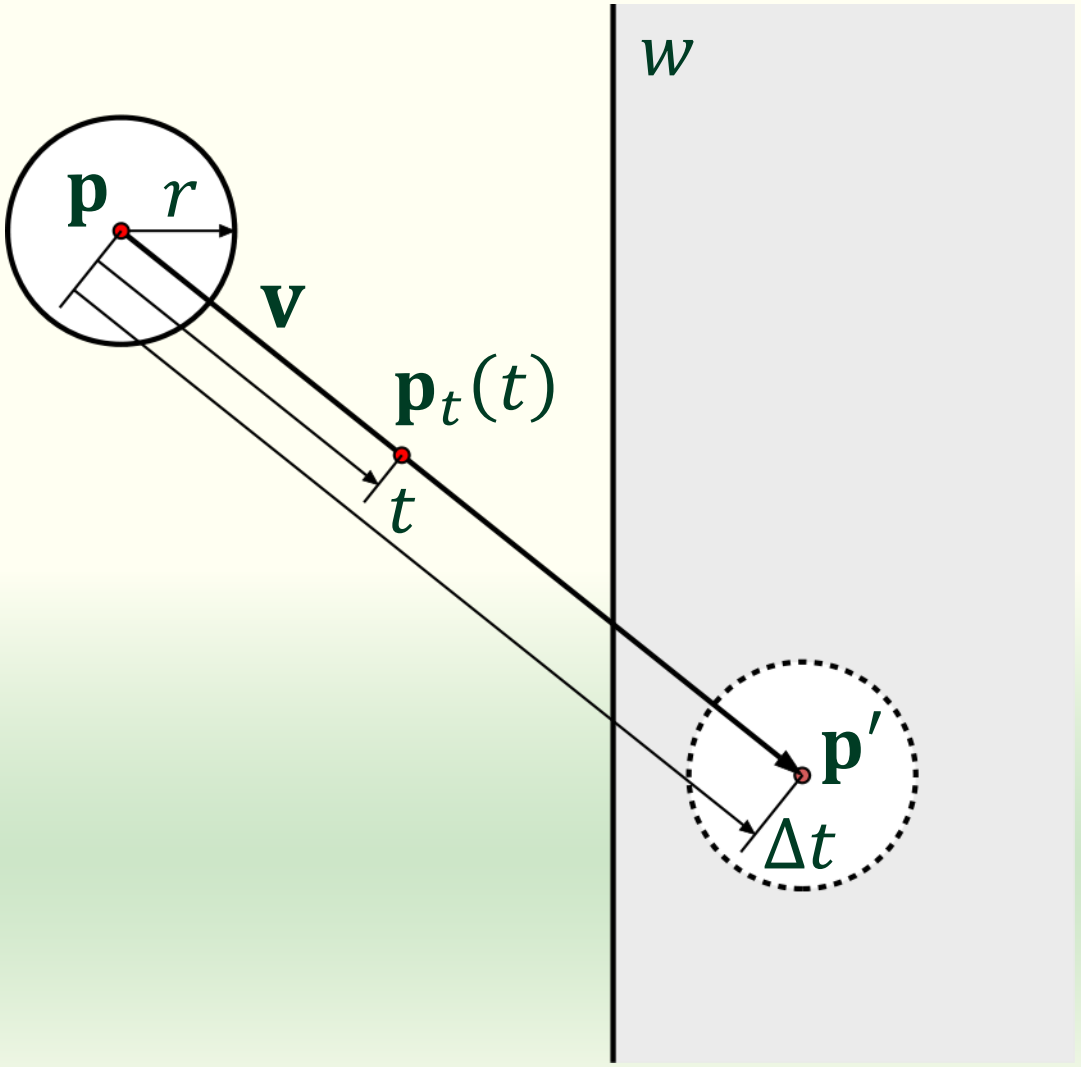
- 円がウィンドウから飛び出して帰ってこない
- 円が壁に当たったら跳ね返るようにする

openFrameworks のウィンドウの座標系



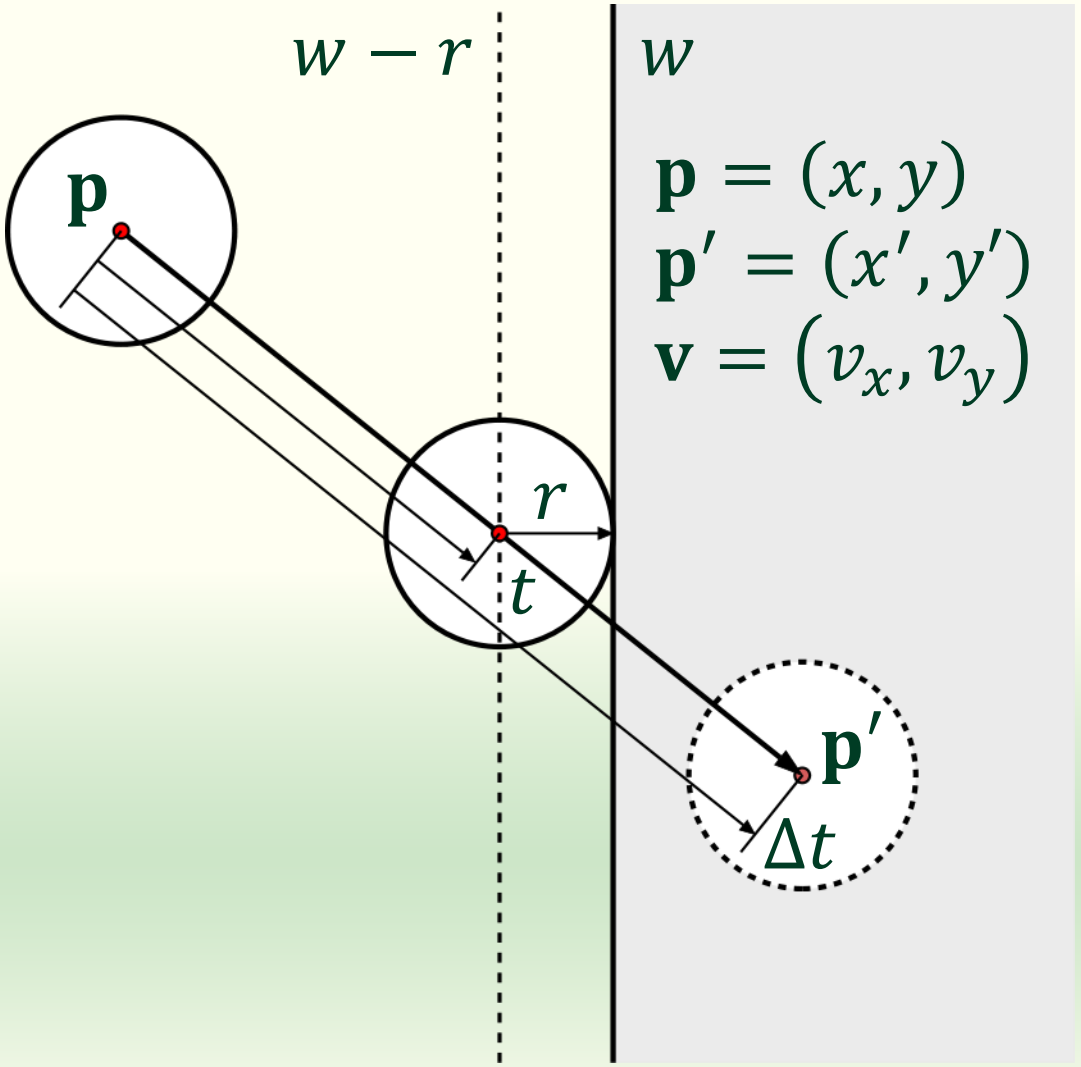
- int ofGetWidth()
  - ofApp() のウィンドウの幅を得る
- int ofGetHeight()
  - ofApp() のウィンドウの高さを得る

円の移動



- 円の移動
  - $\mathbf{p}_t(t) = \mathbf{p} + \mathbf{v}t$
    - $\mathbf{p}_t(\Delta t) = \mathbf{p}'$
- 変数
  - $\mathbf{p}$: position, $\mathbf{v}$: velocity
  - Δt: ofGetLastFrameTime()
  - w: ofGetWidth() - 1
  - r: radius

円が壁に衝突した時刻



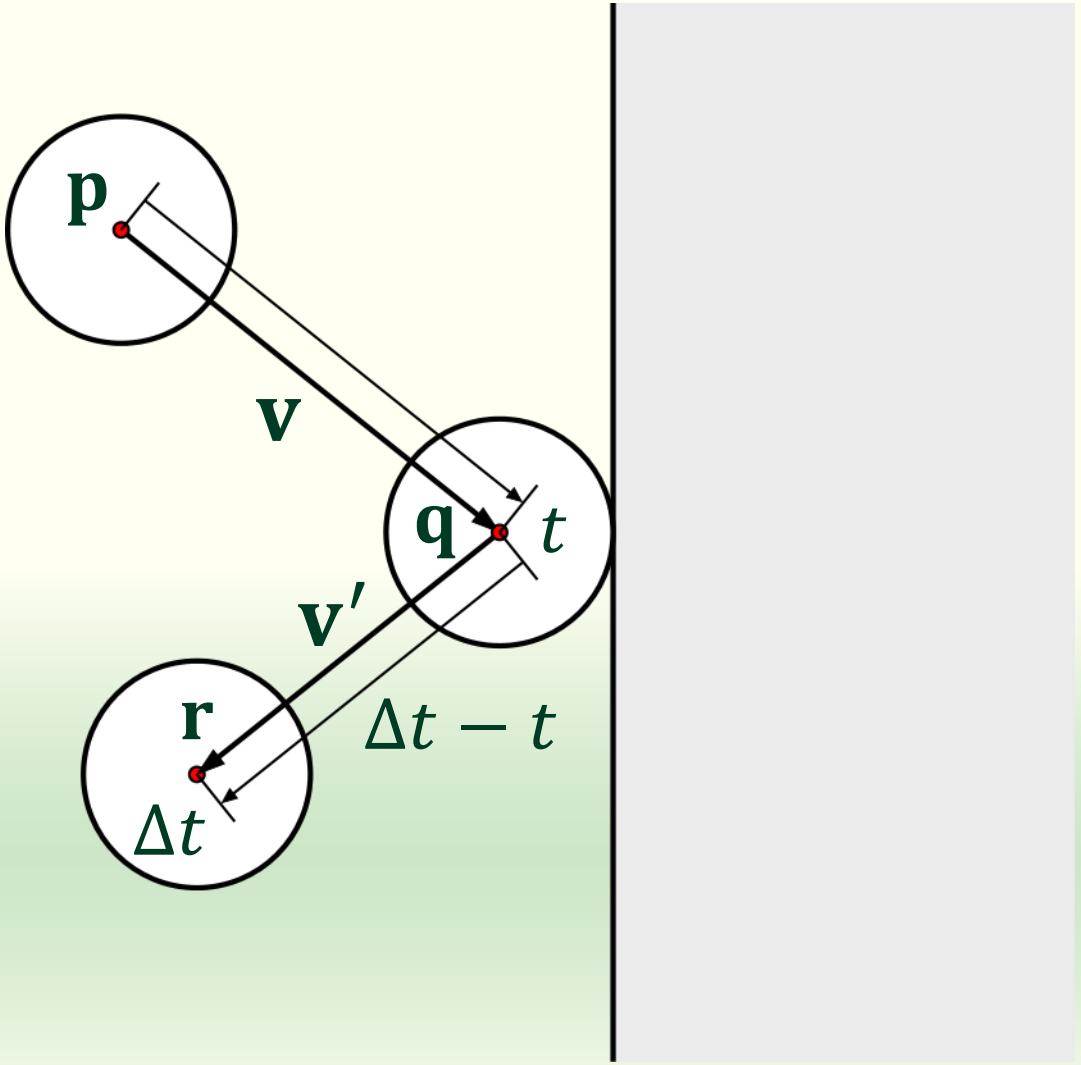
- 現在時刻から Δt 後の円の位置が $x' \geq w-r$ なら壁と衝突している
- そのとき衝突した時刻は

$$x + v_x t = w - r$$

より

$$t = \frac{w-r-x}{v_x}$$

衝突後の円の位置



- 衝突位置

$$\mathbf{q} = (w - r, y + v_y t)$$

- 衝突後の円の速度

$$\mathbf{v}' = (-v_x, v_y)$$

- 円の Δt 後の位置

$$\mathbf{r} = \mathbf{q} + \mathbf{v}'(\Delta t - t)$$

ofApp クラスで velocity を変数宣言する

```
class ofApp : public ofBaseApp{
  vec2 position, velocity;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y );
  void mouseDragged(int x, int y, int button);
  void mousePressed(int x, int y, int button);
  void mouseReleased(int x, int y, int button);
  void mouseEntered(int x, int y);
  void mouseExited(int x, int y);
  void windowResized(int w, int h);
  void dragEvent(ofDragInfo dragInfo);
  void gotMessage(ofMessage msg);

};
```

- velocity を ofApp クラスのメンバ変数にする

setup() の変更

```
const float radius = 30.0f;
const vec2 velocity{ 300.0f, 100.0f };

//--------------------------------------------
void ofApp::setup(){
  position = vec2{ radius };
  velocity = vec2{ 300.0f, 100.0f };
}
```

削除

- velocity はメンバ変数にしたので削除する
- setup() で velocity に初期値を設定する

update() の変更

```
void ofApp::update(){

  // 直前のフレームの時間間隔を保存しておく
  const float dt = ofGetLastFrameTime();

  // 現在の円の位置を記録しておく
  const vec2 p = position;

  // 円の位置を更新する
  position += velocity * dt;
```

変数の値を変更しないなら変数宣言に **const** を付ける

```
  // w - r を計算する
  const float wr = ofGetWidth() - 1.0f - radius;

  // もし円がウィンドウからはみ出たら
  if (position.x >= wr){

    // 壁と衝突した時刻を求める
    const float t = (wr - p.x) / velocity.x;

    // 衝突した位置を求める
    const vec2 q = vec2{wr, p.y + velocity.y * t};

    // 衝突後の速度を変更する
    velocity = vec2{-velocity.x, velocity.y};

    // 円の位置を変更する
    position = q + velocity * (dt - t);
  }
}
```

w: ofGetWidth() - 1

円がウィンドウからはみ出たら速度 velocity と位置 position を変更する

if (position.x >= wr){ … }

```
if (position.x >= wr){

  // この部分は
  // position.x ≧ wr
  // のときのみ実行される

}
```

- もし position.x ≧ wr なら { … } 内の処理を行う
- **条件分岐**
  - if 文

関係演算

x > y	x が y より大きいなら true
x >= y	x が y 以上なら true
x < y	x が y より小さいなら true
x <= y	x が y 以下なら true
x == y	x と y が等しいなら true
x != y	x と y が等しくないなら true

- 数値の比較を行う演算式
- この比較を行う演算子を**関係演算子**という
- 演算の結果は **true** (真) または **false** (偽) の論理値

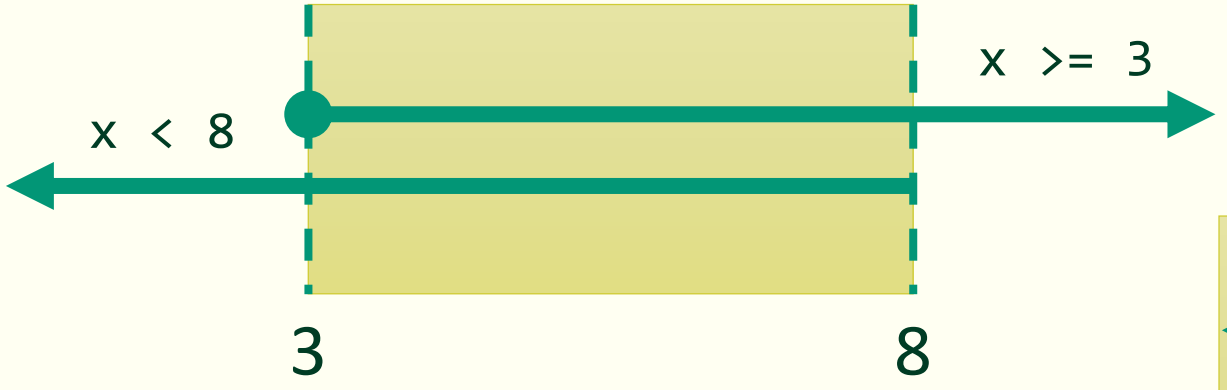
論理演算

x && y	x が y ともに true のとき true
x || y	x と y のいずれか一方が true のとき真
!x	x が false なら true、true なら false （論理反転）

- 論理値 **true** と **false** を対象にした演算式
  - x, y が数値なら 0 は false、0 以外は true として扱う
- この演算を行う演算子を論理演算子という
- 演算の結果は true または false

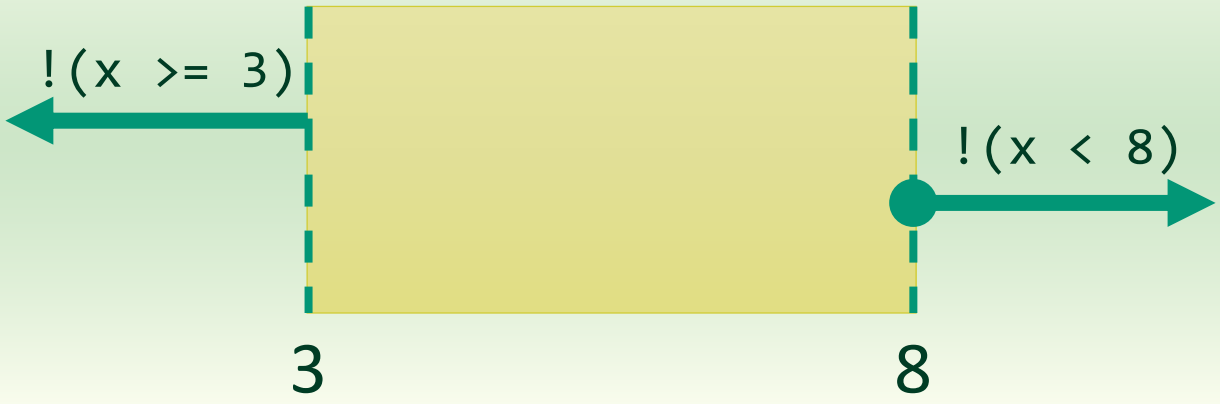
関係演算と論理演算の組み合わせ

```
x >= 3 && x < 8
```




```
!(x >= 3) || !(x < 8)
```


```
!(!(x >= 3) || !(x < 8))
```



条件分岐（if 文）

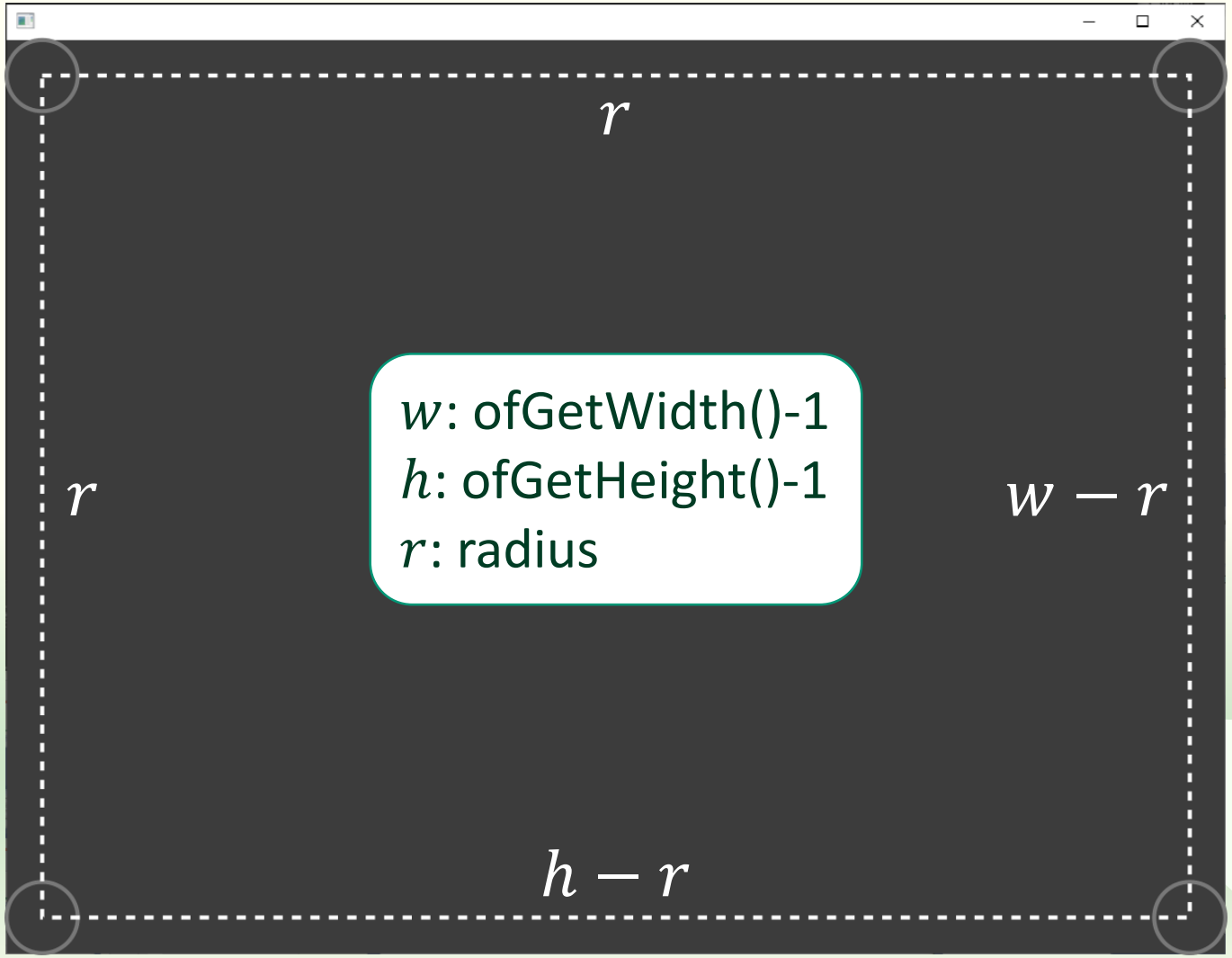
```
if (条件) {
  <条件が true なら実行>
}

if (条件) {
  <条件が true なら実行>
}
else {
  <条件が true でないなら実行>
}

if (条件1) {
  <条件1が true なら実行>
}
else if (条件2) {
  <条件1が true でなく条件2が true なら実行>
}
else {
  <条件1も条件2も true でないなら実行>
}
```

```
if (条件1) {
  <条件1が true なら実行>

  if (条件2) {
    <条件1が true で条件2も true なら実行>
  }
  else {
    <条件1が true で条件2が false なら実行>
  }
}
else {
  if (条件3) {
    <条件1が false で条件3が true なら実行>
  }
  else {
    <条件1が false で条件3が false なら実行>
  }
}
```

ソースプログラムの式と対応する数式

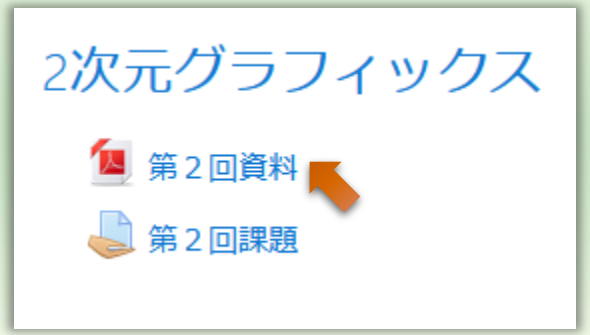
- `const float t = (wr - p.x) / velocity.x;`
  - $t = \frac{w-r-x}{v_x}$
- `vec2 q = vec2{ wr, p.y + velocity.y * t };`
  - $\mathbf{q} = (w - r, y + v_y t)$
- `velocity = vec2{ -velocity.x, velocity.y };`
  - $\mathbf{v}' = (-v_x, v_y)$
- `position = q + velocity * (dt - t);`
  - $\mathbf{r} = \mathbf{q} + \mathbf{v}'(\Delta t - t)$

他の壁に対しても跳ね返るようにしなさい



動画キャプチャのアップロード

- 作成したプログラムの実行中のウィンドウを**５秒以内**で動画キャプチャして、**2-2.mp4** というファイル名で Moodle の第２回課題にアップロードしてください
  - 画面上の動画は「Windows キー」＋「G」でキャプチャできます（macOS では QuickTime Player の「新規画面収録」が使えます）
  - 動画のキャプチャができないときはスクリーンショットを撮って 2-2.png というファイル名でアップロードしてください



課題２－３

発射位置を指定する

マウスをクリックした位置から発射する

- void ofApp::mousePressed(int x, int y, int button)
  - マウスのボタンを押したときに呼ばれる
  - x, y にはマウスのボタンを押した位置が入っている
  - button は押されたボタンの番号で 0:左, 1:中央, 2:右
- 速度 velocity の**初期値**は 0 にしておく
- マウスボタンを押したとき
  - マウスボタンを押した位置を円の位置 position に設定する
  - 速度 velocity に 0 でない値を設定する

課題２－４

発射方向を指定する

マウスボタンを離したときに発射する

- void ofApp::mouseReleased(int x, int y, int button)
  - マウスのボタンを離したときに呼ばれる
  - x, y にはマウスのボタンを押した位置が入っている
  - button は押していたボタンの番号で 0:左, 1:中央, 2:右
- マウスボタンを押したとき
  - マウスボタンを押した位置を覚えておく
  - 速度 velocity を **0 に設定**する
- マウスボタンを離したとき
  - マウスボタンを離した位置から押した位置に向かうベクトルを速度に設定する（定数倍してもよい）

マウスのドラッグ中にマウスで円を動かす

- void ofApp::mouseDragged(int x, int y, int button)
  - マウスをドラッグすると呼ばれる
  - x, y はマウスを移動した先の位置が入る
  - button は押されているボタンの番号で 0:左, 1:中央, 2:右
- ドラッグ中のマウスの位置 x, y を position に設定すると円がマウスに追従して動く
  - このときマウスの速度 velocity が 0 になってないとマウスを止めたときに円が勝手にマウスカーソルから離れていく

課題２－５

重力を与えてみる

重力

- ウィンドウの下の方向（y の正の方向）に重力がかかっているとして円の位置を更新しなさい
  - 重力加速度を $\mathbf{g} = (0, 200)$ とする
- 速度は加速度によって変化する
  - Δt 後の速度を $\mathbf{v}' = \mathbf{v} + \mathbf{g}\Delta t$ により求める
  - これはマウスボタンを**押していないときだけ**実行する
- この速度を使って位置を更新する
  - Δt 後の位置を $\mathbf{p}' = \mathbf{p} + \mathbf{v}'\Delta t$ により求める

bool ofGetMousePressed(int button)

- マウスボタンが押されている間 true を返す

```
if (ofGetMousePressed()) {
  // いずれかのボタンを押していればここの処理が実行される
}
```

```
if (!ofGetMousePressed(2)) {
  // 右ボタンを押していなければここの処理が実行される
  // 0: 左，1: 中央，2 右
}
```

課題のアップロード

- 作成したプログラムの実行中のウィンドウを**５秒以内**で動画キャプチャして、 **2-5.mp4** というファイル名で Moodle の第２回課題にアップロードしてください
  - 動画のキャプチャができないときはスクリーンショットを撮って 2-5.png というファイル名でアップロードしてください
- ソースプログラム **ofApp.h** と **ofApp.cpp** を Moodle の第２回課題にアップロードしてください

時間の余った人向け課題

複数の円を扱えるようにしてください

マウスをクリックするたびに円を増やす

発展課題

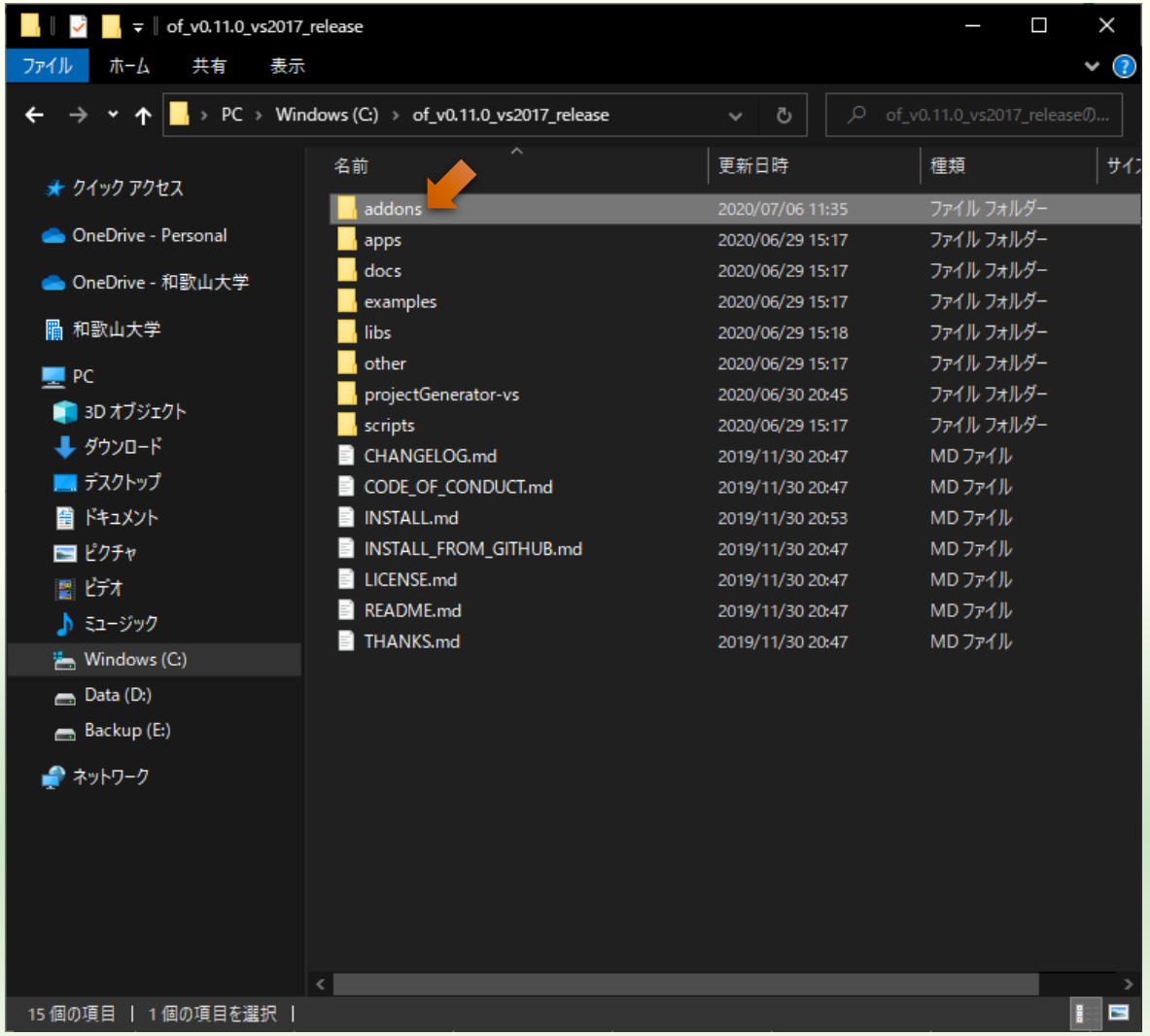
ofxBox2d を使うと楽だけど openFrameworks のバージョン 11 にはまだ対応していないみたいなのでやらなくていいです（対応方法は調査中）

ofxBox2d

- openFrameworks で使える 2 次元の物理エンジン
  - https://github.com/vanderlin/ofxBox2d
  - Box2d (https://box2d.org/) という 2 次元の物理エンジンを openFrameworks で使えるようにしたもの
- これを使うと今日のような課題が楽にできる
  - 多数の物体を扱うことができる
  - 物体同士の衝突も扱うことができる
  - 円以外も扱うことができる
  - 回転も考慮できる

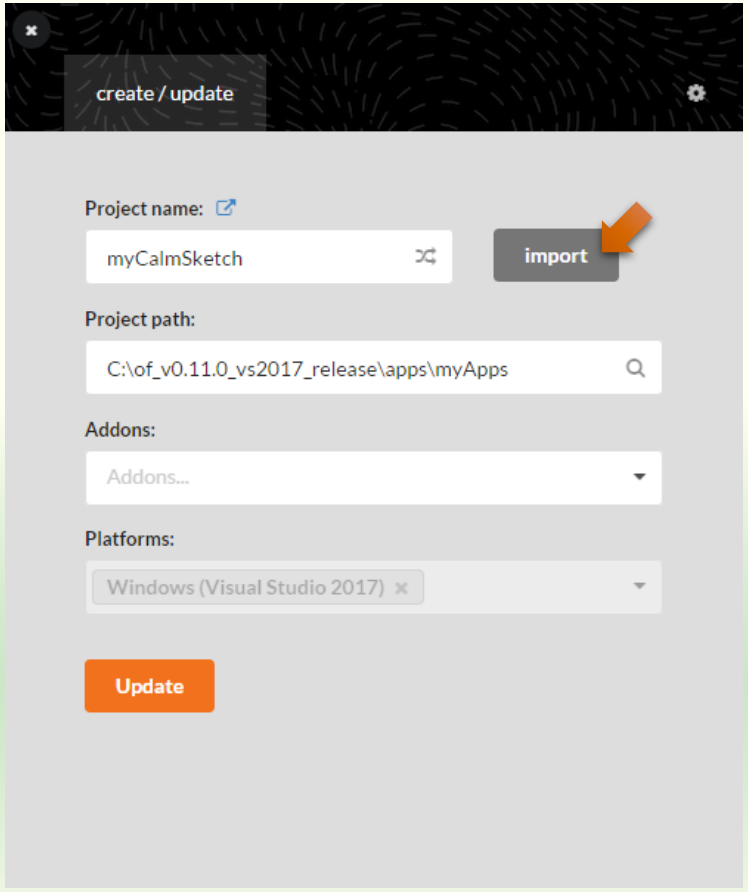
インストール

- https://github.com/vanderlin/ofxBox2d/ をダウンロード
- 展開したフォルダを <openframeworks の展開先>/addons に **ofxBox2d** というフォルダ名で配置

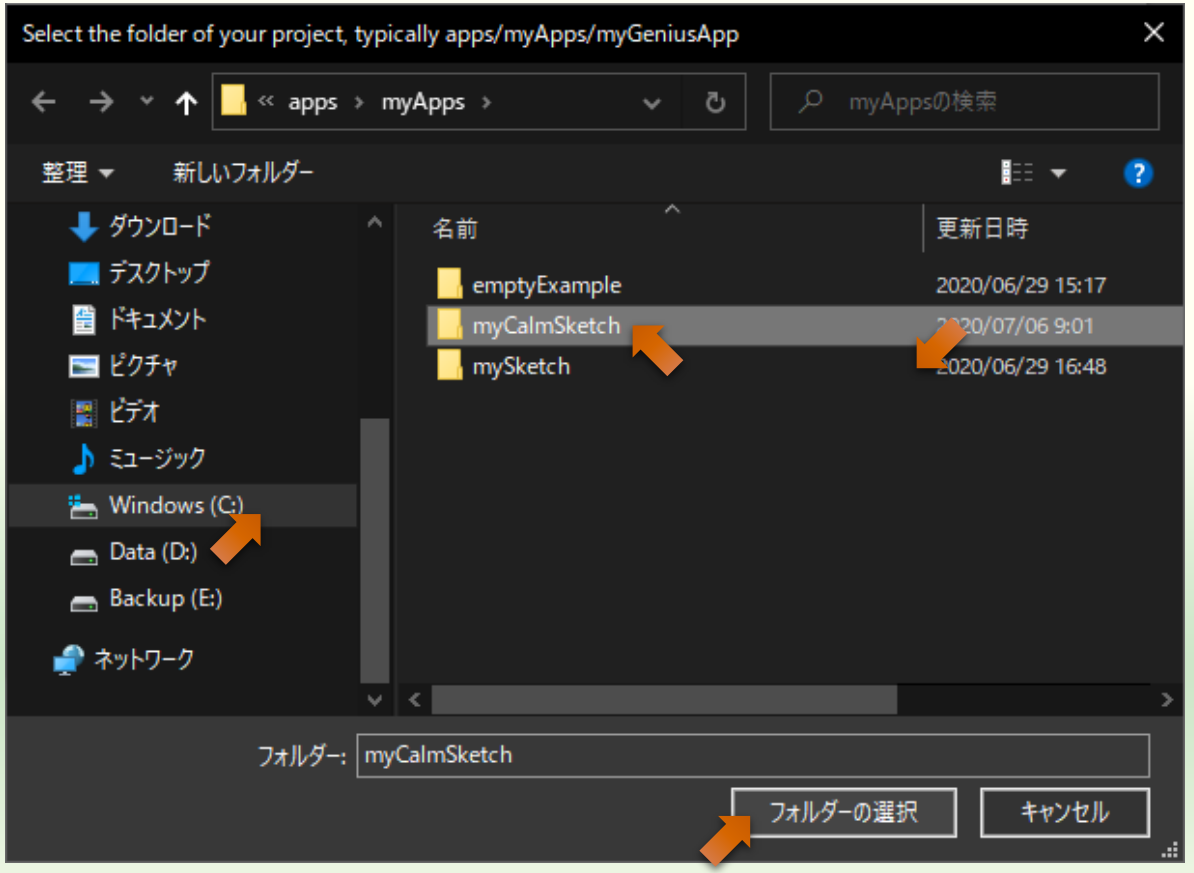


projectBuilder にプロジェクトを読み込む

import をクリック

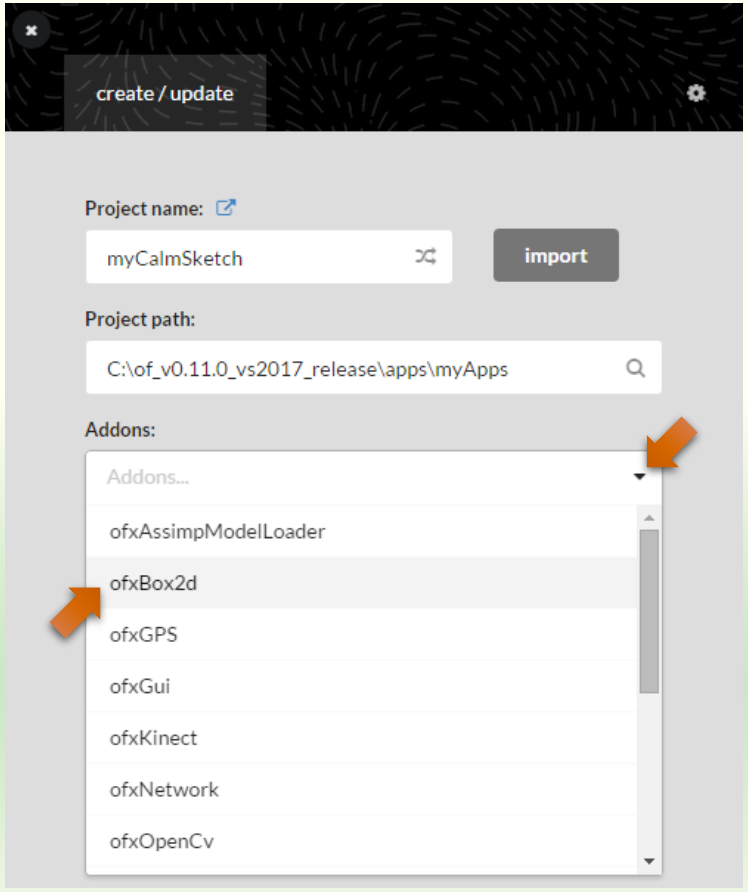


プロジェクトのフォルダを選択

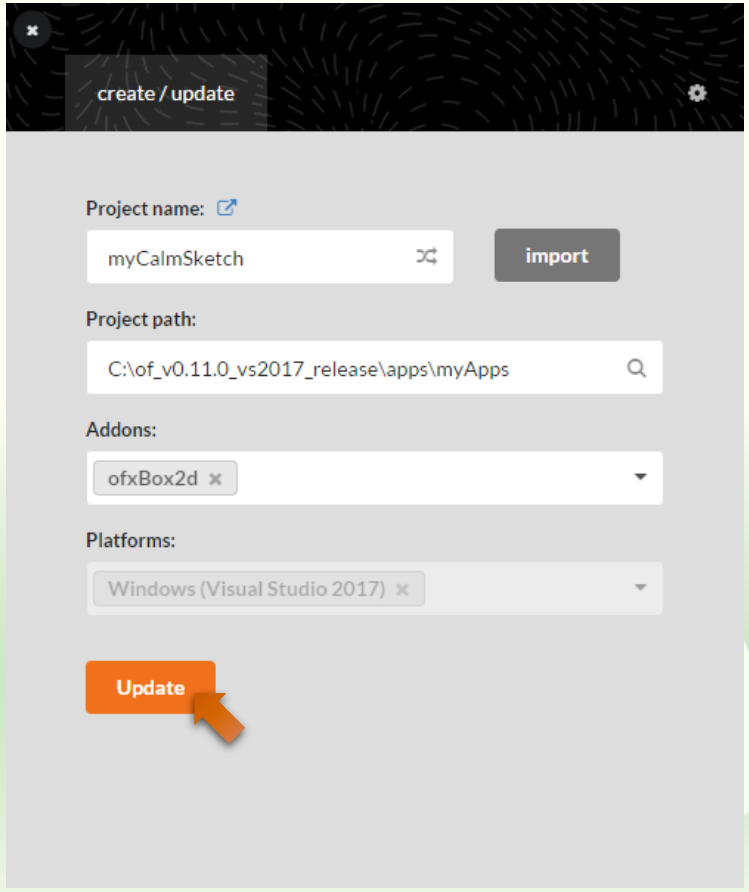


ofxBox2d をプロジェクトに追加する

Addons の中から ofxBox2d を選択



Update してプロジェクトを更新



ofApp.h の変更点

- メンバ変数に以下の内容を追加する

```
#pragma once

#include "ofMain.h"
#include "ofxBox2d.h"

class ofApp : public ofBaseApp{
   ofxBox2d box2d;
  vector<shared_ptr<ofxBox2dCircle>> circles;
```

ofApp.cpp

- setup() で円を生成する

```
// 円を１個生成する
auto circle = std::make_shared<ofxBox2dCircle>();

// 生成した円にパラメータを設定して circles に追加する
circle->setPhysics(3.0, 0.53, 0.1);
circle->setup(box2d.getWorld(), 100, 100, 10);
circles.push_back(circle);
```

ofApp.cpp

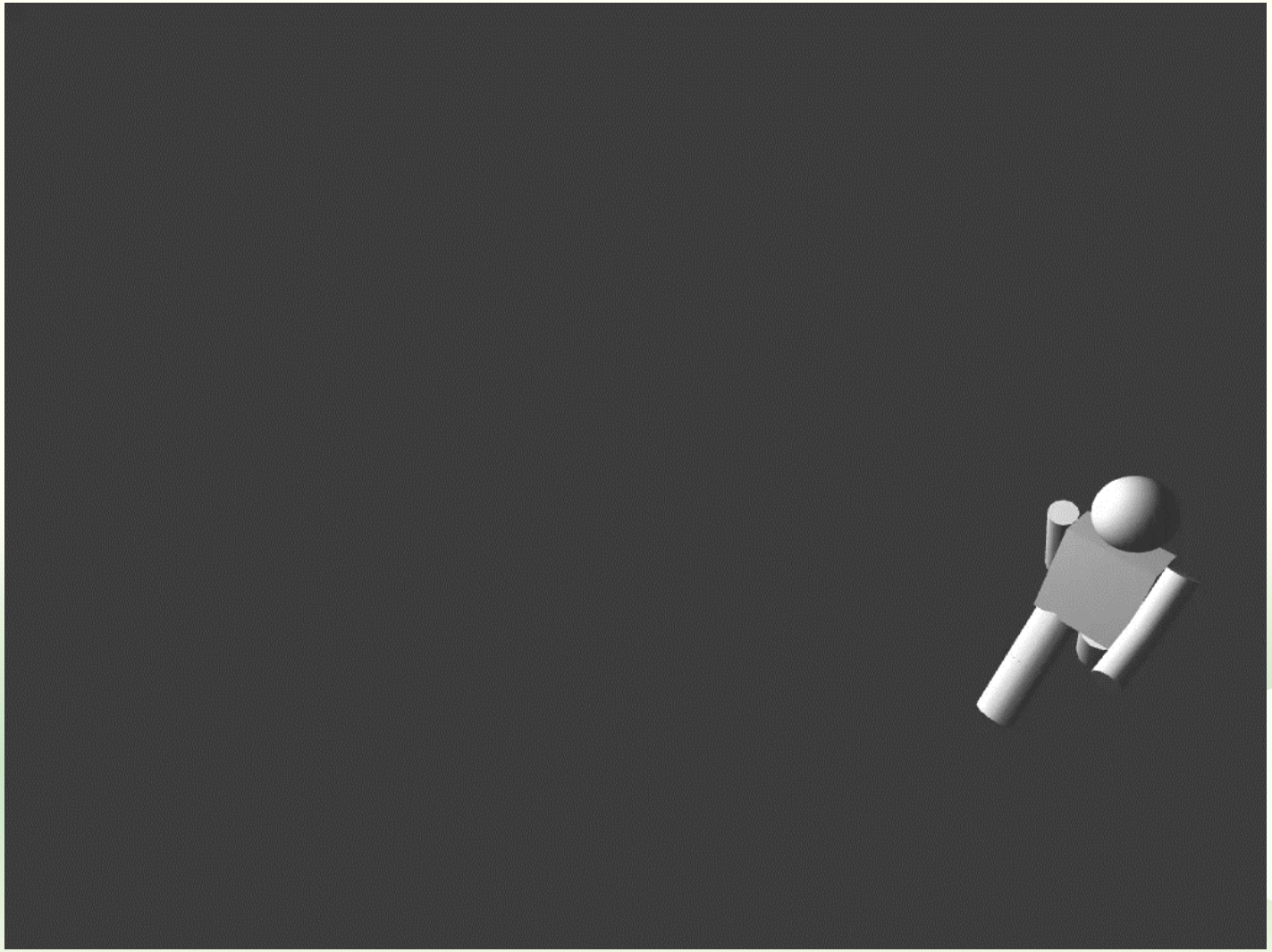
- draw() で円を描画する

```
// circles のすべての circle について
for (auto &circle : circles){

  // 円を１個描画する
  circle->draw();
}
```

メディアプログラミング演習

第3回

本日はロボットの歩行アニメーションの作成

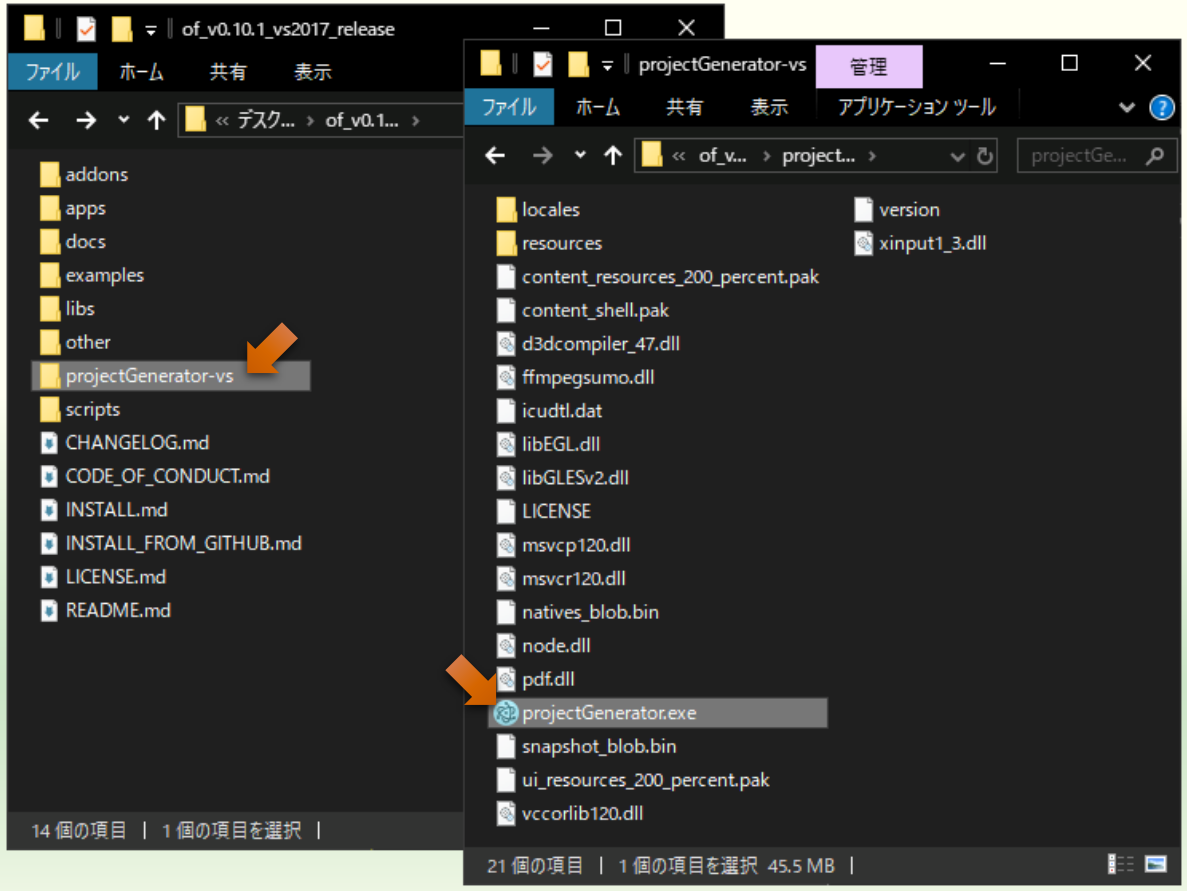


準備

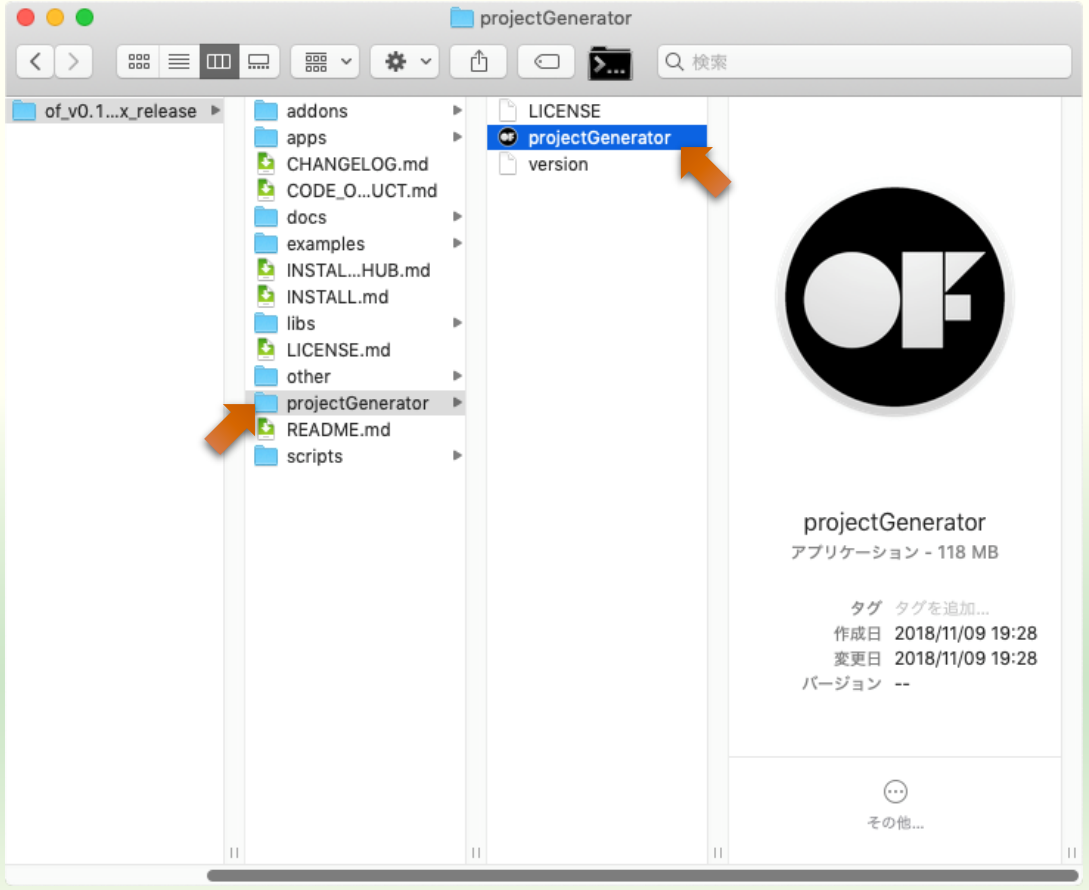
プロジェクトの作成

projectGenerator を起動する

windows 版のパッケージ



macOS 版のパッケージ



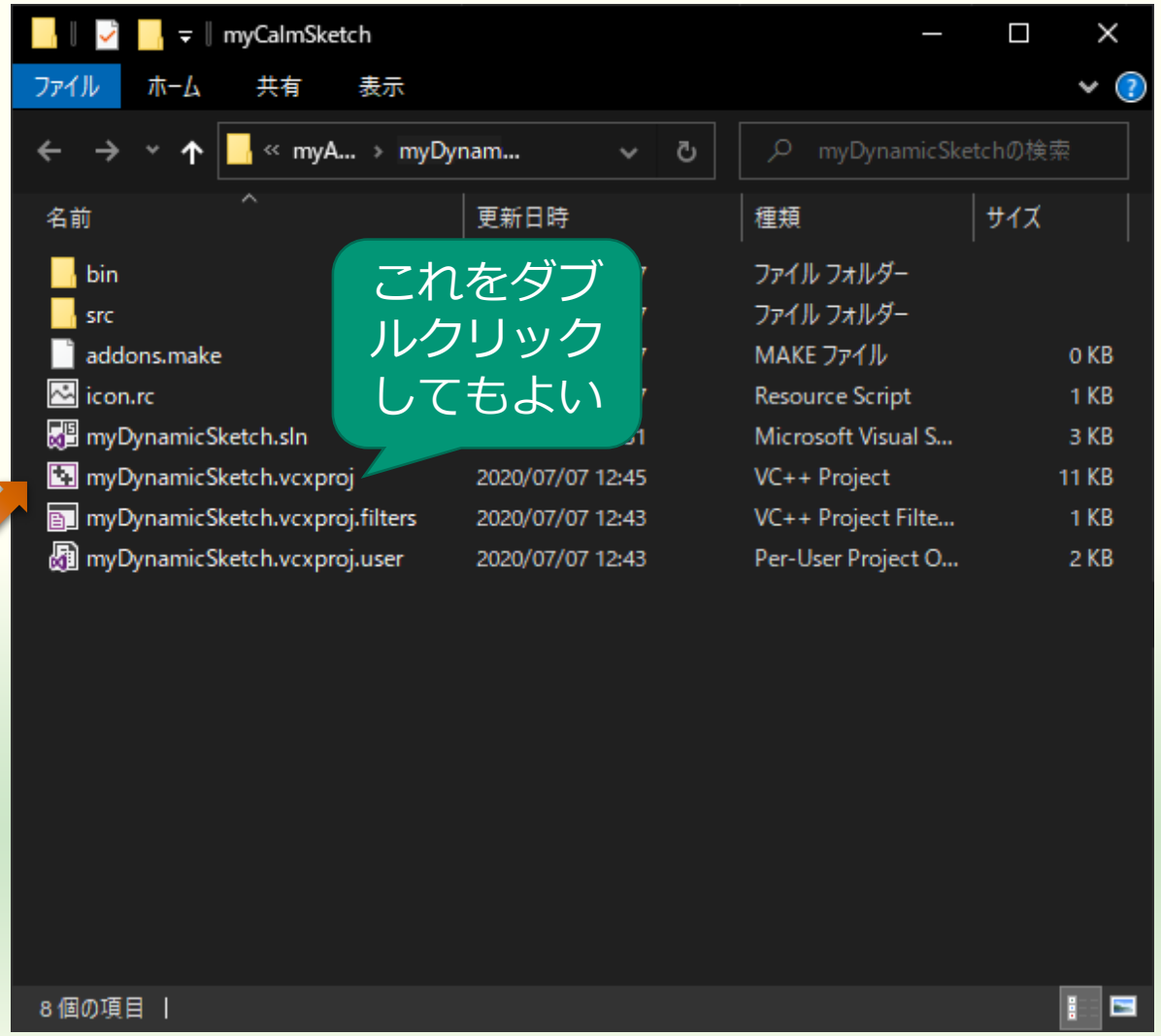
空のプロジェクトの作成

- Project name:
  - 作成するプロジェクト（プログラム）の名前
- Project path:
  - 作成するプロジェクトのファイルを置く場所
  - openFrameworks のパッケージを展開した場所の中の apps¥myApps

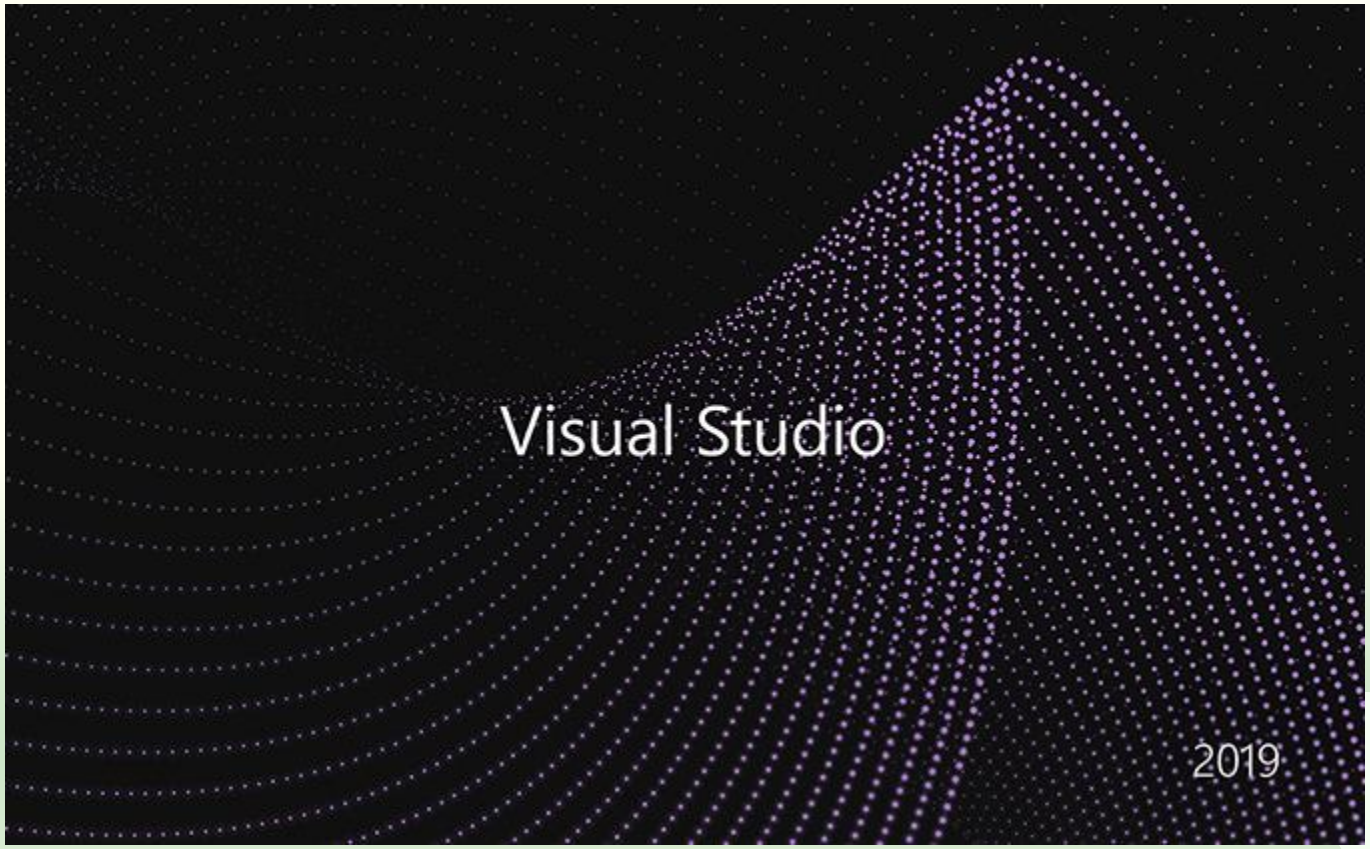
プロジェクトの作成成功

クリックすると開く

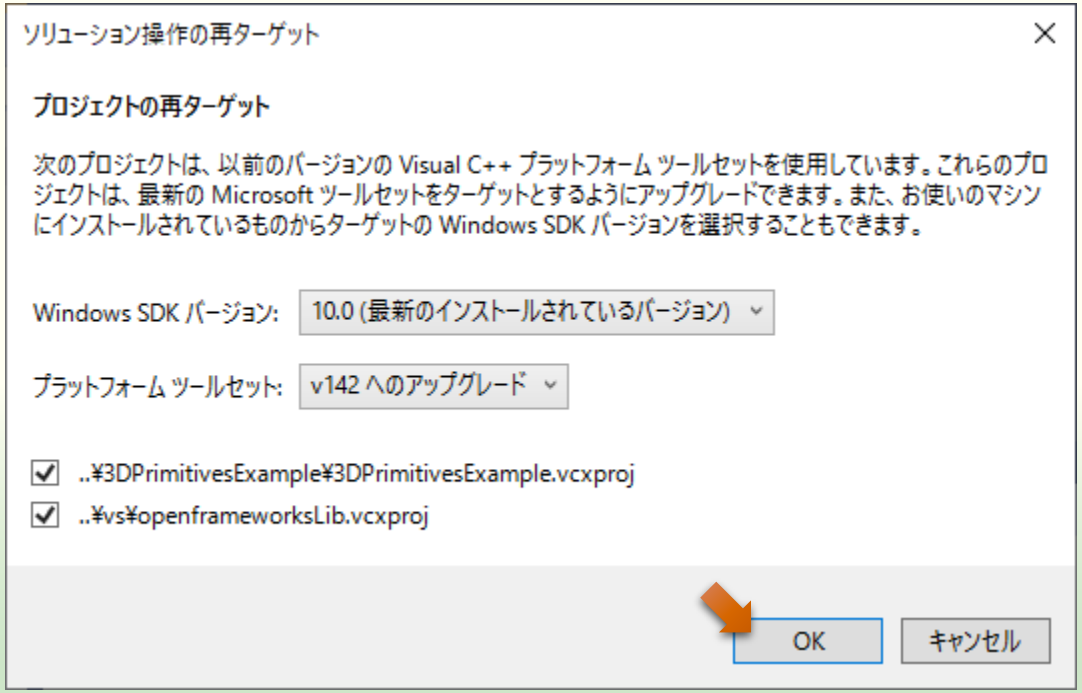
IDE で開く



Visual Studio 2019 が起動する

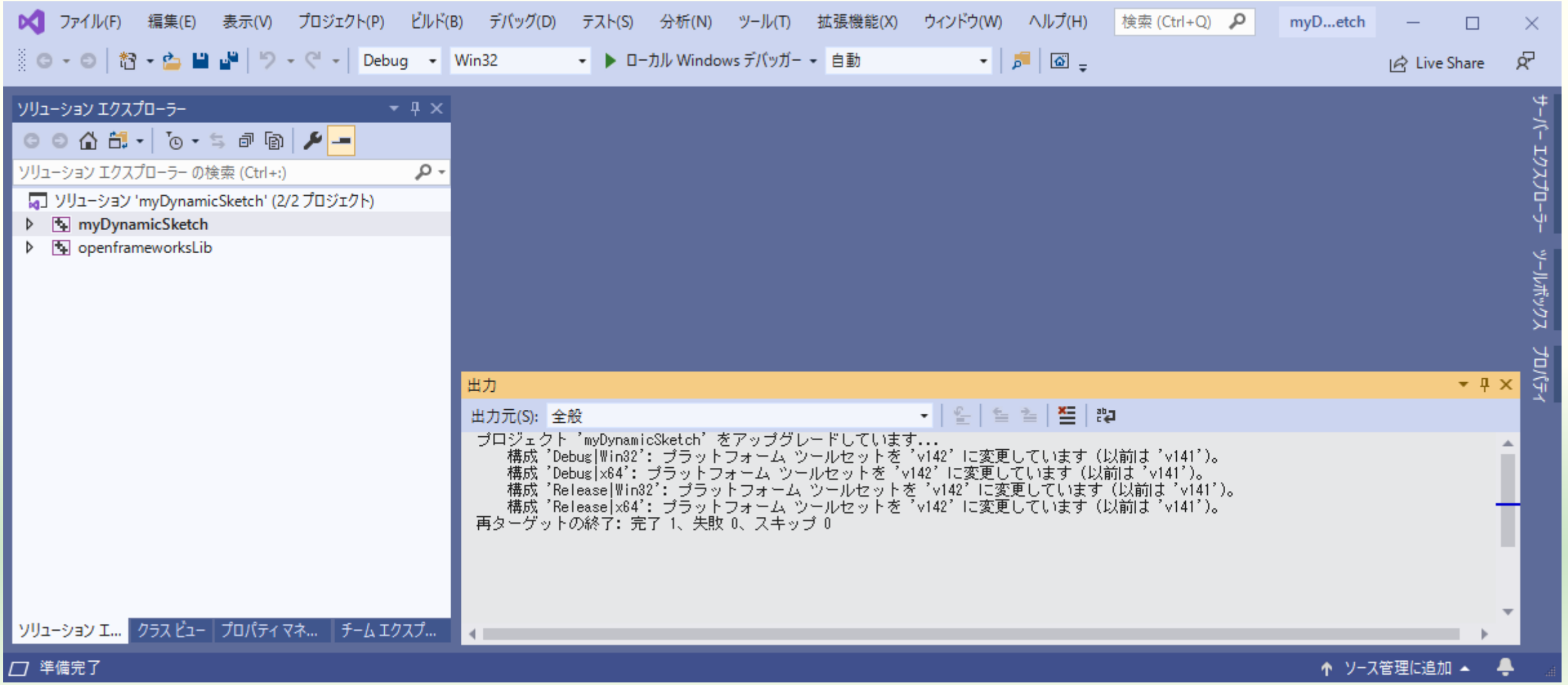


ソリューションの再ターゲット



Visual Studio は頻繁に更新しているので皆さんがお使いの Visual Studio SDK のバージョンと合わない場合がある

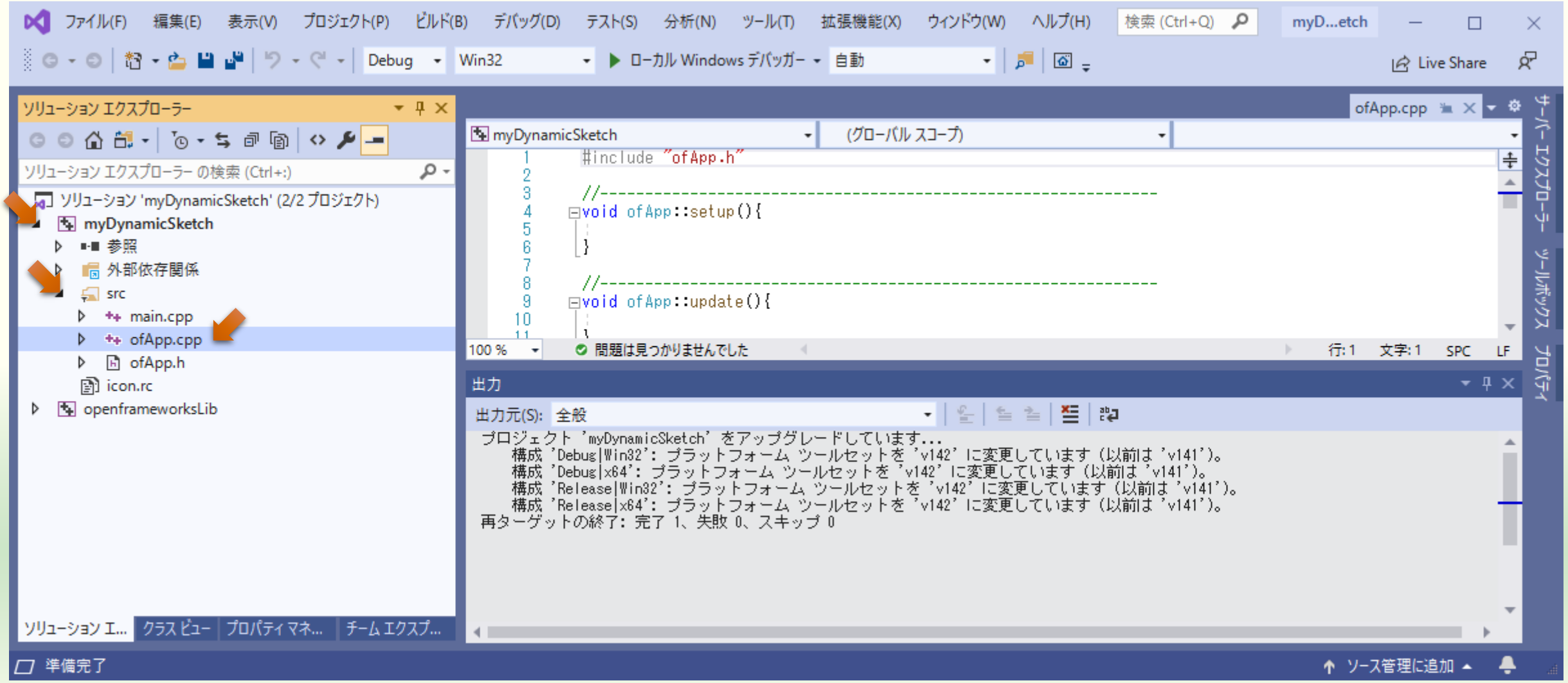
Visual Studio 起動



３次元コンピュータグラフィックス

カメラと照明を配置する

ofApp.cpp を開く



ofDrawSphere() で球を描く

```
//--------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  ofDrawSphere(0.0f, 0.0f, 30.0f);
}

//--------------------------------------------
void ofApp::keyPressed(int key){

}
```

中心位置　半径

- (0, 0) を中心に半径 30 の**球**を ofDrawSphere() 関数で描く

ofDrawSphere() の宣言

- **void ofDrawSphere(float x, float y, float radius)**

ofDrawSphere() 関数には戻り値がない

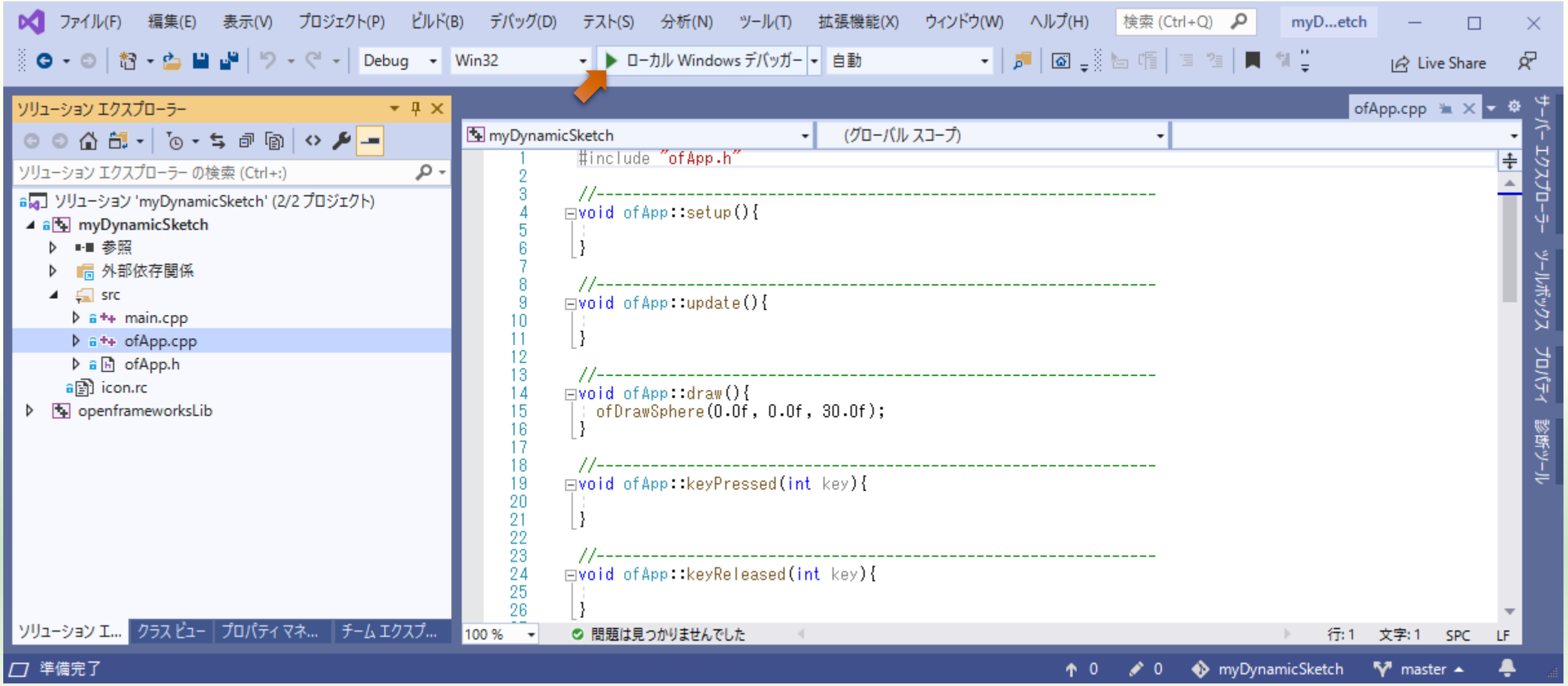
第 1 引数 x は float 型である

第 2 引数 y は float 型である

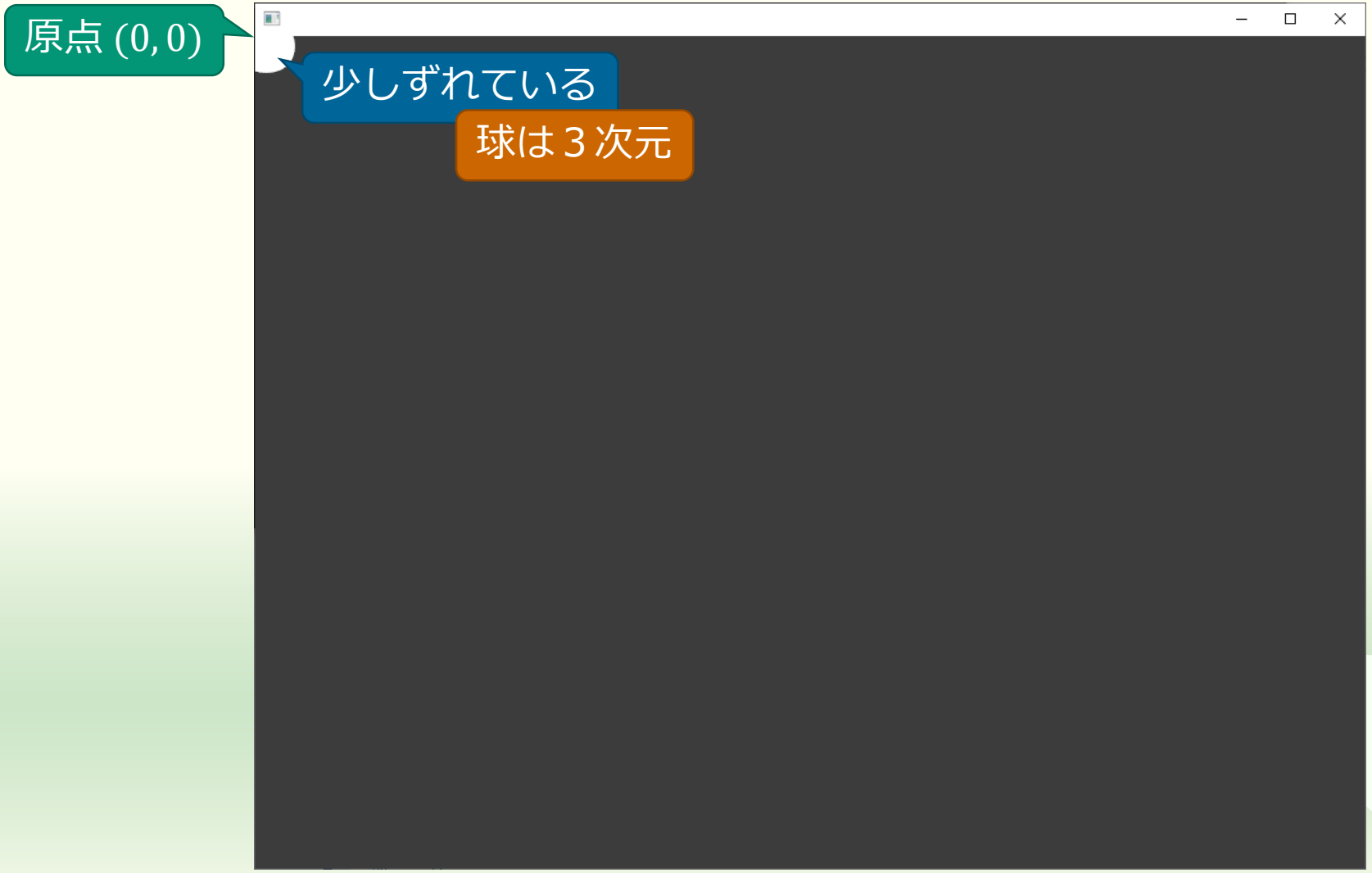
第 3 引数 radius は float 型である

- 仮引数 x, y, radius が何を表すのかは示されていない
  - マニュアルやコメントで説明されている（はず）
  - 宣言では引数名 x, y, radius が省略されている場合がある

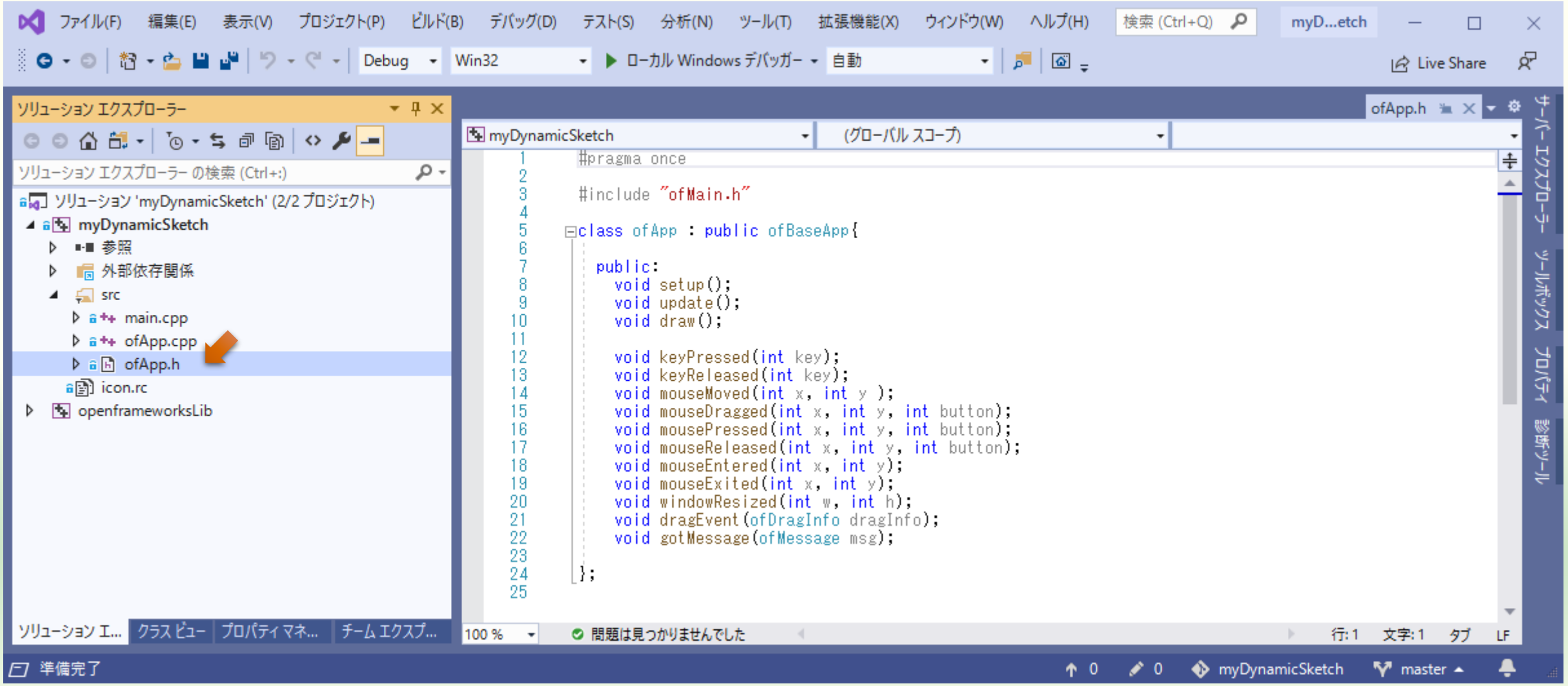
ビルドと実行


ファイル(F)
編集(E)
表示(V)
プロジェクト(P)
ビルド(B)
デバッグ(D)
テスト(S)
分析(N)
ツール(T)
拡張機能(X)
ウィンドウ(W)
ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myDynamicSketch' (2/2 プロジェクト)
myDynamicSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.cpp
myDynamicSketch
(グローバル スコープ)
#include "ofApp.h"
//--
void ofApp::setup(){
}
//--
void ofApp::update(){
}
//--
void ofApp::draw(){
ofDrawSphere(0.0f, 0.0f, 30.0f);
}
//--
void ofApp::keyPressed(int key){
}
//--
void ofApp::keyReleased(int key){
}
100 %
問題は見つかりませんでした
行: 1
文字: 1
SPC
LF
準備完了
0
0
myDynamicSketch
master

左上の原点を中心に球が描かれる



そこで ofApp.h を開く



ofApp クラスにカメラのメンバ変数を追加する

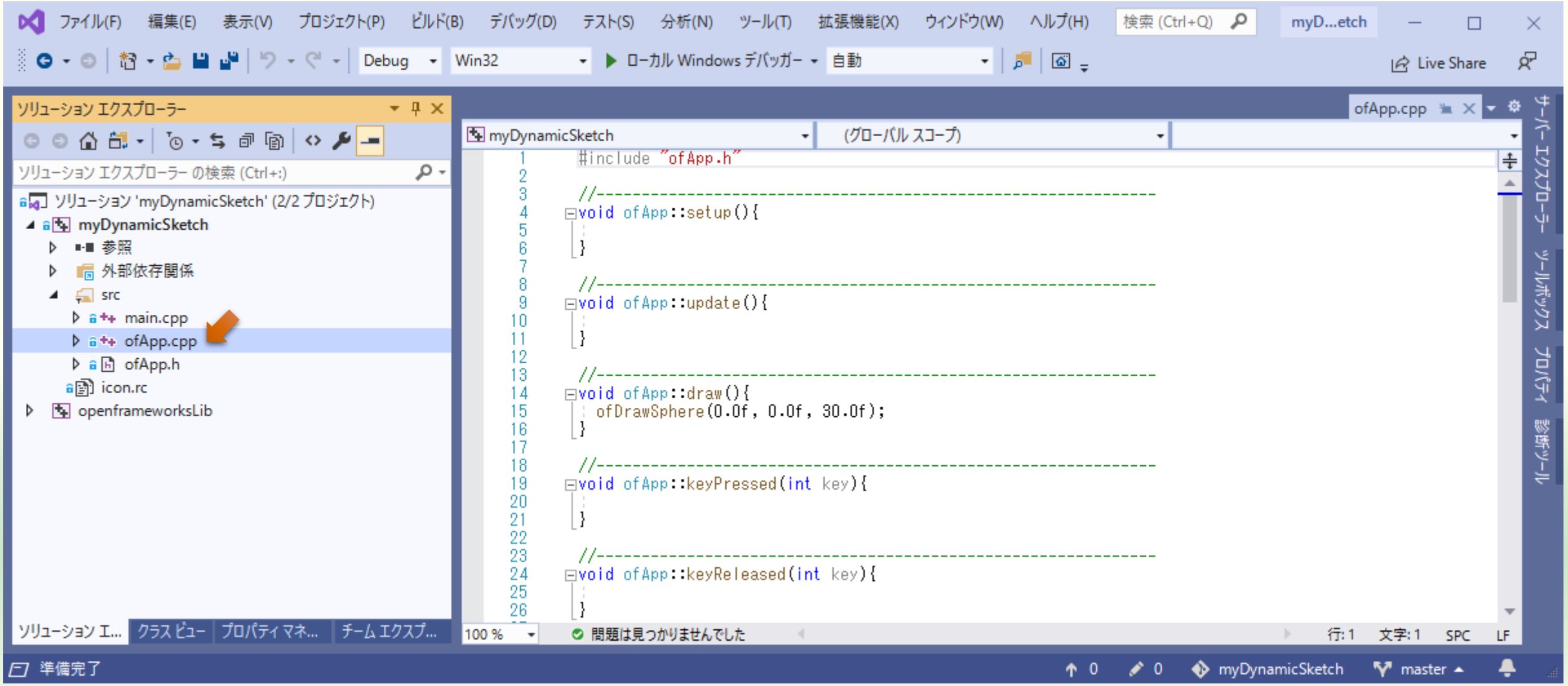
```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{
  ofCamera camera;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  （以下略）
```

- **ofCamera** は 3 次元空間中のカメラのクラス
  - 変数 camera は ofCamera クラスの**インスタンス**

オブジェクトという用語について

- ３次元ＣＧではシーンに配置する**物体**
  - ３次元空間ではカメラもシーンに配置する物体の一つ
  - **シーン**はＣＧの作成対象となる３次元空間
- C++ 言語ではクラスによって定義された構造を持つメモリ上の**データ**
  - camera は位置や方向の情報を保持するメモリが割り当てられた**オブジェクト**
  - オブジェクトはクラスから生成したという文脈では**インスタンス**と呼ばれる

再び ofApp.cpp を開く

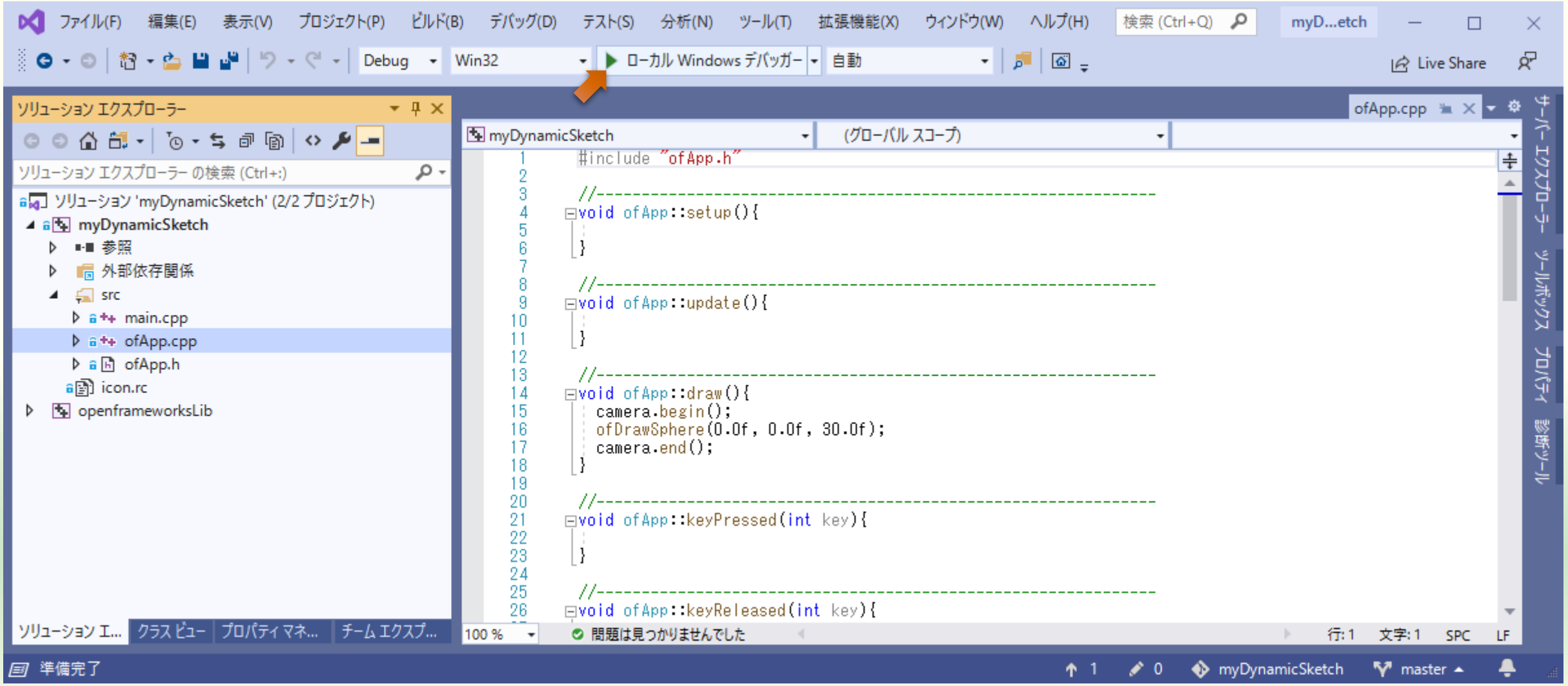


カメラを有効にする

```
//--------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  camera.begin();
  ofDrawSphere(0.0f, 0.0f, 30.0f);
  camera.end();
}

//--------------------------------------------
void ofApp::keyPressed(int key){

}
```

- camera.begin() と camera.end() の間は３次元空間への描画になる
  - begin() , end() は ofCamera クラスの**メンバ関数**
    - camera というオブジェクトを操作する「方法」なので**メソッド**ともいう
    - “.”（ピリオド）はメンバ関数を呼び出す**ドット演算子**

ビルドと実行


ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) テスト(S) 分析(N) ツール(T) 拡張機能(X) ウィンドウ(W) ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myDynamicSketch' (2/2 プロジェクト)
myDynamicSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.cpp
myDynamicSketch
(グローバル スコープ)
#include "ofApp.h"
//--
void ofApp::setup(){
}
//--
void ofApp::update(){
}
//--
void ofApp::draw(){
camera.begin();
ofDrawSphere(0.0f, 0.0f, 30.0f);
camera.end();
}
//--
void ofApp::keyPressed(int key){
}
//--
void ofApp::keyReleased(int key){
100 %
問題は見つかりませんでした
行: 1 文字: 1 SPC LF
準備完了
1 0 myDynamicSketch master

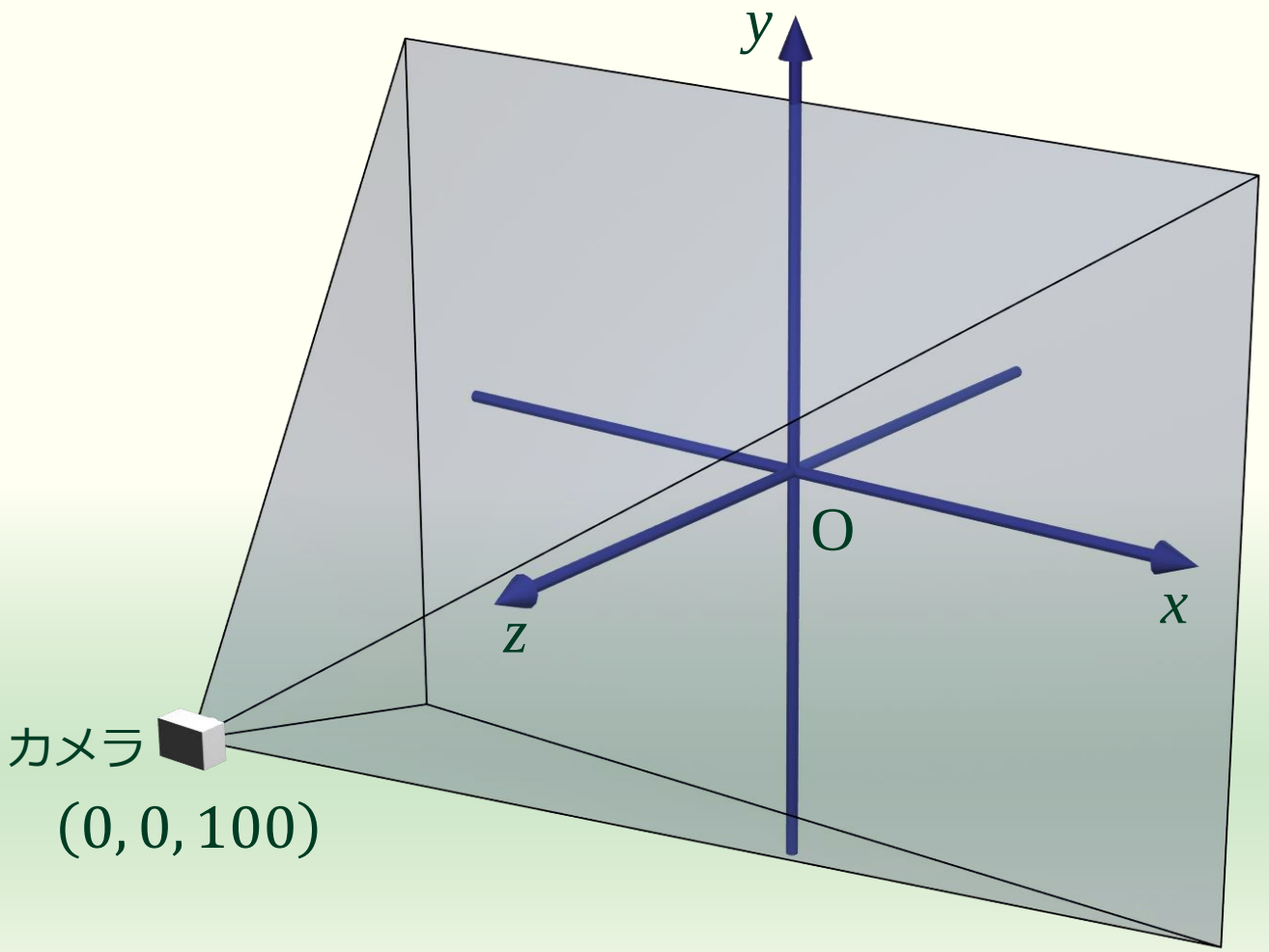
真っ白

カメラを後ろに下げる

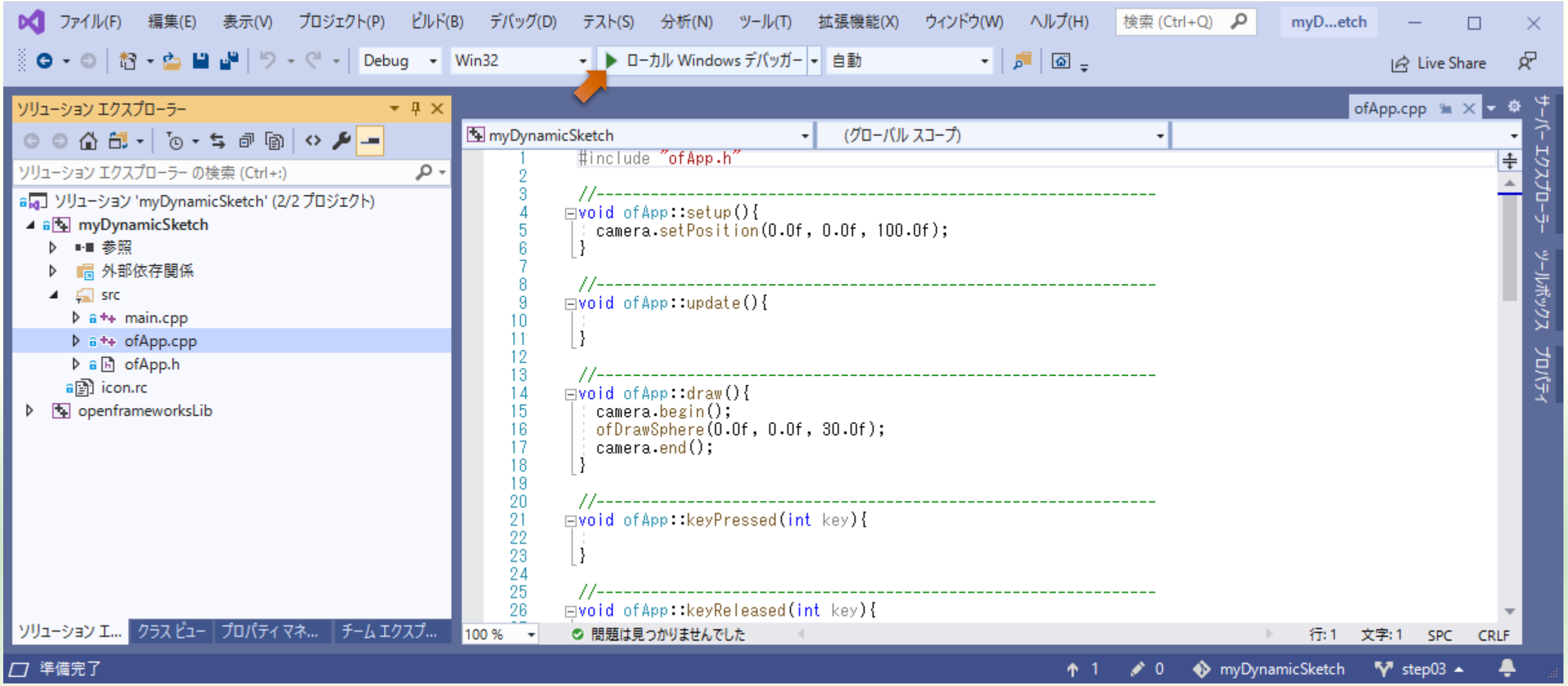
```
//--------------------------------------------
void ofApp::setup(){
  camera.setPosition(0.0f, 0.0f, 100.0f);
}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  camera.begin();
  ofDrawSphere(0.0f, 0.0f, 30.0f);
  camera.end();
}
```

- ofCamera クラスのオブジェクトは setPosition() という**メソッド**でカメラの位置を設定できる
- カメラを動かさないなら setup() で設定する
  - 動かしたければ update() か draw() で設定する

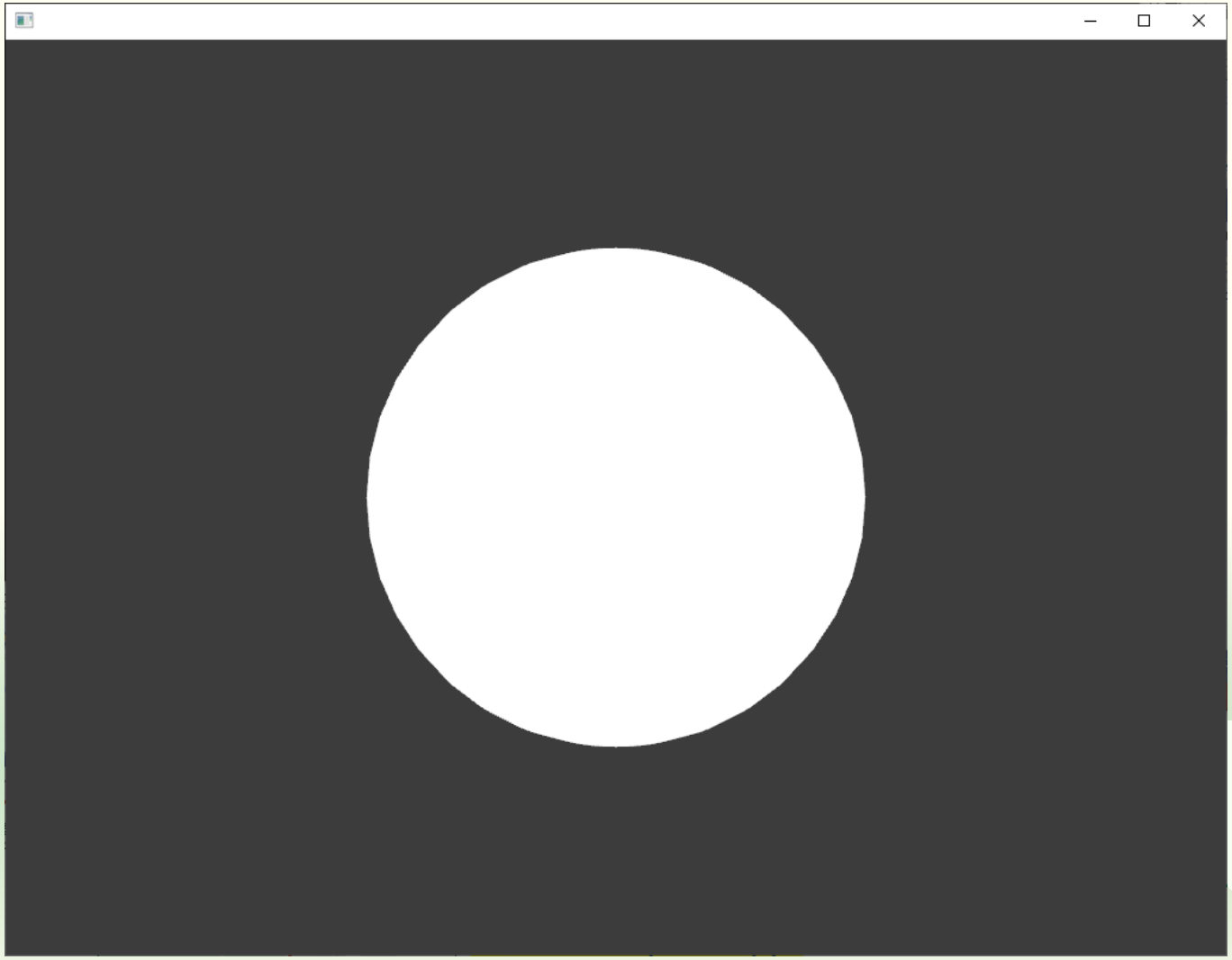
カメラの位置



ビルドと実行


ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) テスト(S) 分析(N) ツール(T) 拡張機能(X) ウィンドウ(W) ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myDynamicSketch' (2/2 プロジェクト)
myDynamicSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.cpp
myDynamicSketch
(グローバル スコープ)
#include "ofApp.h"
//--
void ofApp::setup(){
camera.setPosition(0.0f, 0.0f, 100.0f);
}
//--
void ofApp::update(){
}
//--
void ofApp::draw(){
camera.begin();
ofDrawSphere(0.0f, 0.0f, 30.0f);
camera.end();
}
//--
void ofApp::keyPressed(int key){
}
//--
void ofApp::keyReleased(int key){
100 %
問題は見つかりませんでした
行: 1 文字: 1 SPC CRLF
準備完了
1 0 myDynamicSketch step03

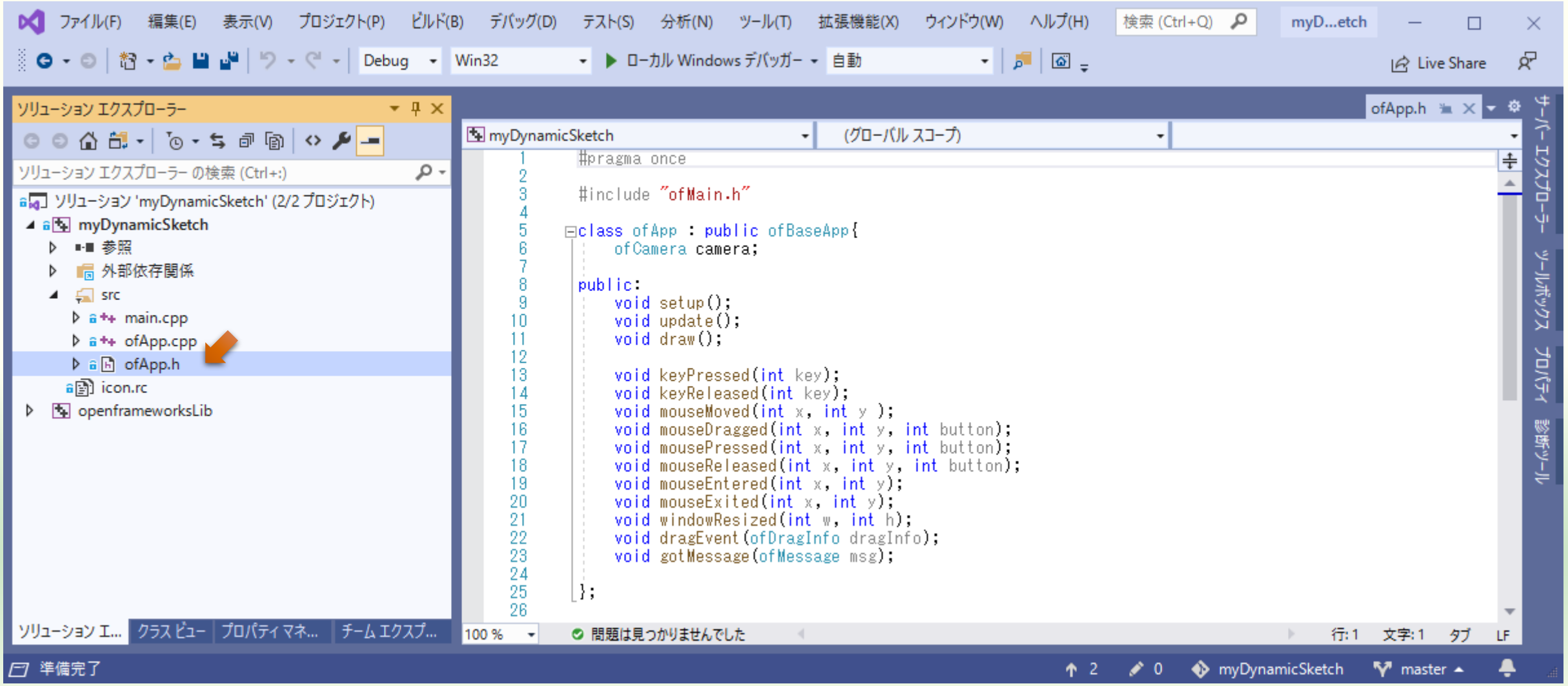
白い円が描かれる



setPosition() メソッド

- void ofNode::setPosition(float px, float py, float pz)
  - px, py, pz: カメラの３次元空間中の位置
- void ofNode::setPosition(const glm::vec3 &p)
  - p: カメラの３次元空間中の位置
  - glm::vec3 は x, y, z の３要素のベクトル
- **ofNode** は３次元のシーンを構成するためのクラス
  - ofCamera のほか３次元のシーンに配置する「部品」の多くはこの ofNode クラスを**継承**（性質を受け継ぐこと）して定義されている
  - 継承元のクラスは**基底クラス**、継承先のクラスは**派生クラス**

また ofApp.h を開く

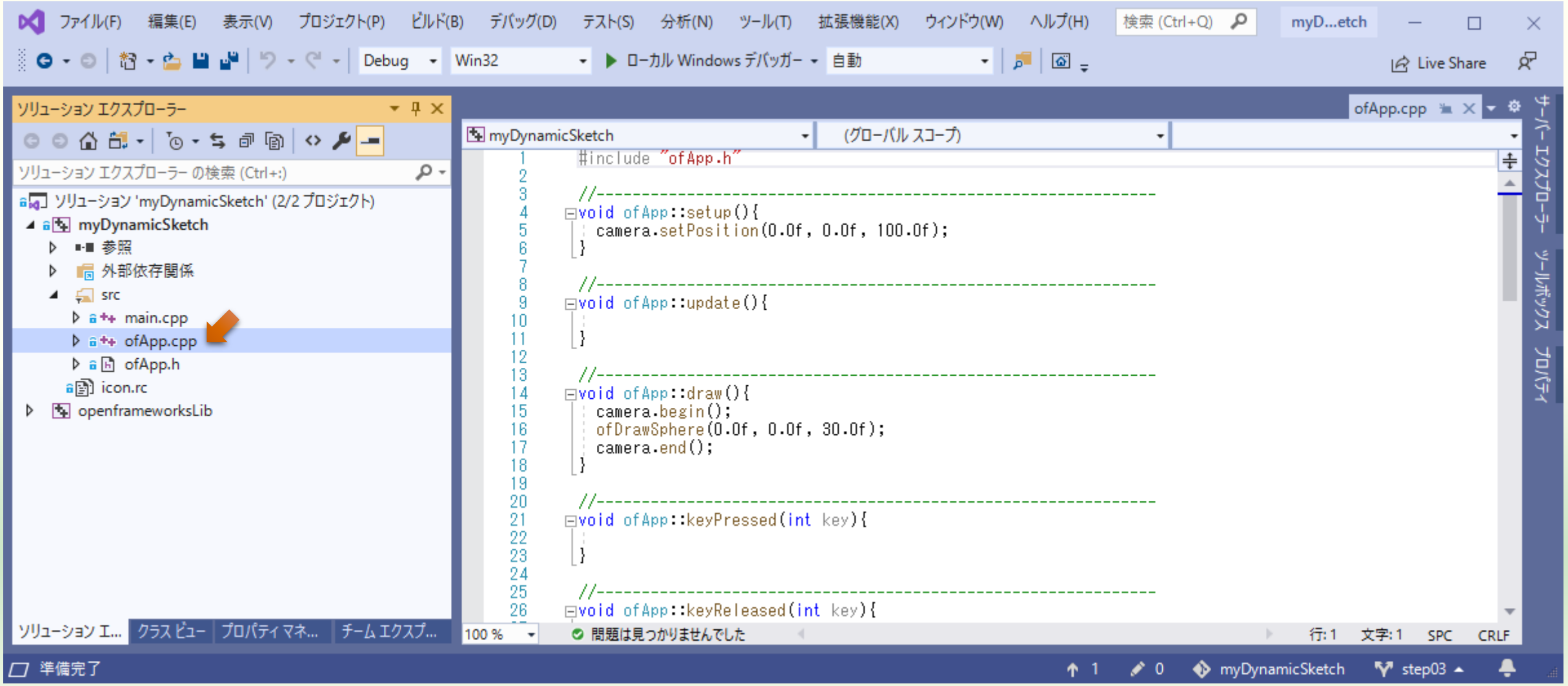


ofApp クラスにライトのメンバ変数を追加する

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{
  ofCamera camera;
  ofLight light;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  （以下略）
```

- **ofLight** は 3 次元空間中の光源のクラス
  - 変数 light は ofLight クラスのインスタンス

そして ofApp.cpp を開く

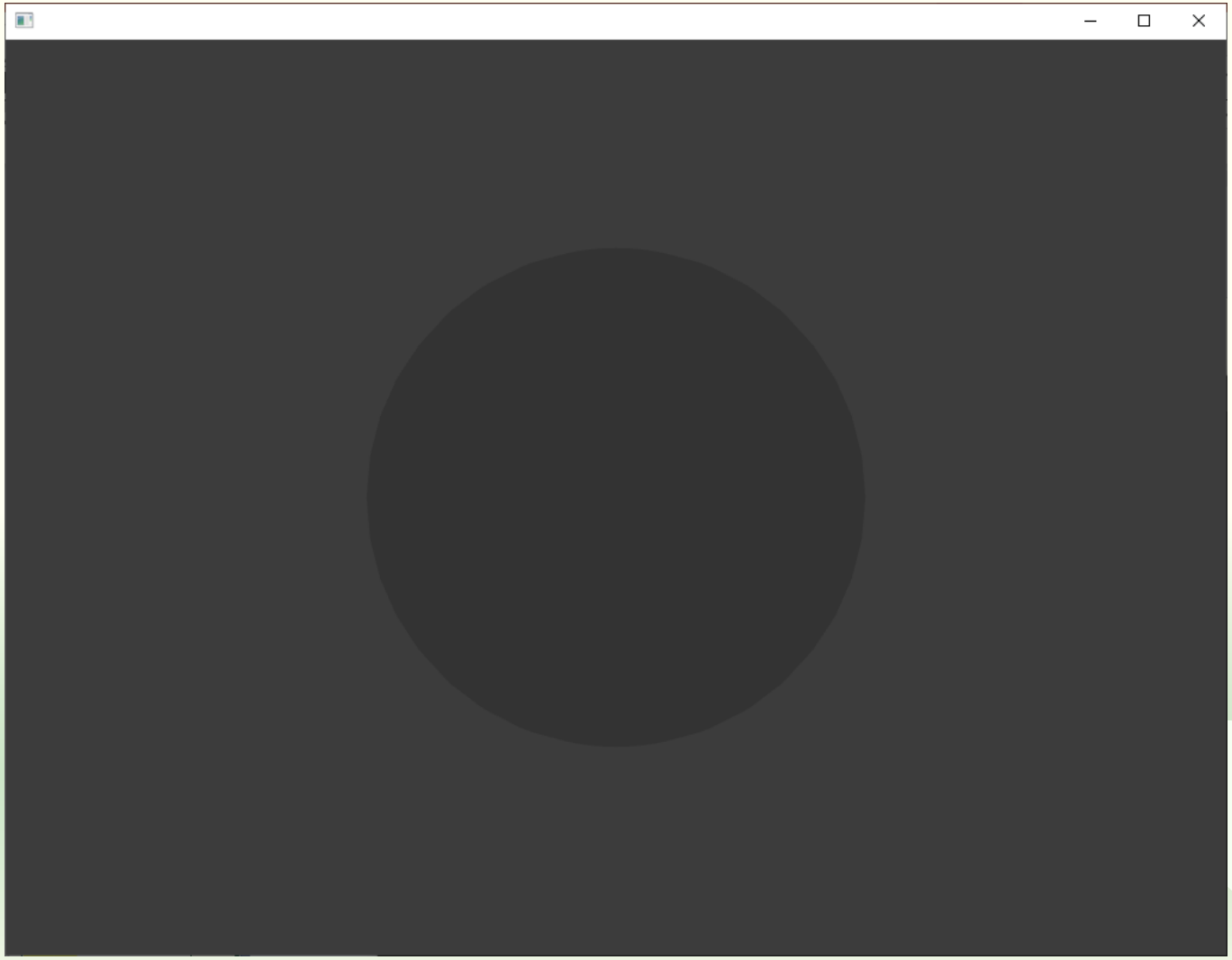


ライトをつける

```
//--------------------------------------------
void ofApp::setup(){
  camera.setPosition(0.0f, 0.0f, 100.0f);
  light.enable();
}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  camera.begin();
  ofDrawSphere(0.0f, 0.0f, 30.0f);
  camera.end();
}
```

- light の enable() はライトを点灯するメソッド
- 消すときは disable()
  - つけっぱなしにするなら setup() で enable() する
  - 付けたり消したりしたいときは update() か draw() で必要に応じて enable() か .disable() する

逆に暗い

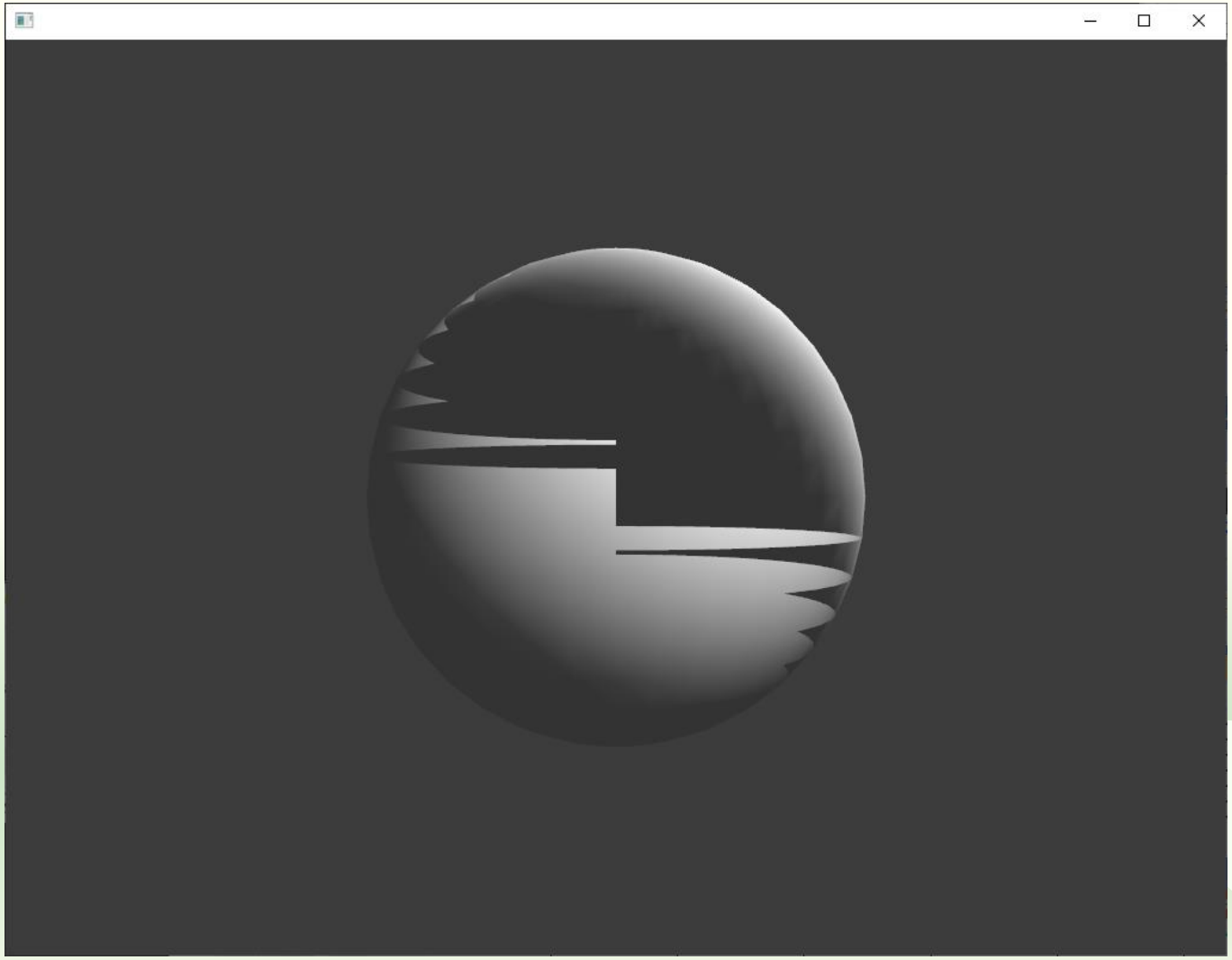


ライトを右上の方に持って行く

```
//--------------------------------------------
void ofApp::setup(){
  camera.setPosition(0.0f, 0.0f, 100.0f);
  light.setPosition(60.0f, 80.0f, 100.0f);
  light.enable();
}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  camera.begin();
  ofDrawSphere(0.0f, 0.0f, 30.0f);
  camera.end();
}
```

- ofLight クラスも ofNode クラスを継承している
  - setPosition() メソッドで位置を指定できる
- light は最初は原点にあった
  - つまり球の内部にあった
  - そのため球の表には光が当たってなかった

明るくなったけど変

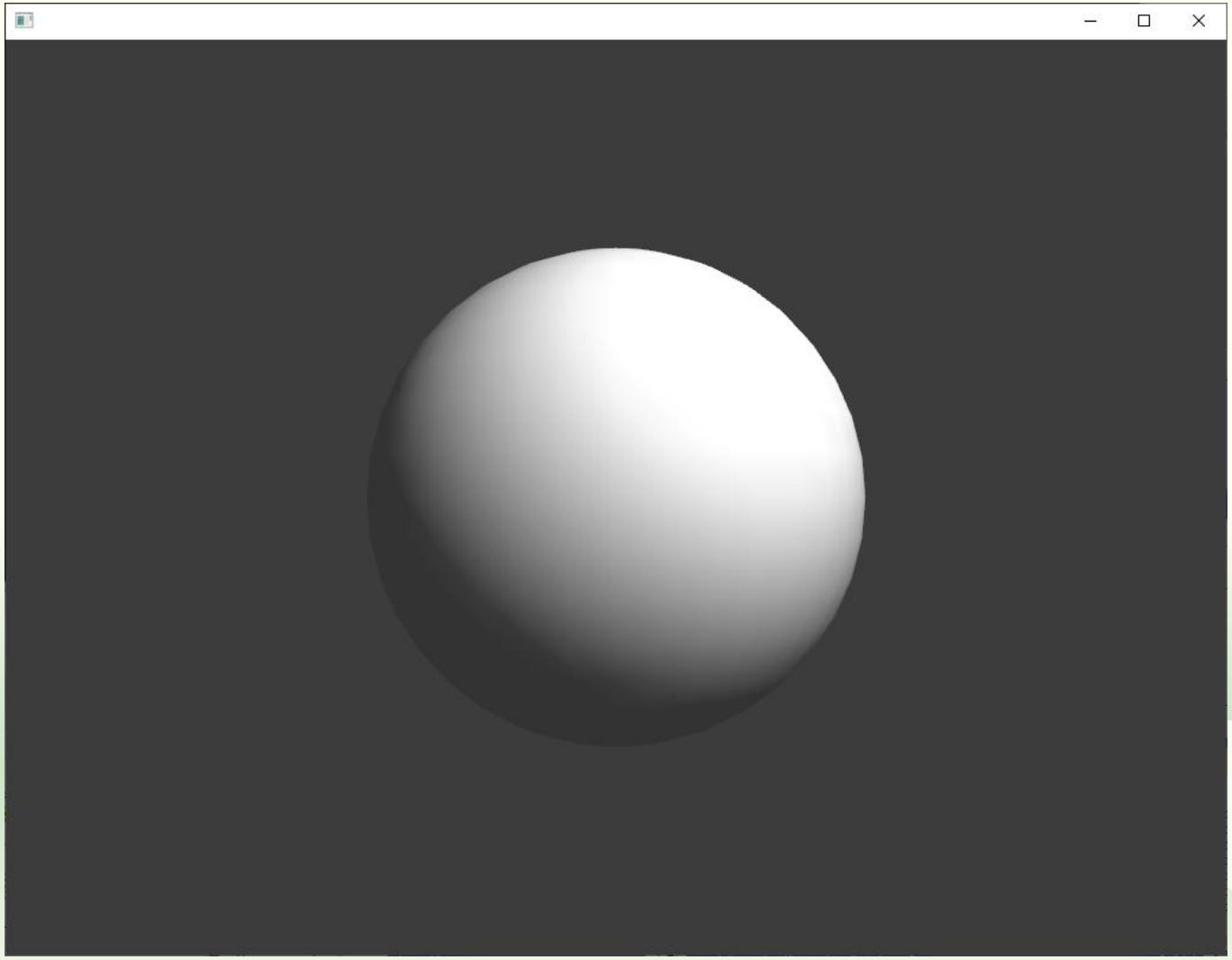


見えないはずの面を描かないようにする

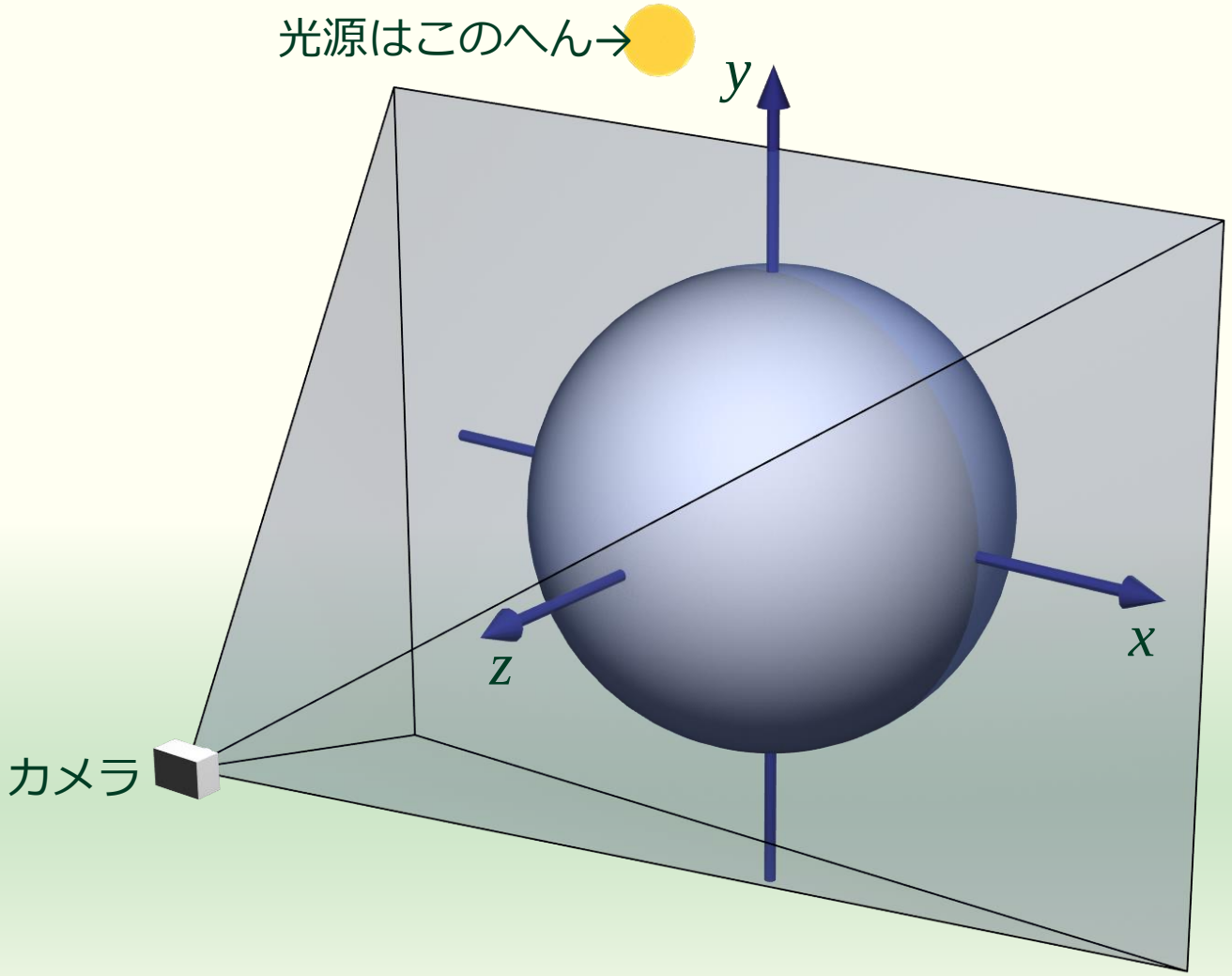
```
//--------------------------------------------
void ofApp::setup(){
  ofEnableDepthTest();
  camera.setPosition(0.0f, 0.0f, 100.0f);
  light.setPosition(60.0f, 80.0f, 100.0f);
  light.enable();
}

//--------------------------------------------
void ofApp::update(){

}

//--------------------------------------------
void ofApp::draw(){
  camera.begin();
  ofDrawSphere(0.0f, 0.0f, 30.0f);
  camera.end();
}
```

- ３次元ＣＧで順序を考えずに面を描くと手前の面の上に奥の面が後から重ね描きされてしまう
- 面の**奥行き**を画素単位に比較して手前の面だけが描かれるようにする
- **隠面消去処理**という

ついにきれいに描けた



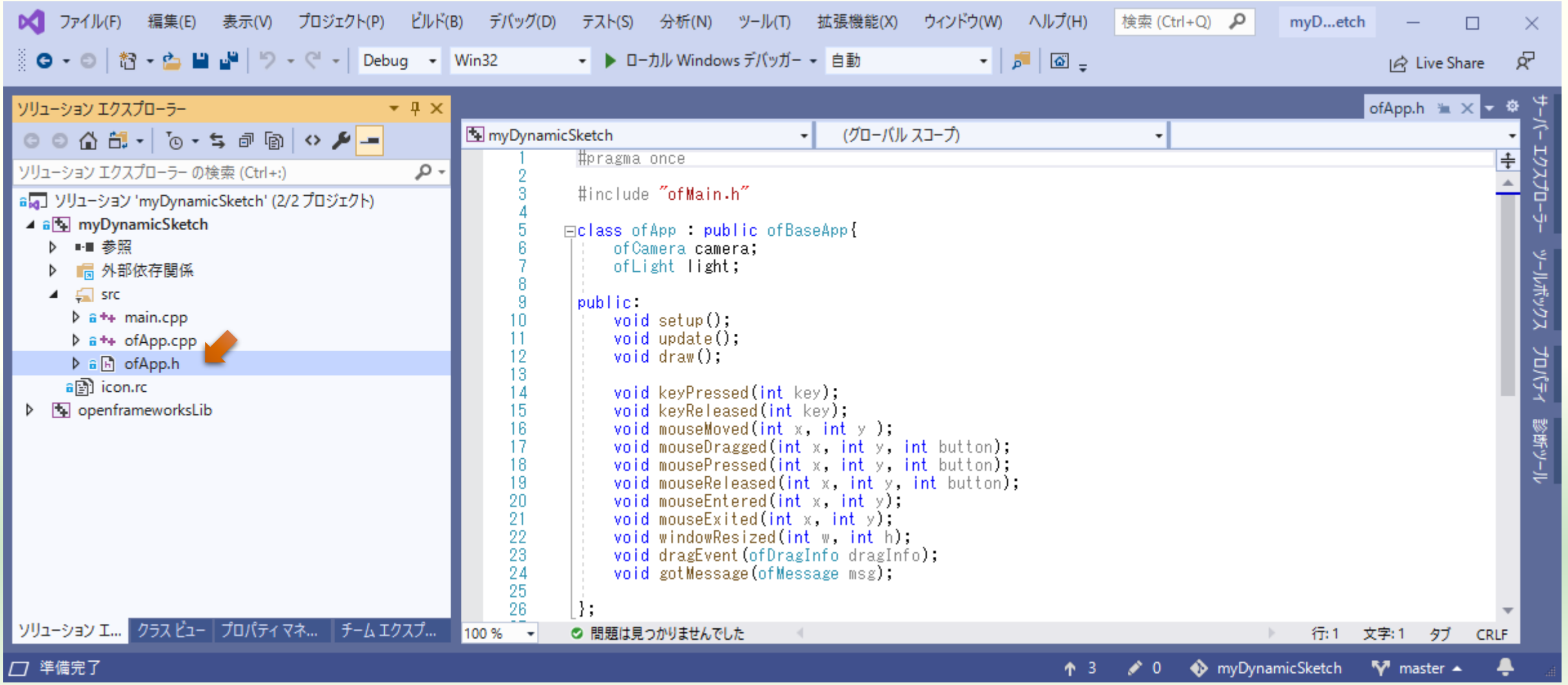
カメラ、光源、球の位置関係



オブジェクト

オブジェクトを使ってオブジェクト（物体）を描く

またまた ofApp.h を開く

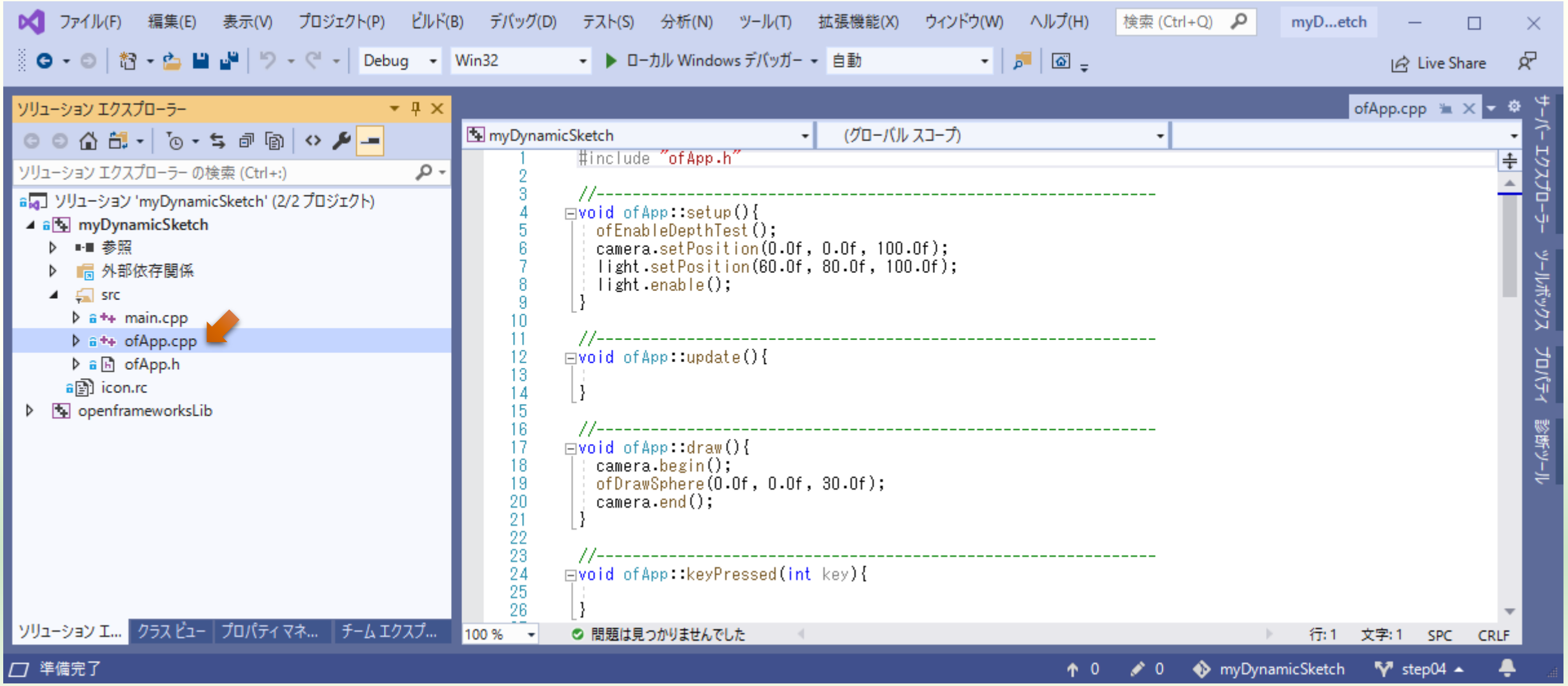


ofApp クラスに球のメンバ変数を追加する

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{
  ofCamera camera;
  ofLight light;
  ofSpherePrimitive sphere;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  （以下略）
```

- **ofSpherePrimitive** は球の（物体という意味での）オブジェクトの**クラス**
- sphere は
  - 「球という**物体**」という意味でのオブジェクトで
  - 「メモリを与えられた**実体**」という意味でのオブジェクトでもある

そしてもう一度 ofApp.cpp を開く



球の半径と解像度を設定して描画する

```
//--------------------------------------------
void ofApp::setup(){
  ofEnableDepthTest();
  camera.setPosition(0.0f, 0.0f, 100.0f);
  light.setPosition(60.0f, 80.0f, 100.0f);
  light.enable();

  sphere.set(30.0f, 20);
}

 (途中略)

//--------------------------------------------
void ofApp::draw(){
  camera.begin();
  sphere.draw();
  camera.end();
}
```

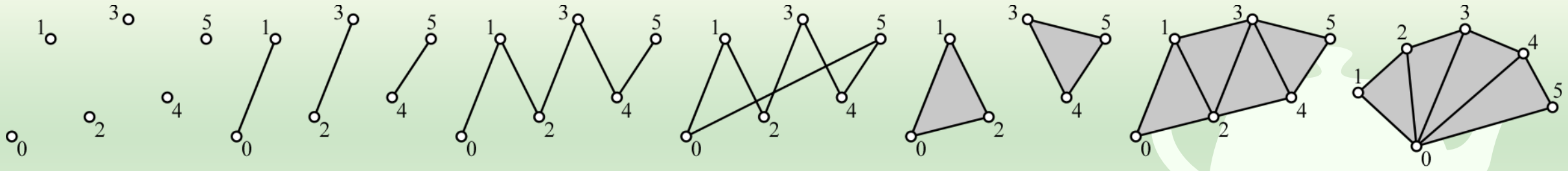
半径

解像度

- set() メソッドを使って半径と解像度を指定する
  - setRadius() メソッドで半径だけを指定できる
  - setResolution() メソッドで解像度だけを指定できる
- draw() メソッドにより図形の描画が行われる

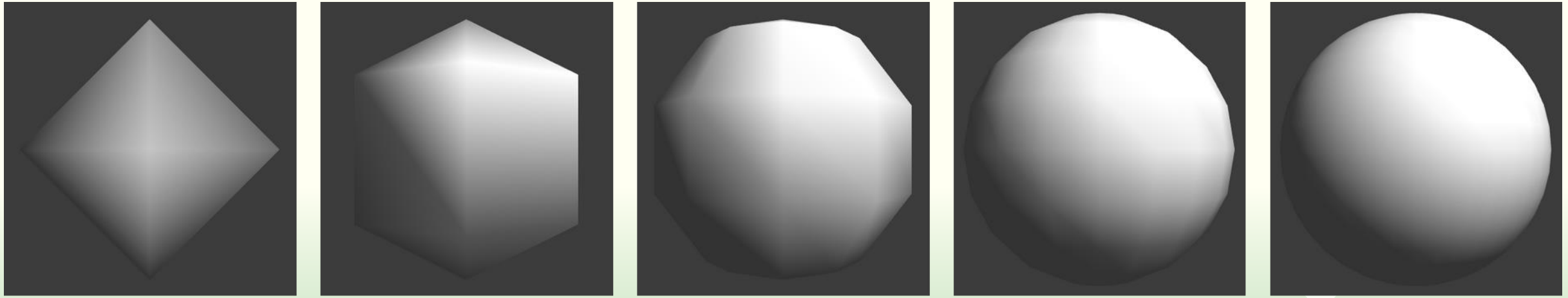
ofSpherePrimitive の set() メソッド

- void ofSpherePrimitive::set(float radius, int resolution, ofPrimitiveMode mode=OF_PRIMITIVE_TRIANGLE_STRIP)
  - radius: 球の半径
  - resolution: 球のメッシュの解像度
  - mode: 球を描画するのに使う図形要素、デフォルトの OF_PRIMITIVE_TRIANGLE_STRIP を使えばよいので省略する



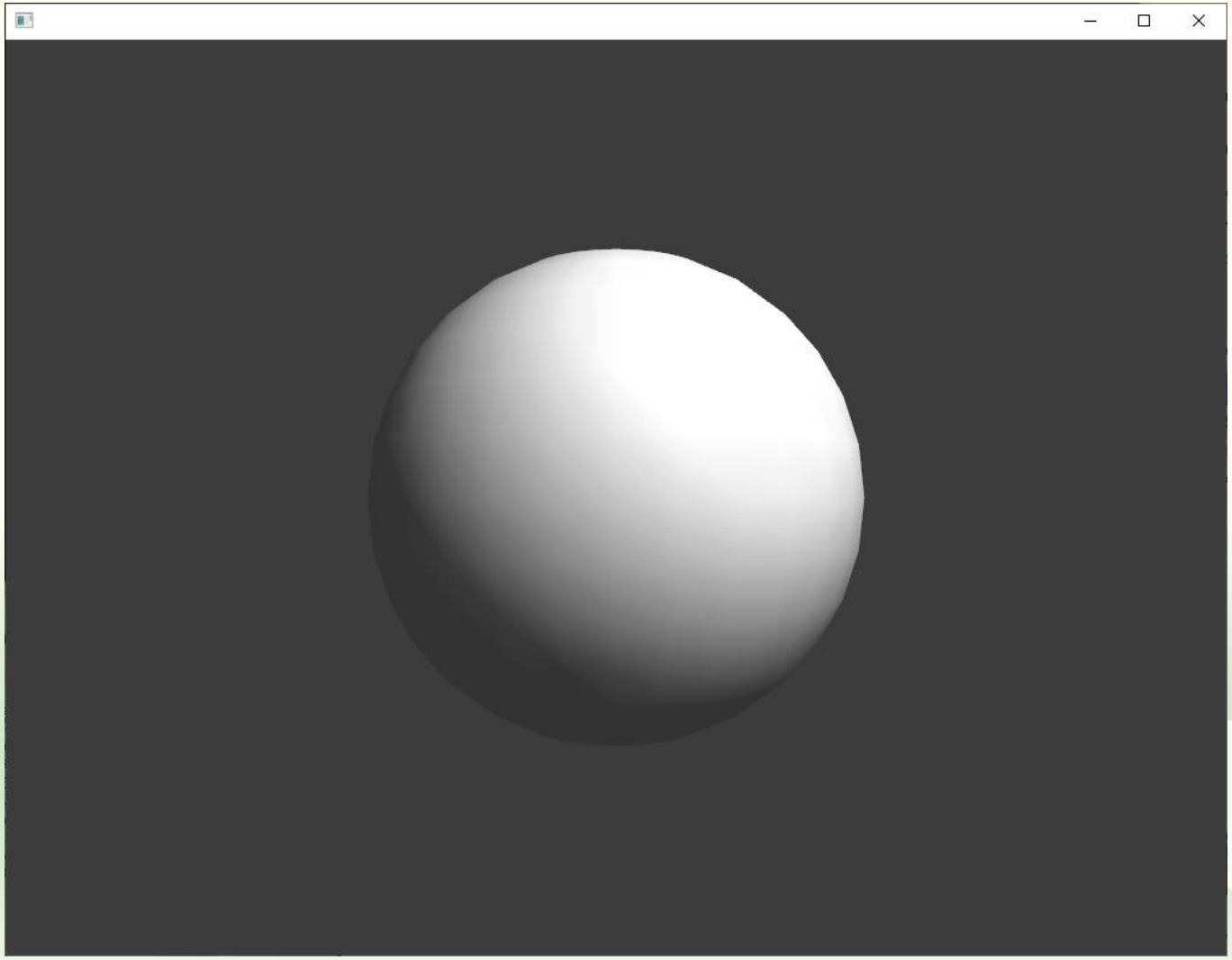
他にOF_PRIMITIVE_LINES_ADJACENCY, OF_PRIMITIVE_LINE_STRIP_ADJACENCY, OF_PRIMITIVE_TRIANGLES_ADJACENCY, OF_PRIMITIVE_TRIANGLE_STRIP_ADJACENCY, OF_PRIMITIVE_PATCHES

球の解像度 (resolution)



resolution = 2　resolution = 3　resolution = 5　resolution = 10　resolution = 20

結果は同じ



関数とオブジェクト

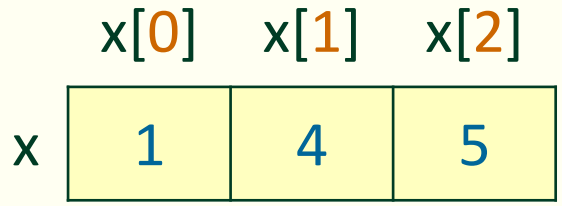
- 関数
  - void ofDrawSphere(float x, float y, float radius) は x, y, radius を指定することにより図形（球）を描画する
  - 関数は球の描き方を知っているが、どういう球を描けばいいか引数に（描画処理とは別に管理する）データを与える必要がある
- オブジェクト
  - sphere はそれ自体が球の描き方（メンバ関数＝メソッド）と、それに必要なデータ（メンバ変数）を保持している
  - オブジェクトは描くという行為を含めてデータ化できる

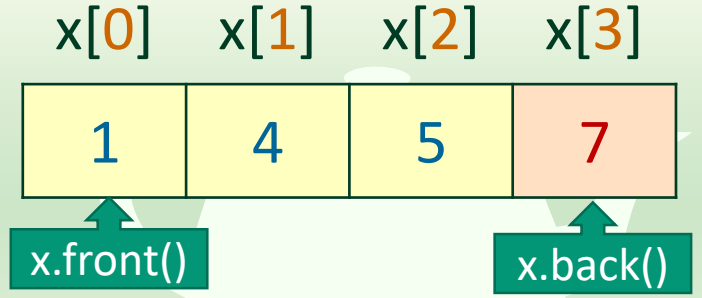
可変長配列

std::vector を使って複数のオブジェクトを描く

可変長配列 std::vector

- 複数のデータを保持できる（配列）
  - vector<int> x { 1, 4, 5 };
- 各要素は添え字でアクセスできる
  - x[0] → 1、x[1] → 4、x[2] → 5、x.size() → 3
- データを後から追加できる（可変長）
  - x.push_back(7);
    - x[3] → 7、x.size() → 4
    - x.front() → x[0]、x.back() → x[3]



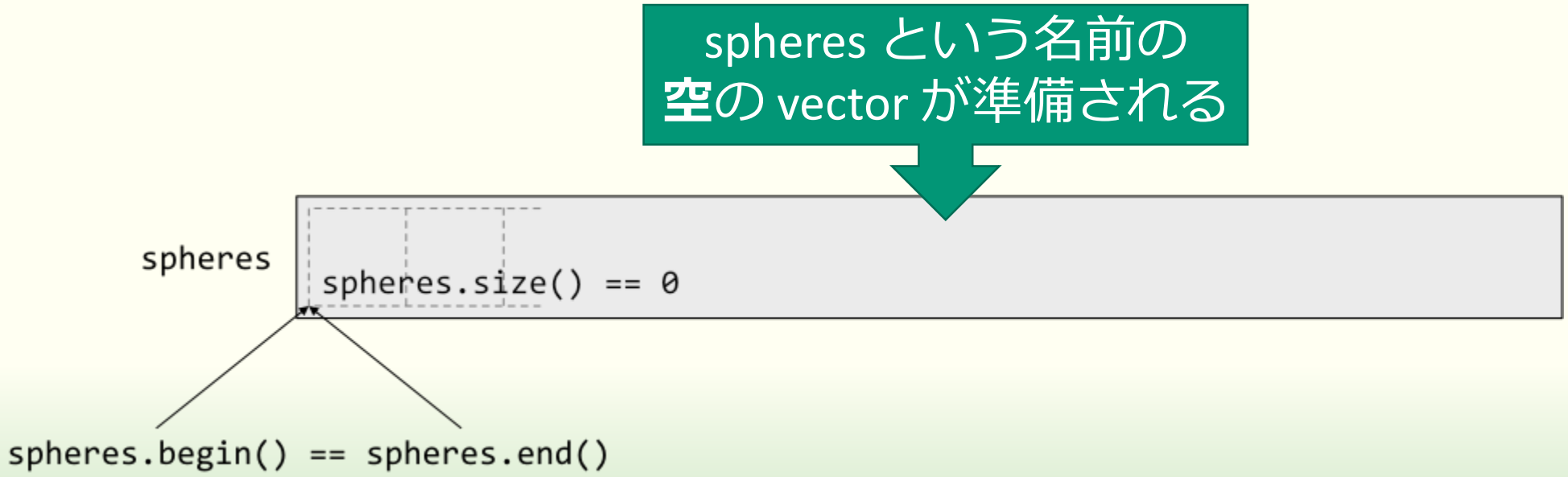


ofApp.h で球を保持する vector を用意する

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{
  ofCamera camera;
  ofLight light;
  vector<ofSpherePrimitive> spheres;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  （以下略）
```

- ofSpherePrimitive クラス（球）を **vector** にする
  - 複数の ofSpherePrimitive クラスのオブジェクトを保持する**コンテナ**（入れ物）ができる
  - 複数の球が入るので変数名は **spheres** にしている
- vector は標準ライブラリ
  - ofMain.h に using namespace std; が入っているので std:: は省略可能

vector<ofSpherePrimitive> spheres;



ofApp.cpp の setup() で vector に球を追加する

```
//--------------------------------------------
void ofApp::setup(){
  ofEnableDepthTest();
  camera.setPosition(0.0f, 0.0f, 100.0f);
  light.setPosition(60.0f, 80.0f, 100.0f);
  light.enable();

  ofSpherePrimitive sphere{ 10.0f, 20 };
  spheres.push_back(sphere);
}
```

半径

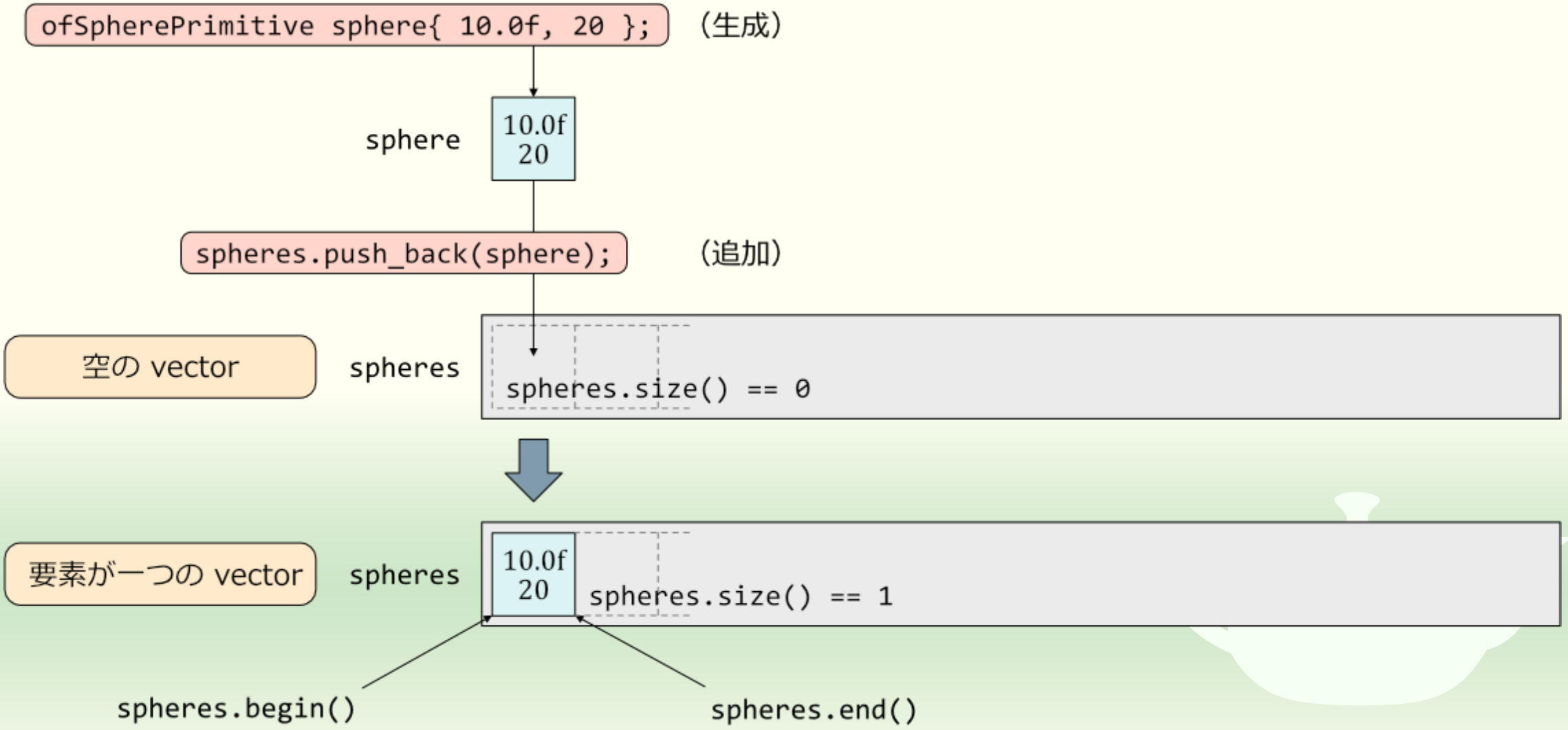
解像度

生成

spheres の
最後に追加

- setup() で球のオブジェクトを一つ生成する
  - ofSpherePrimitive クラスは変数宣言で**半径**と**解像度**を指定できる
- vector の spheres に生成した球を追加する
  - 生成されたオブジェクトのデータは vector の最後に追加されたメモリにコピーされる

spheres という vector に sphere を追加する



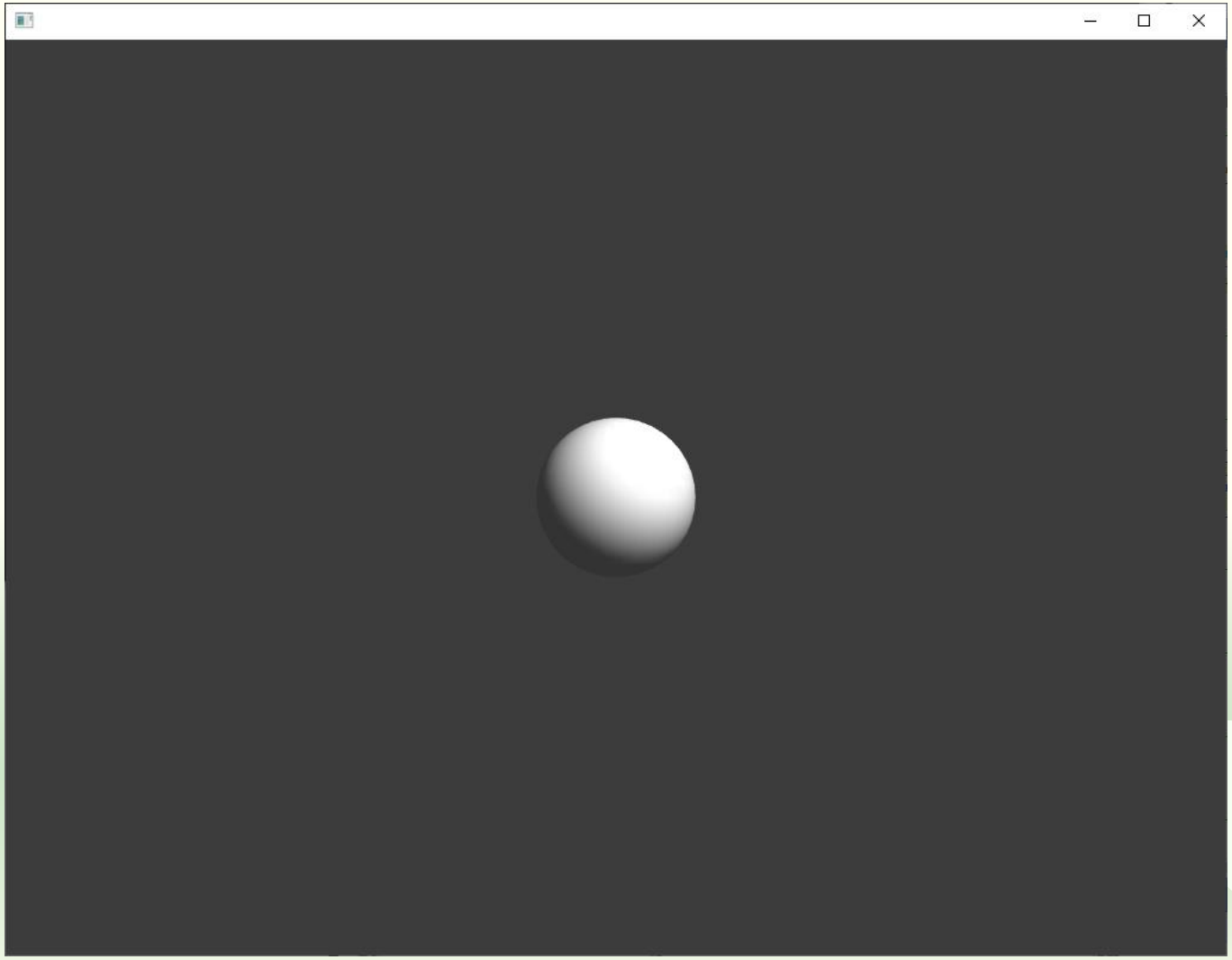
ofApp.cpp の draw() で vector の球を描画する

```
//--------------------------------------------
void ofApp::draw(){
  camera.begin();
  spheres.back().draw();
  camera.end();
}
```

spheres の最後の要素

- vector の spheres に最後に追加した球を描画する
  - **back()** メソッドは vector の最後の要素を取り出す
  - まだ球が一つしか入っていないので spheres.front() でも spheres[0] でも同じ
  - spheres[1] とか spheres[2] とかは実行時にエラーになる
    - spheres.size() で得られる vector の要素数よりも小さくないといけない

球が 1 個描かれる



setup() で球を上に移動して回転する

```
using namespace glm;

//--------------------------------------------
void ofApp::setup(){
  ofEnableDepthTest();
  camera.setPosition(0.0f, 0.0f, 100.0f);
  light.setPosition(60.0f, 80.0f, 100.0f);
  light.enable();

  ofSpherePrimitive sphere{ 10.0f, 20 };
  sphere.move(0.0f, 40.0f, 0.0f);
  sphere.rotateAroundDeg(60.0f,  // 回転角
    vec3{ 0.0f, 0.0f, 1.0f },    // 回転軸
    vec3{ 0.0f, 0.0f, 0.0f });   // 回転中心
  spheres.push_back(sphere);
}
```

- using namespace glm; を入れて vec3 を使うときに glm:: を省略できるようにする
- move() メソッドで球の高さを少し上げる
- rotateAroundDeg() メソッドは第３引数の位置を通る第２引数のベクトルを回転軸として第１引数の角度だけ回転する

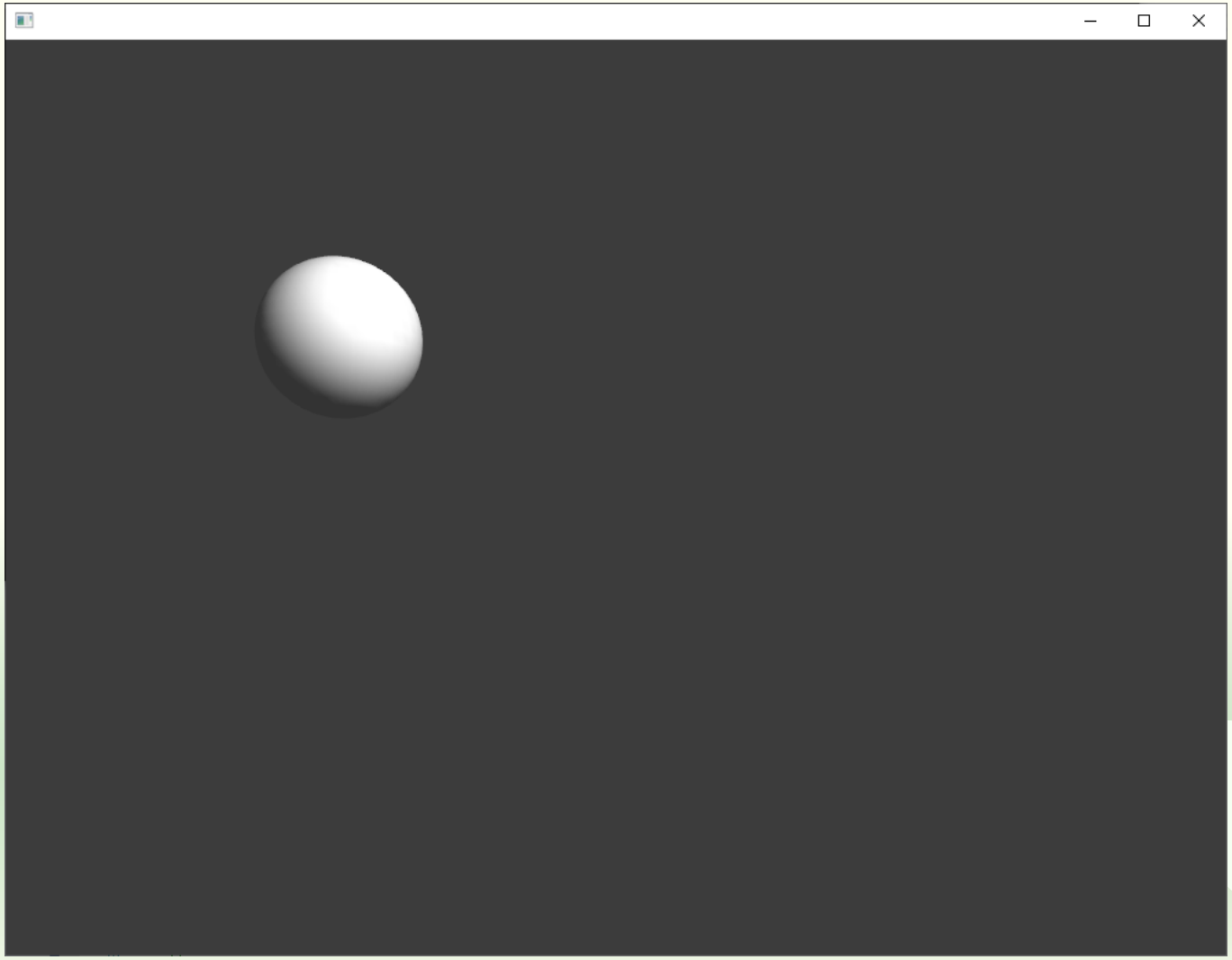
move() メソッド

- void ofNode::move(float x, float y, float z)
  - x, y, z: 物体の平行移動量
- void ofNode::move(const glm::vec3 &offset)
  - offset:物体の平行移動量のベクトル（vec3{ x, y, z }）
- 物体を**現在の位置から** x, y, z だけ平行移動する
  - setPosition() は指定した位置に配置する

rotateAroundDeg() メソッド

- void ofNode::rotateAroundDeg(float degrees, const glm::vec3 &axis, const glm::vec3 &point)
  - degrees: 回転角（度）
  - axis: 回転軸のベクトル（vec3{ vx, vy, vz }）
  - point: 回転中心（vec3{ px, py, pz }）
- 物体の姿勢を point を通り axis の方向を向いた軸を中心に degrees 度回転する
  - 回転角にラジアンが使いたければ rotateAroundRad() を使う

球が上に移動して左に傾く

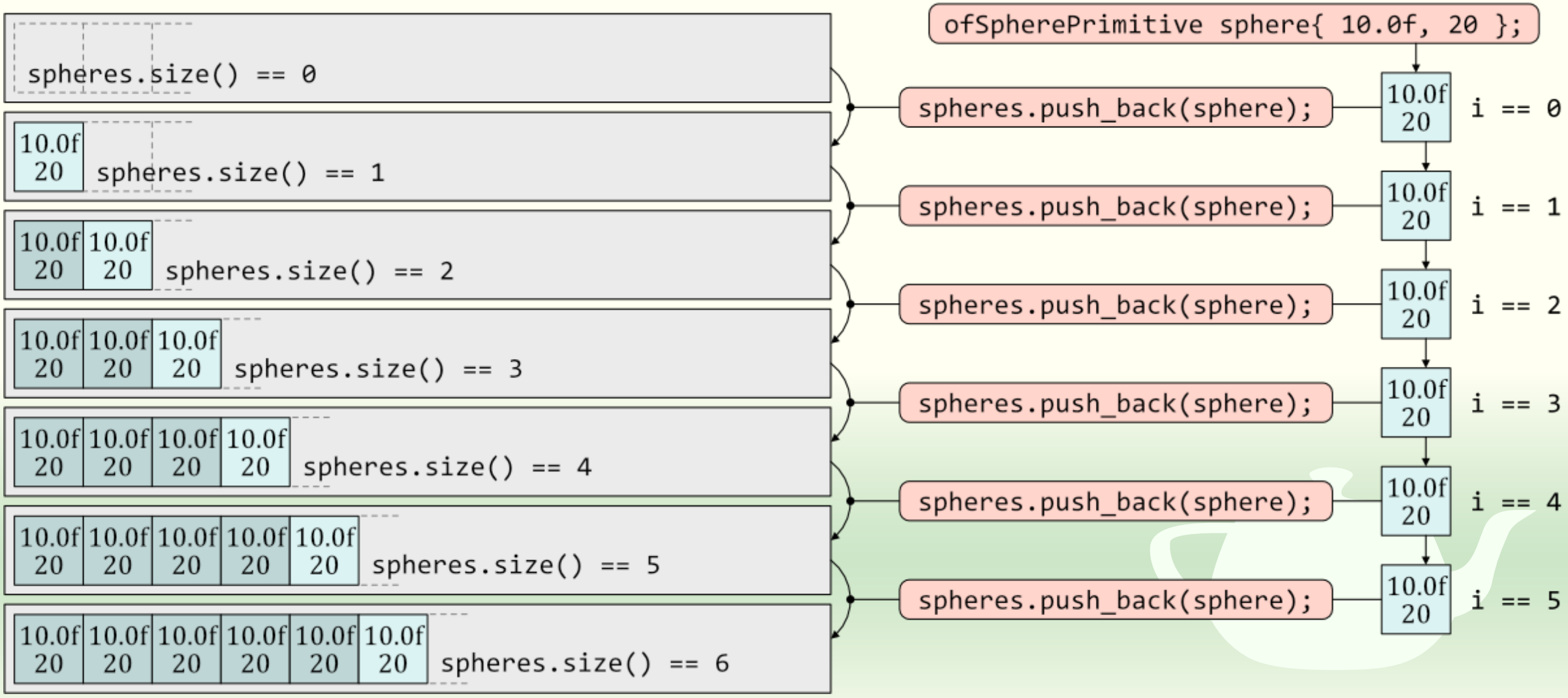


複数の球を回転しながら spheres に追加する

```
using namespace glm;

//--------------------------------------------
void ofApp::setup(){
  ofEnableDepthTest();
  camera.setPosition(0.0f, 0.0f, 100.0f);
  light.setPosition(60.0f, 80.0f, 100.0f);
  light.enable();

  for (int i = 0; i < 6; ++i){
    ofSpherePrimitive sphere{ 10.0f, 20 };
    sphere.move(0.0f, 40.0f, 0.0f);
    sphere.rotateAroundDeg(60.0f * i,
      vec3{ 0.0f, 0.0f, 1.0f },
      vec3{ 0.0f, 0.0f, 0.0f });
    spheres.push_back(sphere);
  }
}
```

- sphere の設定を行ってから push_back() する
  - 設定された内容で sphere が spheres に追加される
  - 角度が同じだと重なってしまうので60°ずつずらす
    - i は 0, 1, 2, 3, 4, 5 と変化するので **60 * i** は 0, 60, 120, 180, 240, 300, 360 になる
- 6 回繰り返す

spheres に複数の sphere を push_back() する



for 文による繰り返し処理

例

```
// 合計
int sum = 0;

// 1 から 5 までの合計を sum に求める
for (int i = 1; i <= 5; ++i) {
    sum += i;
}
```

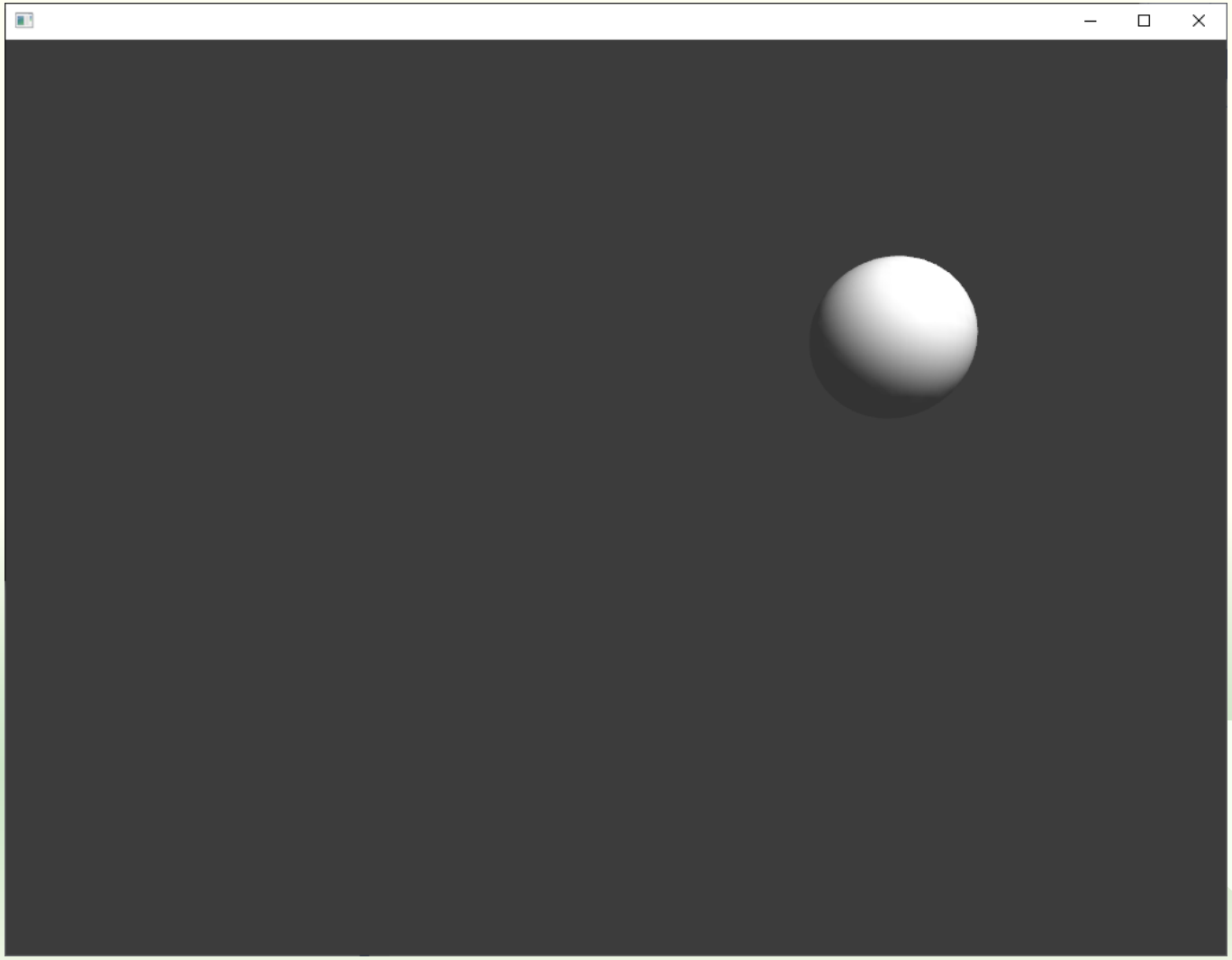
この処理が5回繰り返される

sum=0　i=1

sum	i	i <= 5	sum += i	++i
0	1	true	0+1	1+1
1	2	true	1+2	2+1
3	3	true	3+3	3+1
6	4	true	6+4	4+1
10	5	true	10+5	5+1
15	**6**	**false**	**おわり**	

- `int i = 1`
  - int 型の変数 i を宣言して 1 に初期化する
- `i <= 5`
  - i <= 5 が **true**（i が 5 以下）の間 { } 内を**繰り返す**
- `++i`
  - i に 1 を加算する

しかし最後の１個しか描いてくれない



draw() で vector が保持する全部の球を描画

```
//--------------------------------------------
void ofApp::draw(){
  camera.begin();
  for (auto &sphere : spheres){
    sphere.draw();
  }
  camera.end();
}
```

- vector の spheres には複数の球が入っている
- spheres から球を一つずつ sphere に取り出して描画する
  - sphere に**参照演算子 &** をつけているので sphere に入っているのは spheres の一つの要素の実体になる

vector で範囲ベースの for を使う

例

```
// 合計
int sum = 0;

// データ
std::vector<int> data { 1, 2, 3, 4, 5 };

// data に入っている全部の数値の合計を sum に求める
for (auto &i : data) {
   sum += i;
}
```

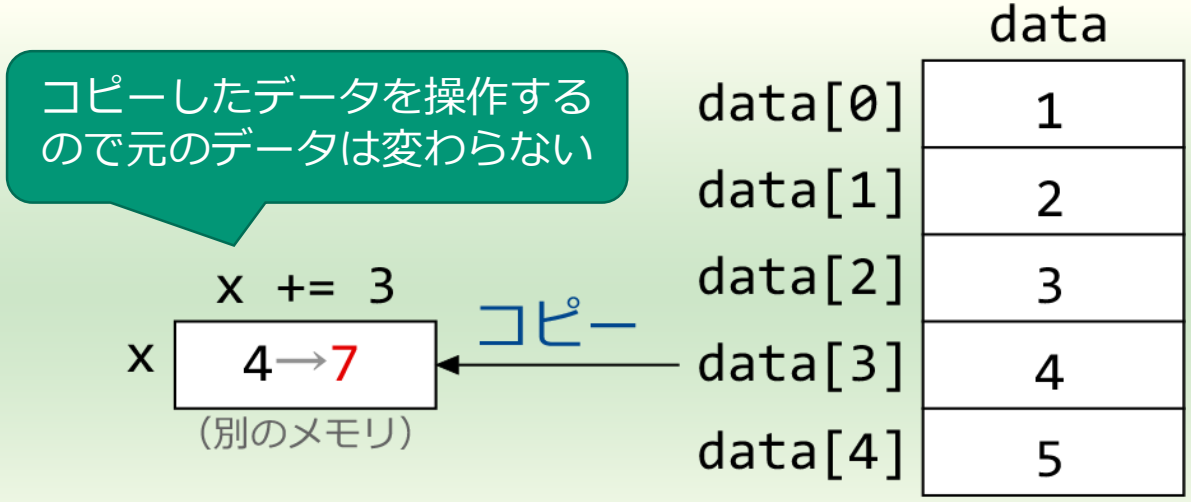
この処理がデータの個数回繰り返される

- data の要素を一つずつ取り出して i に入れる
- { } 内の処理をデータの数だけ繰り返す
- **auto**
  - 型推論、文脈から自動的にデータ型を推定する機能
  - 例のプログラムでは vector である data の一つの要素のデータ型 (int) になる

代入と参照

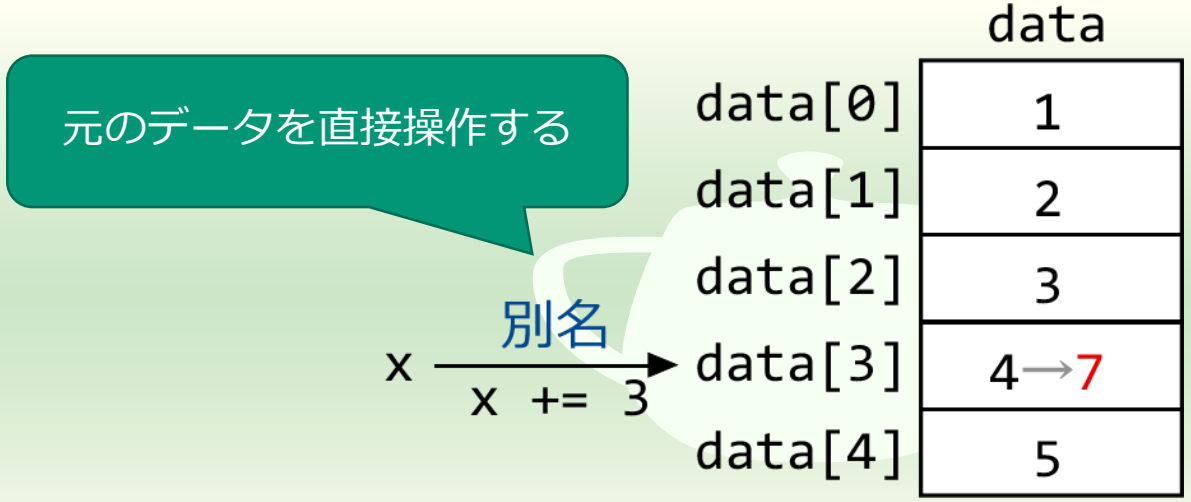
代入はデータのコピー

```
std::vector<int> data { 1,2,3,4,5 };

for (auto x : data){
  x += 3;
}
```

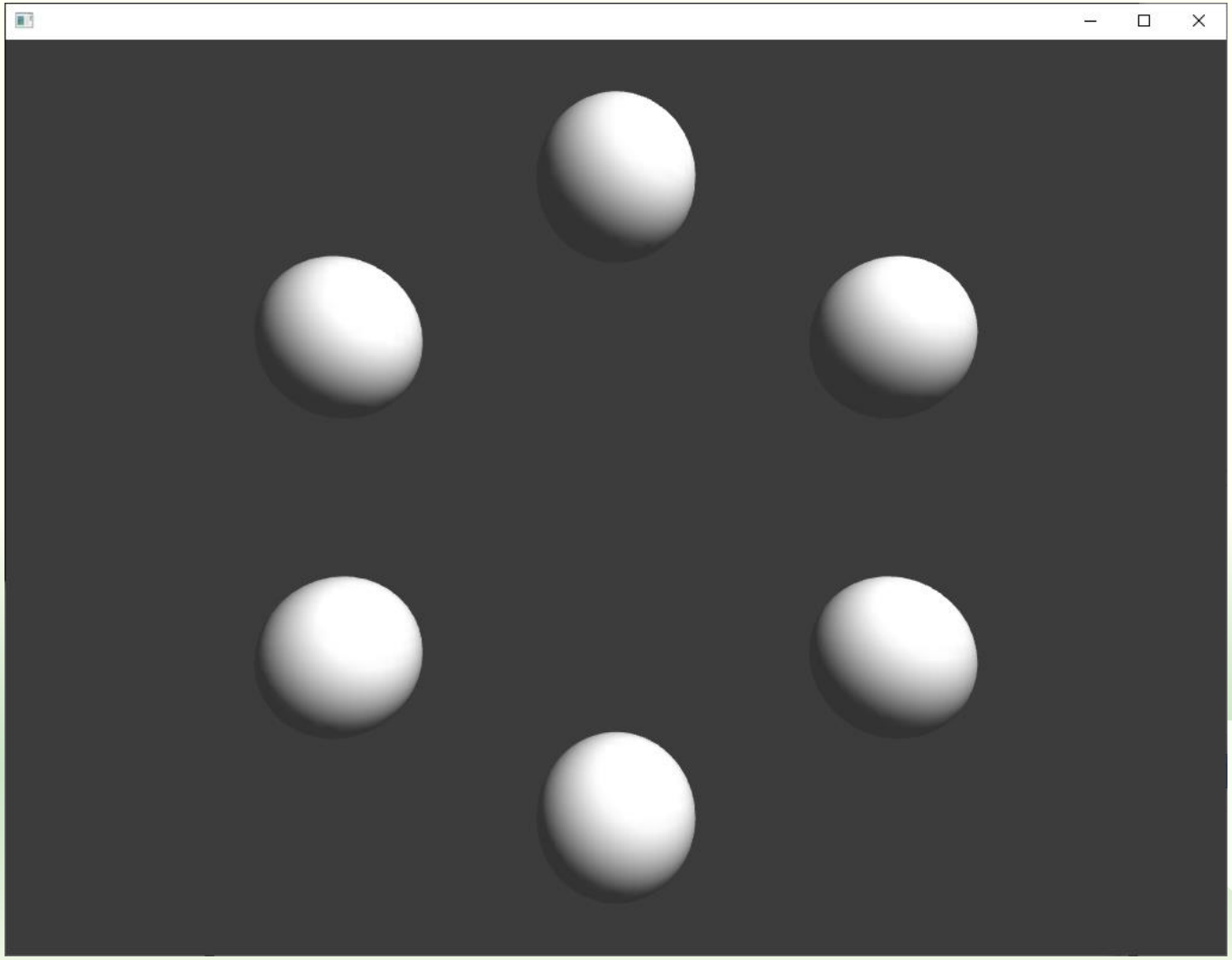


参照は格納場所の共有

```
std::vector<int> data { 1,2,3,4,5 };

for (auto &x : data){
  x += 3;
}
```



全部の球が描かれる



emplace_back() を使う

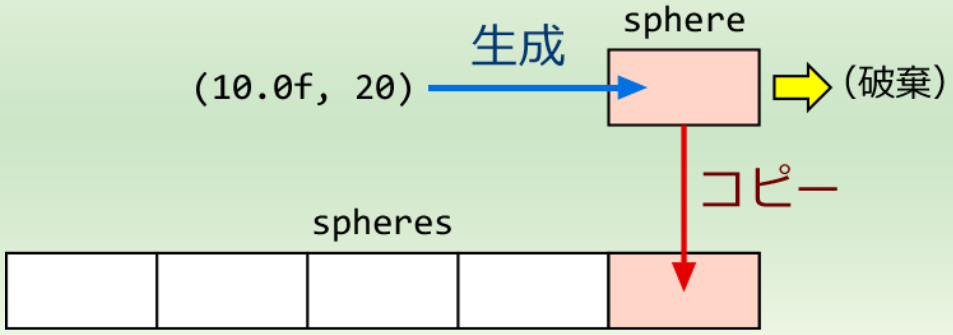
```
using namespace glm;

//--------------------------------------------------
void ofApp::setup(){
  ofEnableDepthTest();
  camera.setPosition(0.0f, 0.0f, 100.0f);
  light.setPosition(60.0f, 80.0f, 100.0f);
  light.enable();

  for (int i = 0; i < 6; ++i){
    spheres.emplace_back(10.0f, 20);
    spheres.back().setPosition(0.0f, 40.0f, 0.0f);
    spheres.back().rotateAroundDeg(60.0f * i,
      vec3{ 0.0f, 0.0f, 1.0f },
      vec3{ 0.0f, 0.0f, 0.0f });
  }
}
```

- emplace_back() メソッドは vector の最後に直接要素を生成する
  - 設定はback() メソッドで最後の要素取り出して行っている
    - 追加する要素を保持する単独の**変数がない**ため

push_back() と emplace_back()

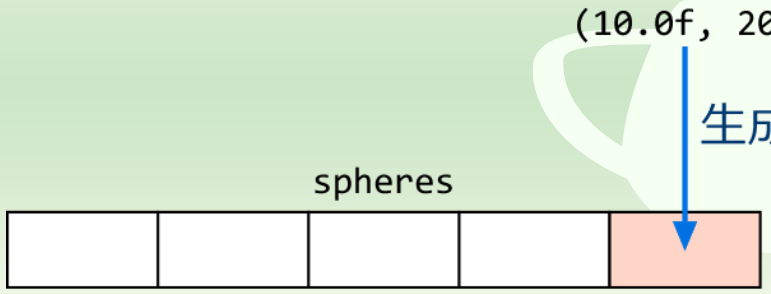
push_back()

```
// vector
std::vector<ofSpherePrimitive> spheres;

// 要素を一つ作成
ofSpherePrimitive sphere(10.0f, 20);

// 作成した要素を vector の最後にコピーして追加
spheres.push_back(sphere);
```

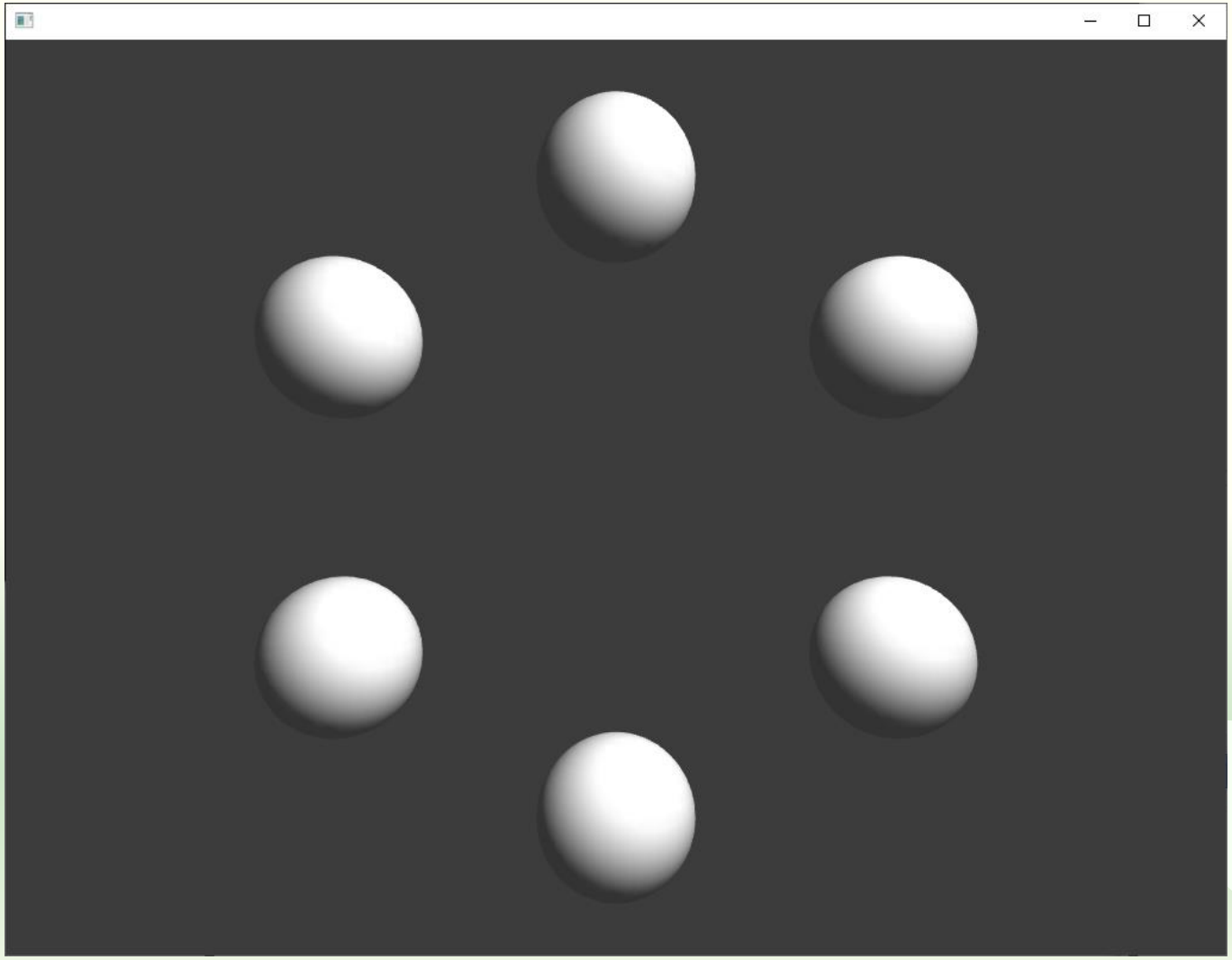


emplace_back()

```
// vector
std::vector<ofSpherePrimitive> spheres;



// vector の最後に要素を作成して追加
spheres.emplace_back(10.0f, 20);
```



特に変わらない



階層構造

図形の親子関係

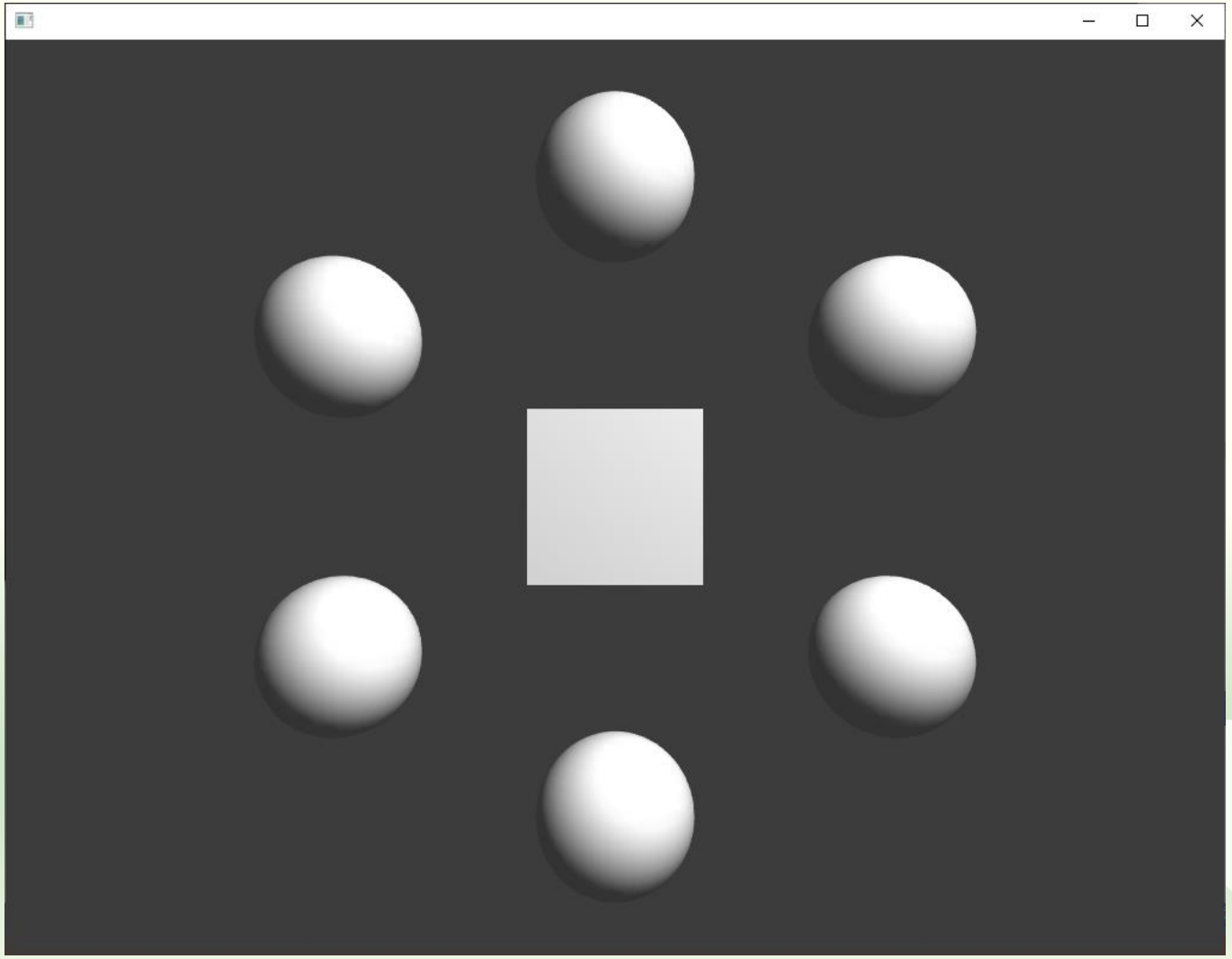
箱を一つ作成して draw() で描画する

幅　高さ　深さ

```
using namespace glm;

static ofBoxPrimitive box{ 20.0f, 20.0f, 20.0f };

 (途中略)

//--------------------------------------------
void ofApp::draw(){
  camera.begin();
  for (auto &sphere : spheres){
    sphere.draw();
  }
  box.draw();
  camera.end();
}
```

- **ofBoxPrimitive** は箱のクラス
  - 変数宣言の時に箱の幅、高さ、深さを指定できる
  - ofApp.h いじるのが面倒なので ofApp.cpp 内に static で宣言する
- draw() で描画する
  - box の draw() メソッド

箱が追加される



update() で箱を回転する

```
using namespace glm;

static ofBoxPrimitive box{ 20.0f, 20.0f, 20.0f };

 (途中略)

//--------------------------------------------
void ofApp::update(){
  box.rotateDeg(1.0f, 0.0f, 0.0f, 1.0f);
}
```

回転角（度）

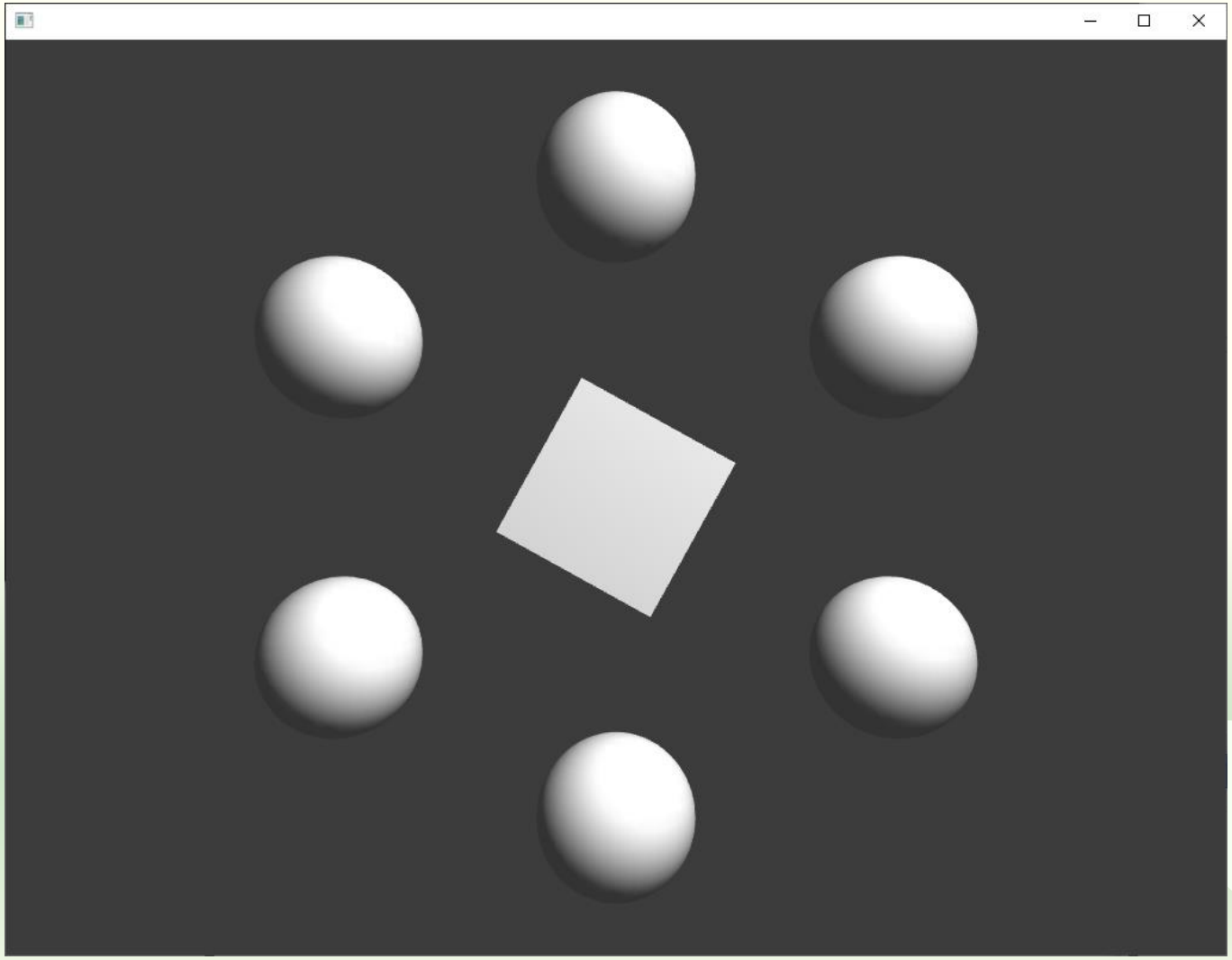
回転軸

- update() で箱を回転する
- rotateDeg() メソッド
  - 物体を**現在の状態から**指定した回転軸を中心に指定した回転角だけ回転する
  - 回転角の単位は度
    - ラジアンで指定したいときは rotateRad() メソッドを使う

rotateDeg() メソッド

- void ofNode::rotateDeg(float degrees, float vx, float vy, float vz)
  - degrees: 回転角（度）
  - vx, vy, vz: 回転軸
- void ofNode::rotateDeg(float degrees, const glm::vec3 &v)
  - degrees: 回転角（度）
  - v: 回転軸（vec3{ vx, vy, vz}）

箱だけが回転する

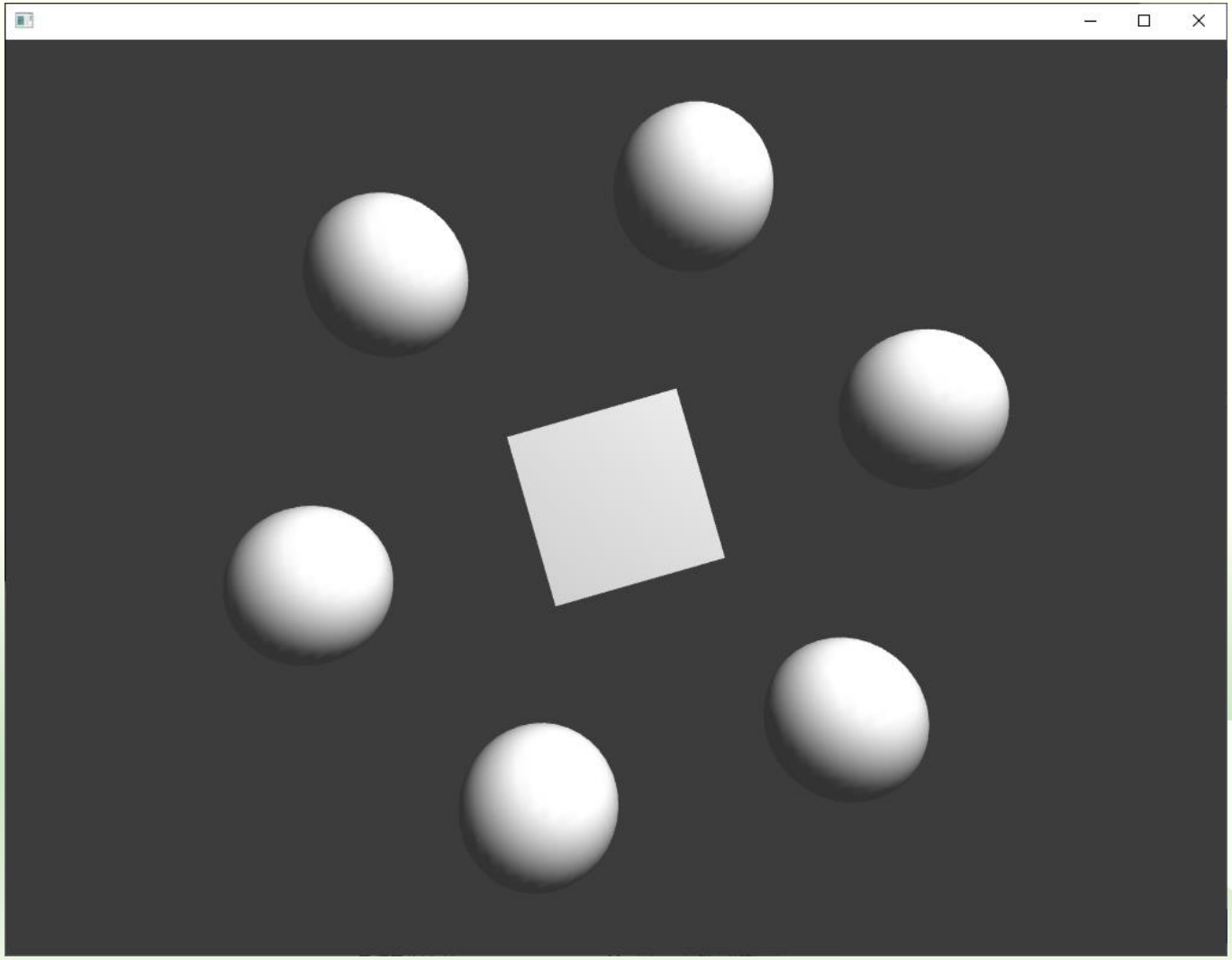


箱をすべての球の親にする

```
using namespace glm;

static ofBoxPrimitive box{ 20.0f, 20.0f, 20.0f };

//--------------------------------------------------
void ofApp::setup(){
  ofEnableDepthTest();
  camera.setPosition(0.0f, 0.0f, 100.0f);
  light.setPosition(60.0f, 80.0f, 100.0f);
  light.enable();

  for (int i = 0; i < 6; ++i){
    spheres.emplace_back(10.0f, 20);
    spheres.back().setPosition(0.0f, 40.0f, 0.0f);
    spheres.back().rotateAroundDeg(60.0f * i,
      vec3{ 0.0f, 0.0f, 1.0f },
      vec3{ 0.0f, 0.0f, 0.0f });
    spheres.back().setParent(box);
  }
}
```

- setParent() メソッドは物体の「親」になる物体を指定する
  - 「子」の物体の位置や姿勢は親の物体を基準にして決定される

全体が回転する



ofApp.h の vector に箱も入れるようにする

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{
  ofCamera camera;
  ofLight light;
  vector<unique_ptr<of3dPrimitive>> parts;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  （以下略）
```

- parts という vector にオブジェクトの**ポインタ**を保存する
- **unique_ptr**< … > は他の変数と同じポインタを格納しないポインタのコンテナのデータ型
  - この変数を削除すると入っているポインタが指す実体も削除される
- **of3dPrimitive** は ofBoxPrimitive や ofSpherePrimitive の**基底クラス**
  - 基底クラスのコンテナには**派生クラス**のポインタを格納できる

球や箱は動的に生成する

```
using namespace glm;

static ofBoxPrimitive box{ 20.0f, 20.0f, 20.0f };

//--------------------------------------------------------------
void ofApp::setup(){
  ofEnableDepthTest();
  camera.setPosition(0.0f, 0.0f, 100.0f);
  light.setPosition(60.0f, 80.0f, 100.0f);
  light.enable();

  parts.emplace_back(new ofBoxPrimitive{ 20.0f, 20.0f, 20.0f });

  for (int i = 0; i < 6; ++i){
    parts.emplace_back(new ofSpherePrimitive{ 10.0f, 20 });
    parts.back()->setPosition(0.0f, 40.0f, 0.0f);
    parts.back()->rotateAroundDeg(60.0f * i,
      vec3{ 0.0f, 0.0f, 1.0f },
      vec3{ 0.0f, 0.0f, 0.0f });
    parts.back()->setParent(*parts.front());
  }
}
```

削除

parts の最初には箱のオブジェクトのポインタを入れる

箱のポインタが指す実体を取り出す

- **new** はオブジェクトの生成（メモリの確保と初期化）を行ってデータの格納場所（**ポインタ**）を返す**演算子**
- “->” はポインタが指しているオブジェクトのメンバを示す**アロー演算子**

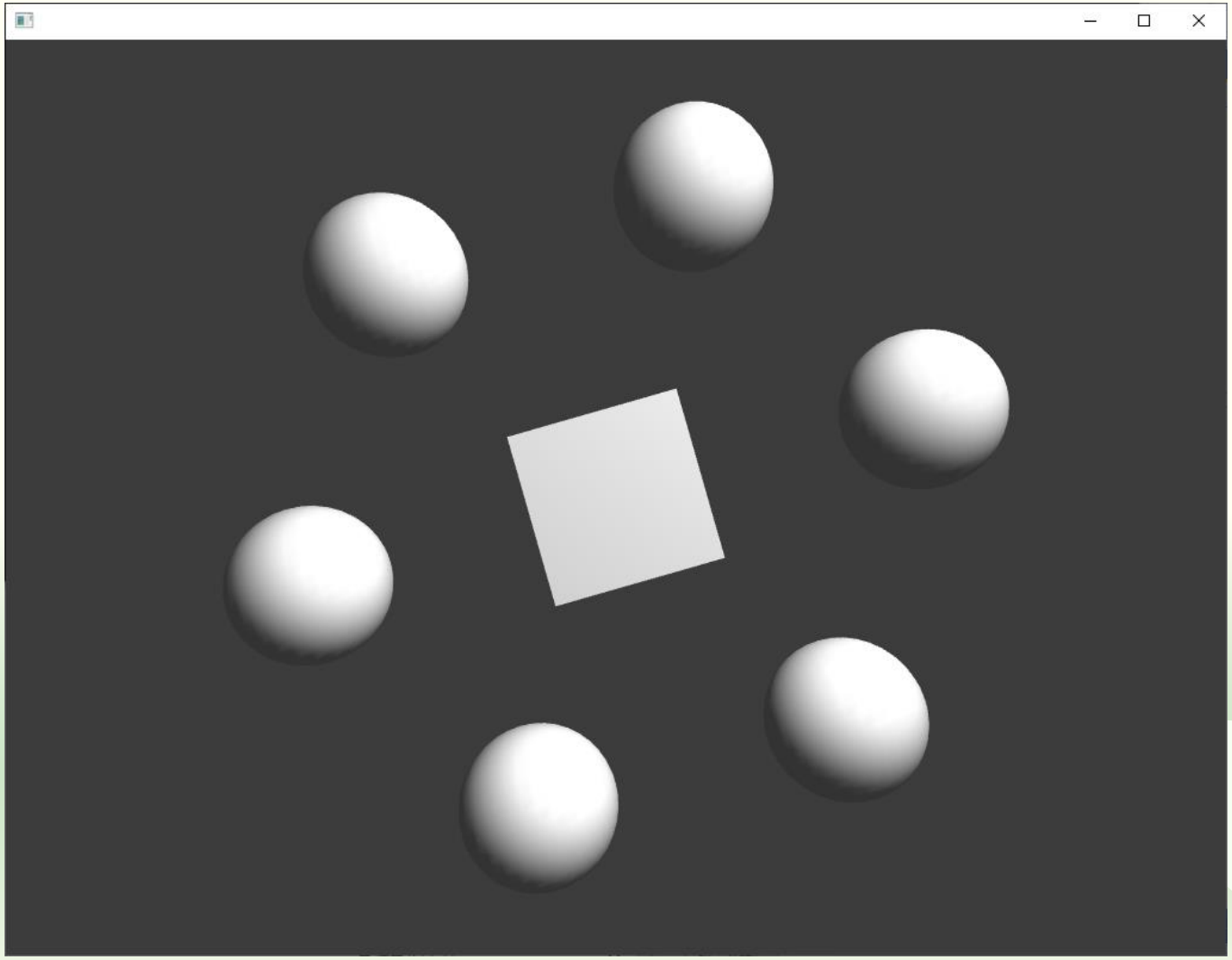
箱も球も parts という vector に入っている

```
//--------------------------------------------------------------
void ofApp::update(){
  parts.front()->rotateDeg(1.0f, 0.0f, 0.0f, 1.0f);
}

//--------------------------------------------------------------
void ofApp::draw(){
  camera.begin();
  for (auto &part : parts){
    part->draw();
  }
  box.draw();
  camera.end();
}
```

削除

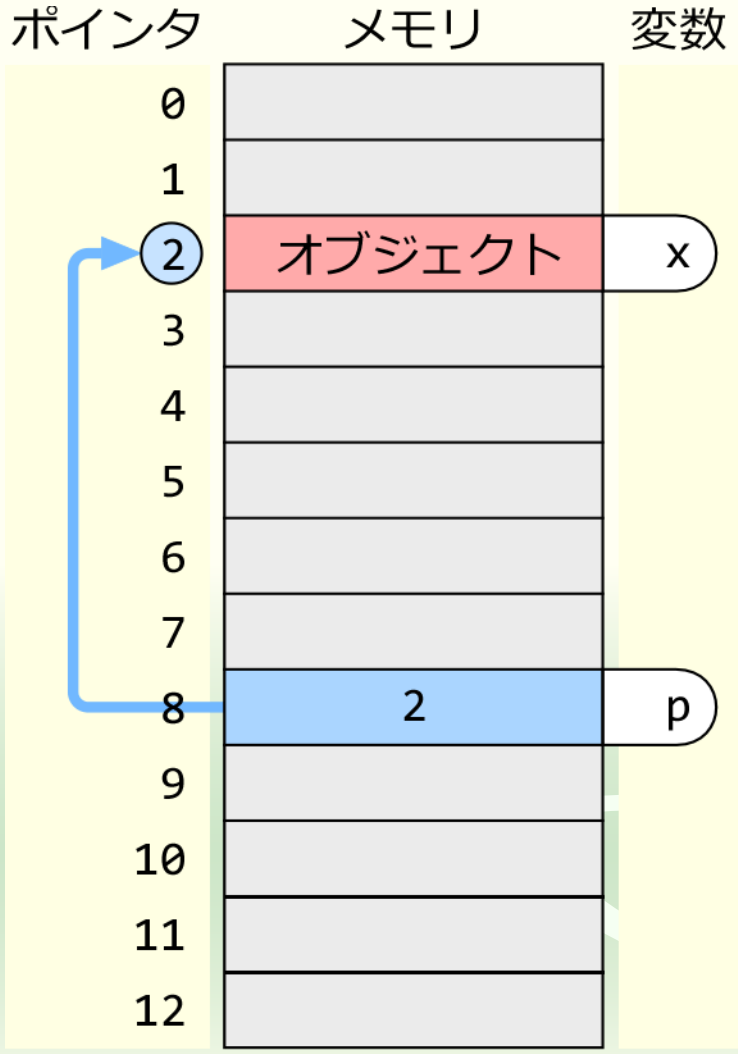
- parts.front() には parts に一番最初に 追加した箱 ofBoxPrimitive のポインタが入っている
- part->draw() で箱も球も混在して描画される

box.draw() がなくても箱が描かれる



オブジェクトとポインタ

- オブジェクト
  - クラスで定義された構造を持つメモリ上のデータ
- 変数
  - データを格納するために確保したメモリの領域に名前（変数名）を付けたもの
- ポインタ
  - データの格納場所を示すデータ



変数 x のメモリの場所②がポインタである

これは参照演算子 & を使って取り出せる

&x → ②

変数 p には変数 x のメモリの場所②がデータとして格納されているとする

p ← &x

変数 p に格納されているポインタ②を使って変数 x に格納されているオブジェクトは関節参照演算子 * を使って取り出せる

*p → オブジェクト

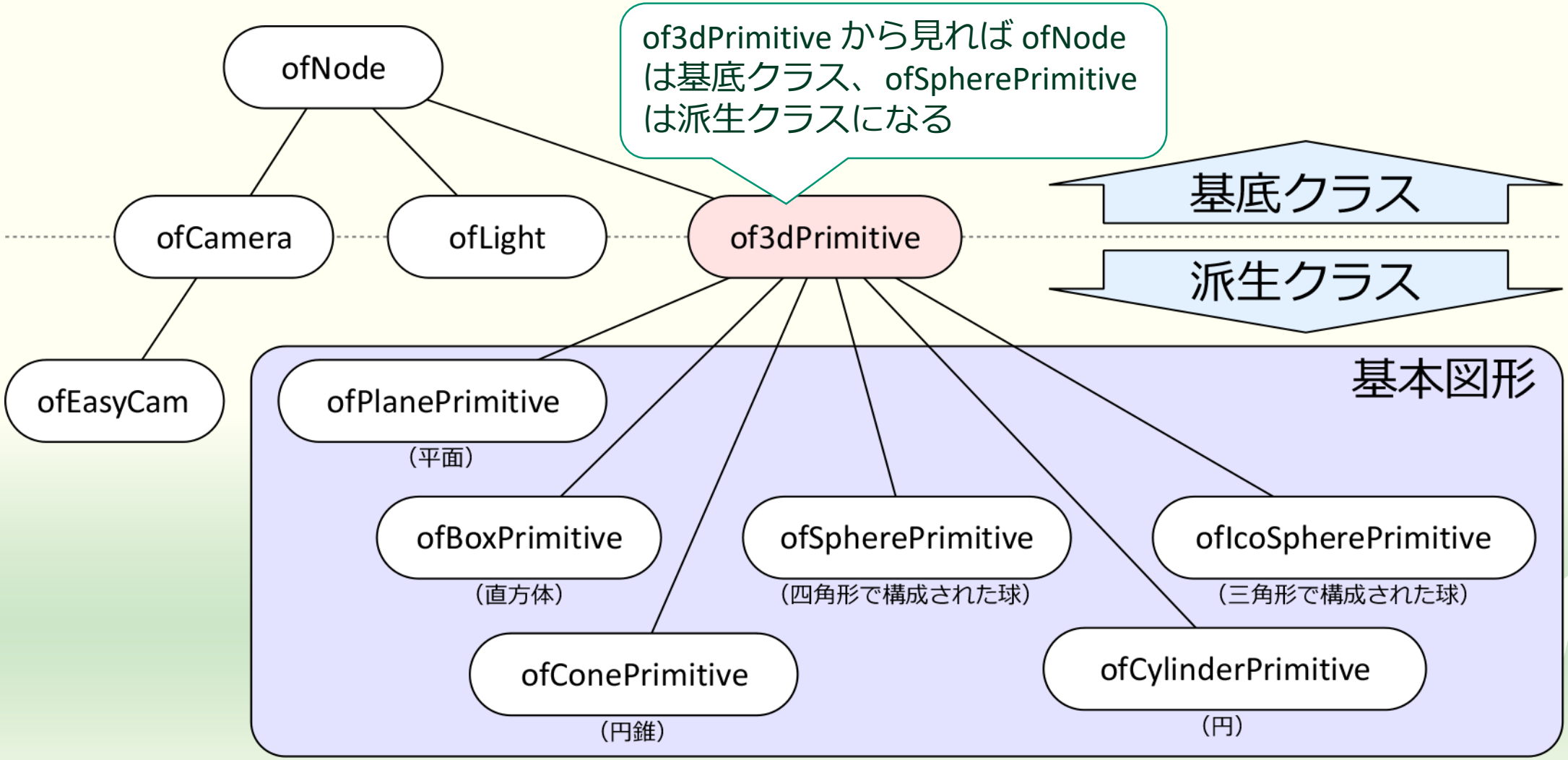
変数と new 演算子

- 変数のオブジェクト
  - 変数宣言によってメモリが確保されそこに生成される
  - 変数のスコープから外れると消去されメモリが解放される
- new 演算子によるオブジェクト
  - new 演算子によりメモリが確保されそこに生成される
  - delete 演算子により消去されメモリが解放される
  - new 演算子はポインタを返すがメモリに**変数名は付かない**
    - ポインタを保存しておかないと使えないし後で削除できない

new 演算子と std::unique_ptr

- new 演算子で生成したオブジェクトは使わなくなったら delete 演算子で**削除**しないといけない
  - delete 演算子により**後始末**も実行される
  - 削除し忘れると使えないメモリが残る（**メモリリーク**）
  - new 演算子が返したポインタを失うと削除できない
- ポインタを失ったらオブジェクトも削除したい
  - ポインタを **std::unique_ptr** 型の変数に入れておけば変数が削除された（スコープを外れた）ときにそのポインタが指しているオブジェクトも自動的に削除される（**スマートポインタ**）

of3dPrimitive



vector<**unique_ptr**<of3dPrimitive>> parts;

- of3dPrimitive クラスのスマートポインタのコンテナの vector
  - parts に格納した of3dPrimitive クラスのポインタが指すオブジェクトは parts が削除されるときに一緒に削除される
- 基底クラスのポインタのコンテナには派生クラスのポインタを格納できる
- parts へのポインタの追加方法の実際
  - parts.emplace_back(new **ofBoxPrimitive**{ 20.0f, 20.0f, 20.0f });
  - parts.emplace_back(new **ofSpherePrimitive**{ 10.0f, 20 });
    - ofBoxPrimitive と ofSpherePrimitive はどちらも of3dPrimitive の**派生クラス**なので of3dPrimitive のポインタのコンテナに格納できる

parts.**back**()->setParent(*parts.**front**());

- parts.back() は最後 parts に追加したデータ（球）のポインタを取り出す
- parts.front() は最初に parts に追加したデータ（箱）のポインタを取り出す
- *（関節参照演算子）はポインタが指しているデータの実体（データを格納しているメモリ）を取り出す
- つまり**箱**のデータを**球の親**に設定している

for (auto &part : parts) { **part->draw**(); }

- part には parts の要素の of3dPrimitive クラスまたはそれを継承したクラスのスマートポインタである
- draw() は of3dPrimitive のメソッドである
- of3dPrimitive クラスを継承している ofSpherePrimitive クラスや ofBoxPrimitive クラスの draw() メソッドは of3dPrimitive クラスのもの

課題３－１

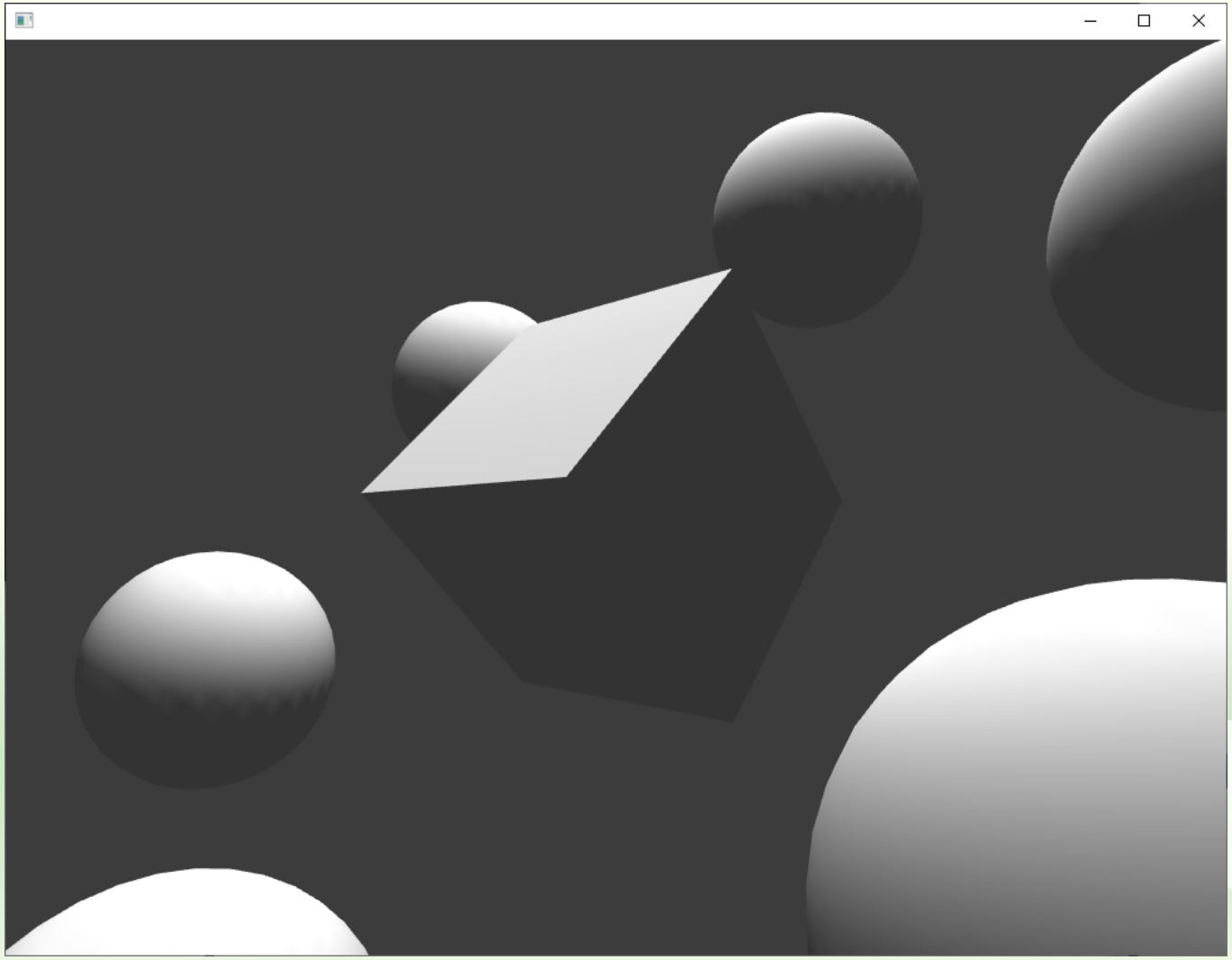
左腕を振る

ofApp.h の ofCamera を ofEasyCam に替える

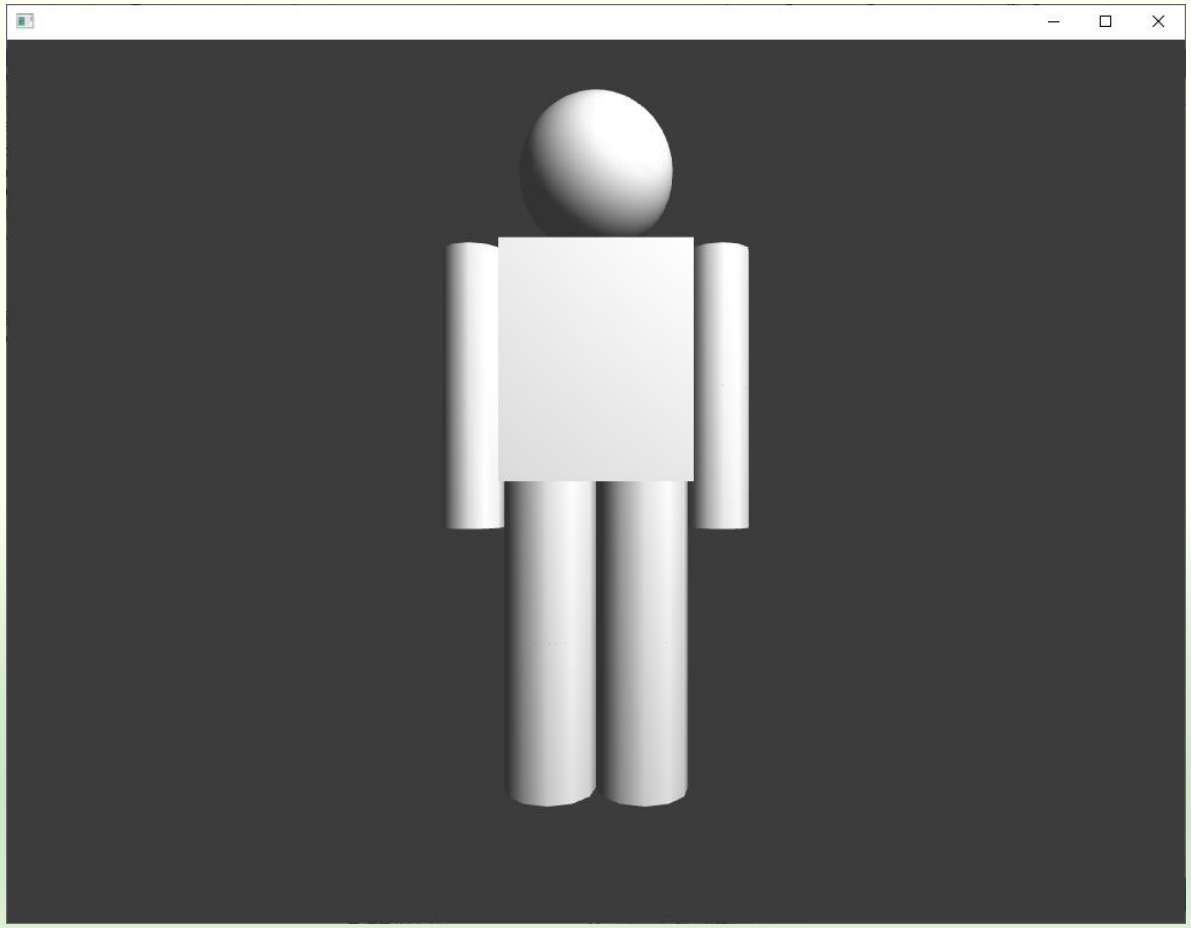
```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{
  ofEasyCam camera;
  ofLight light;
  vector<unique_ptr<of3dPrimitive>> parts;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
   (以下略)
```

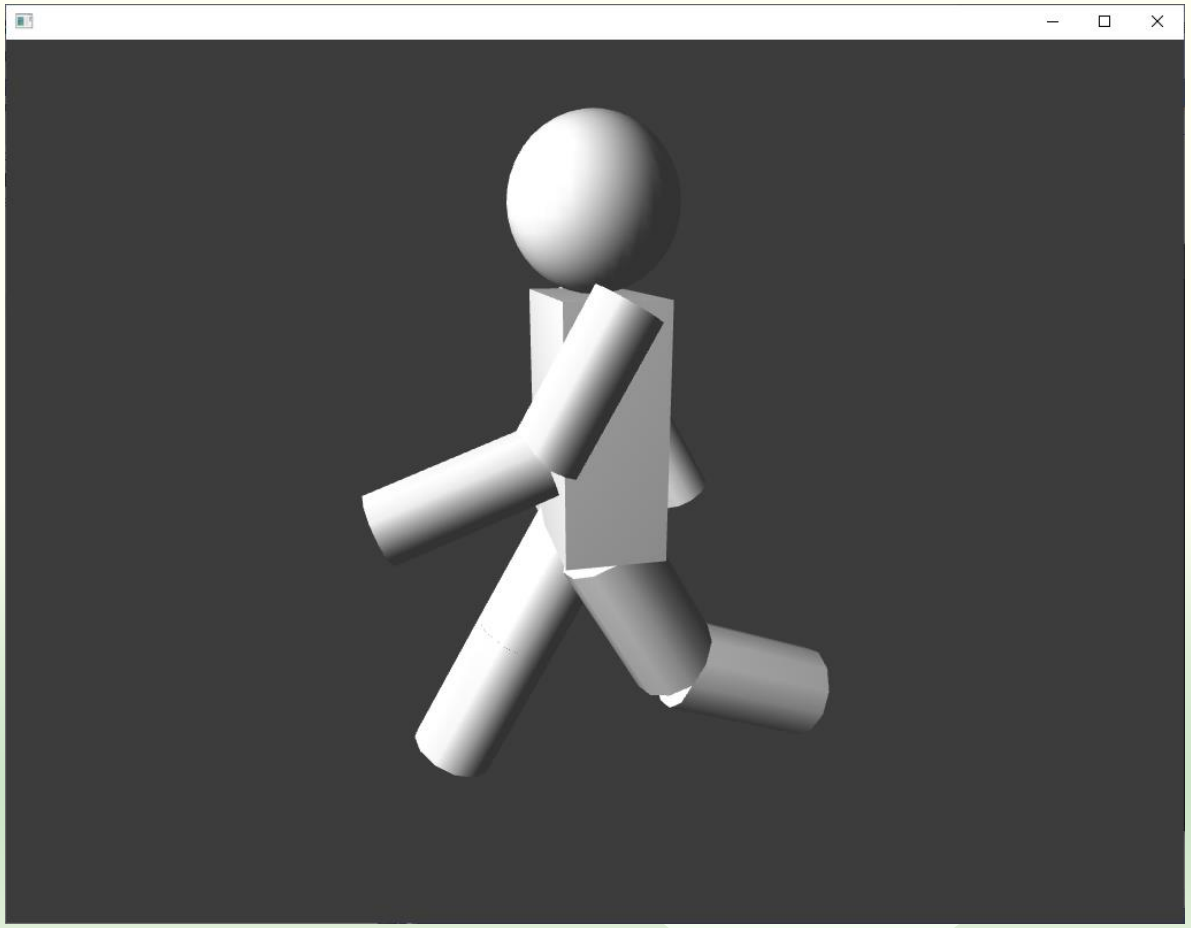
- **ofEasyCam** はマウス操作で視点位置を変更できる
  - 左ドラッグ：回転
  - 中ドラッグ：平行移動
  - 右・ホイール：前後移動
- ofCamera から**派生**したクラス
  - ofCamera を継承している

ofEasyCam でカメラの位置を変更する

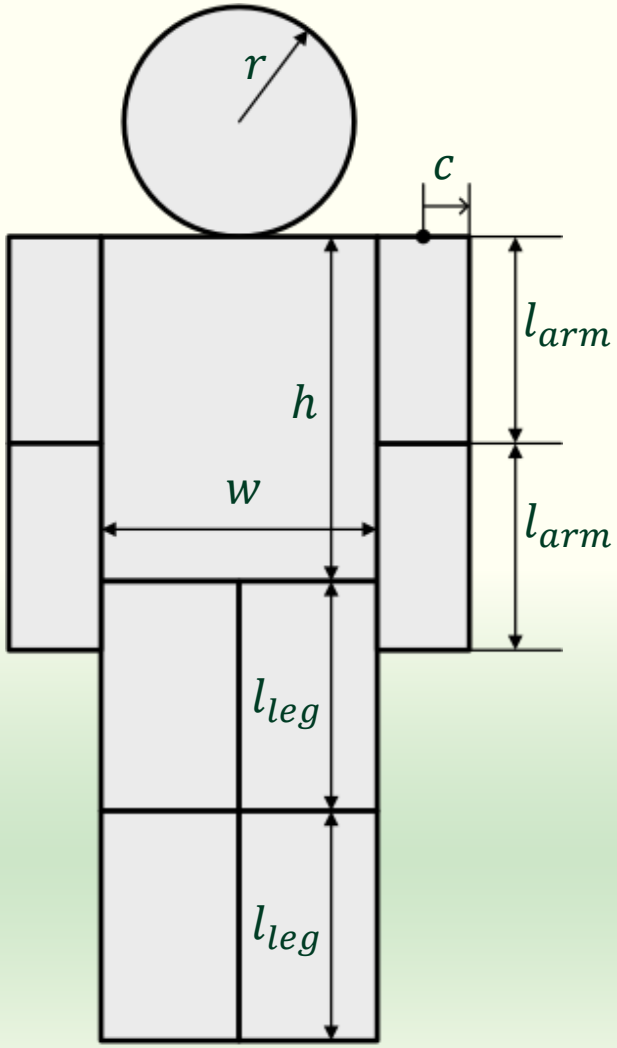


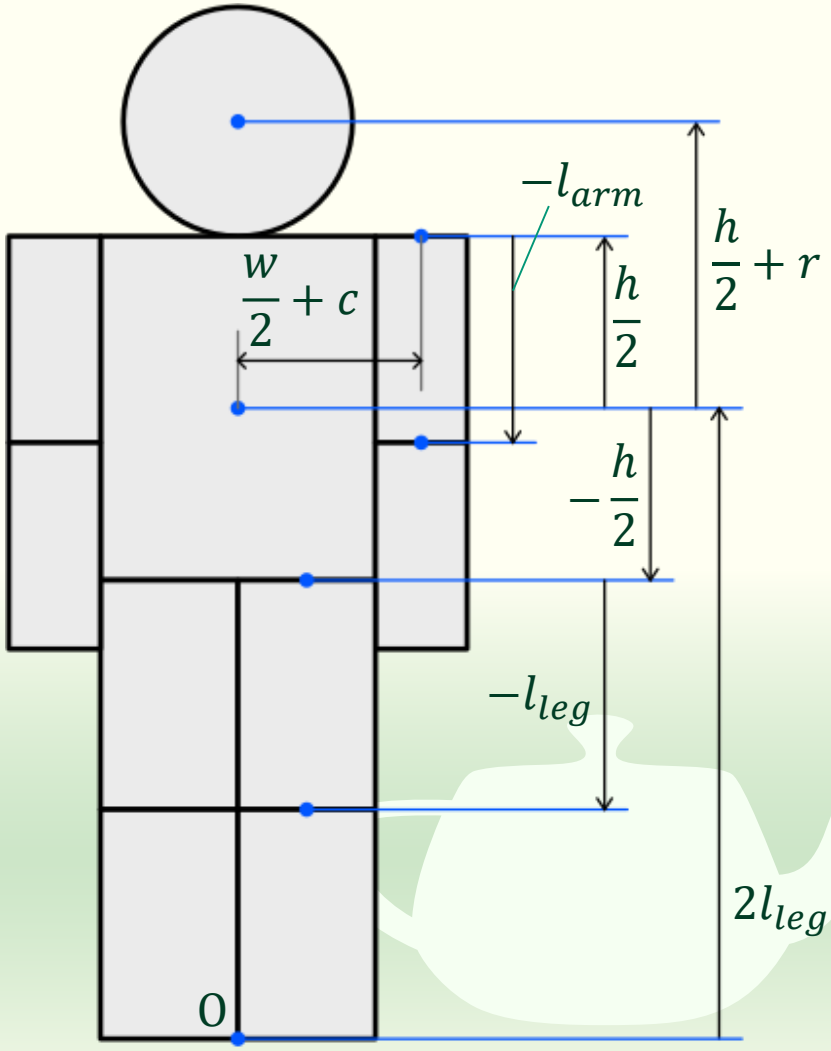
表示する形状をこういう形に作り替える





各パーツのサイズと位置





胴を parts に追加する

```
#include "ofApp.h"

using namespace glm;

//--------------------------------------------------
void ofApp::setup(){
  ofEnableDepthTest();
  camera.setPosition(0.0f, 0.0f, 100.0f);
  light.setPosition(60.0f, 80.0f, 100.0f);
  light.enable();

  // 0: 胴
  auto body = new ofBoxPrimitive{ w, h, d };
  parts.emplace_back(body);
  parts.back()->move(0.0f, 2 l_leg + h / 2, 0.0f);
```

- ofBoxPrimitive のオブジェクトを生成する
  - ポインタを body に入れておく
  - body を parts に追加する
    - 変数 body は setup() が終了すると失われるが parts にいれるので問題ない
  - w, h, d（d はパーツの奥行あるいは厚さ）や l_{leg}（腿・脛の長さ）は適当に決めてください
  - $2\,l_{leg} + h / 2$ は自分で計算してください

頭を parts に追加して胴体を親にする

```
#include "ofApp.h"

using namespace glm;

//--------------------------------------------------
void ofApp::setup(){
  ofEnableDepthTest();
  camera.setPosition(0.0f, 0.0f, 100.0f);
  light.setPosition(60.0f, 80.0f, 100.0f);
  light.enable();

  // 0: 胴
  auto body = new ofBoxPrimitive{ w, h, d };
  parts.emplace_back(body);
  parts.back()->move(0.0f, 2 l_leg + h / 2, 0.0f);

  // 1: 頭
  parts.emplace_back(new
    ofSpherePrimitive{ r, 20 });
  parts.back()->move(0.0f, h / 2 + r, 0.0f);
  parts.back()->setParent(*parts.front());
```

- ofSpherePrimitive のオブジェクトを生成して parts に追加する
  - ポインタを body に入れておく
  - body という変数は setup() が終了すると失われてしまうが parts にいれてしまうので問題ない
  - r を適当に決めて $h / 2 + r$ は自分で計算してください
- parts.front() は parts の先頭の要素の胴体のポインタを取り出す
- setParent() で胴体を親にする

時間の計測の準備

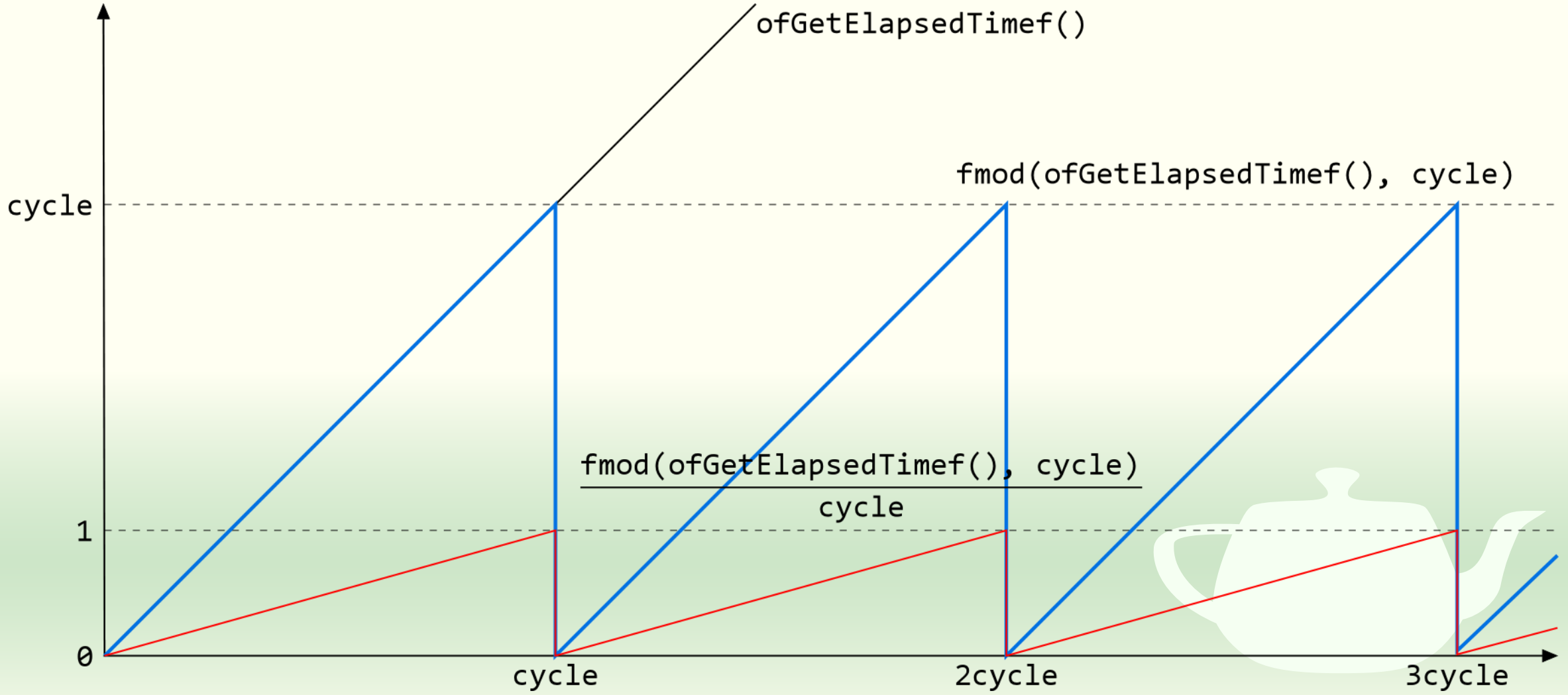
```
#include "ofApp.h"

using namespace glm;

const double cycle = 2.0;

//--------------------------------------------
void ofApp::setup(){
   (途中略)
}

//--------------------------------------------
void ofApp::update(){
  const float t = TWO_PI
    * fmod(ofGetElapsedTimef(), cycle) / cycle;
```

アニメーションの周期

t は 2 秒間で 0 → 2π に変化する

- openFrameworks には円周率 π が **PI**、2π が **TWO_PI** という記号定数で定義されている
  - 4π は **FOUR_PI**、$\pi/2$ は **HALF_PI**
- ofGetElapsedTimef() はプログラム起動時からの経過時間を返す
- fmod(x, y) は実数 x を実数 y で割った剰余を実数で返す
  - さらに y で割ると 0～1 の値になる

fmod(ofGetElapsedTimef(), cycle)



左上腕を追加する

```
//--------------------------------------------------------------
void ofApp::setup(){
（途中略）

  // 0: 胴
  auto body = new ofBoxPrimitive{ w, h, d };
  parts.emplace_back(body);
  parts.back()->move(0.0f, 2 l_leg + h / 2, 0.0f);

  // 1: 頭
  parts.emplace_back(new ofSpherePrimitive{ r, 20 });
  parts.back()->move(0.0f, h / 2 + r, 0.0f);
  parts.back()->setParent(*parts.front());

  // 2: 左上腕
  auto larm = new ofCylinderPrimitive{ c, l_arm, 12, 1 };
  for (auto &vertex : larm->getMeshPtr()->getVertices()) {
    vertex.y -= l_arm / 2;
  }
  parts.emplace_back(larm);
  parts.back()->setParent(*body);
```

- 腕には ofCylinderPrimitive のオブジェクト（円柱）を使う
  - c や l_{arm} は適当に決めてください
  - ofCylinderPrimitive 等 of3dPrimitive を継承したオブジェクトは原点が図形の中心にある
  - 回転中心の原点が円柱の上面に来るように頂点の y 座標値をずらす
    - この処理の説明は割愛
- parts に追加する
- body を親にする

左前腕を追加する

```
// 0: 胴
auto body = new ofBoxPrimitive{ w, h, d };
parts.emplace_back(body);
parts.back()->move(0.0f, 2 l_leg + h / 2, 0.0f);

// 1: 頭
parts.emplace_back(new ofSpherePrimitive{ r, 20 });
parts.back()->move(0.0f, h / 2 + r, 0.0f);
parts.back()->setParent(*parts.front());

// 2: 左上腕
auto larm = new ofCylinderPrimitive{ c, l_arm, 12, 1 };
for (auto &vertex : larm->getMeshPtr()->getVertices()) {
  vertex.y -= l_arm / 2;
}
parts.emplace_back(larm);
parts.back()->setParent(*body);

// 3: 左前腕
parts.emplace_back(new ofCylinderPrimitive{ *larm });
parts.back()->setParent(*larm);
```

データをコピーした
オブジェクトが生成される

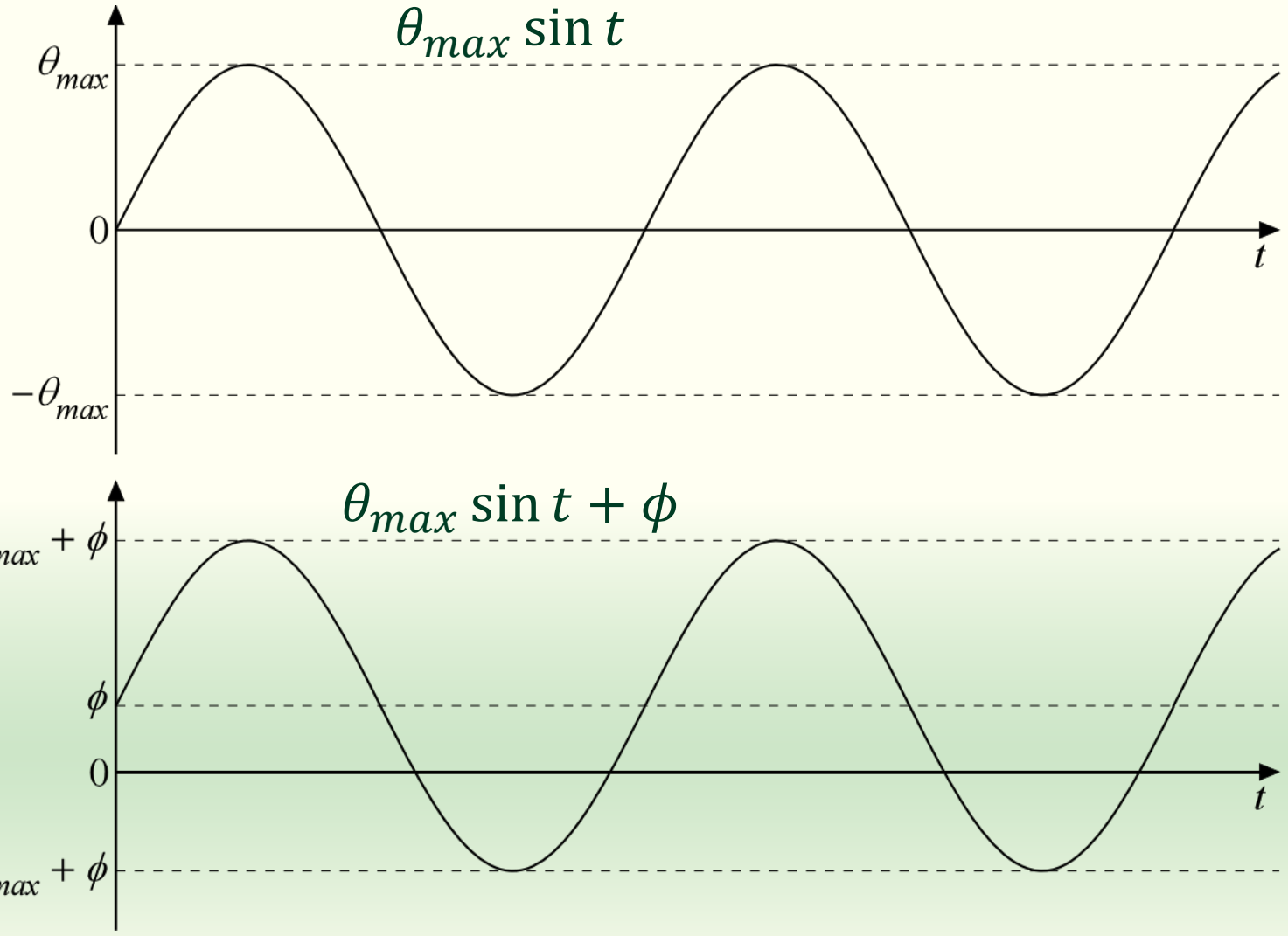
- 左前腕は左上腕をコピーする
  - 左上腕を使って左前腕のオブジェクトを生成する
- 左前腕の親は左上腕にする

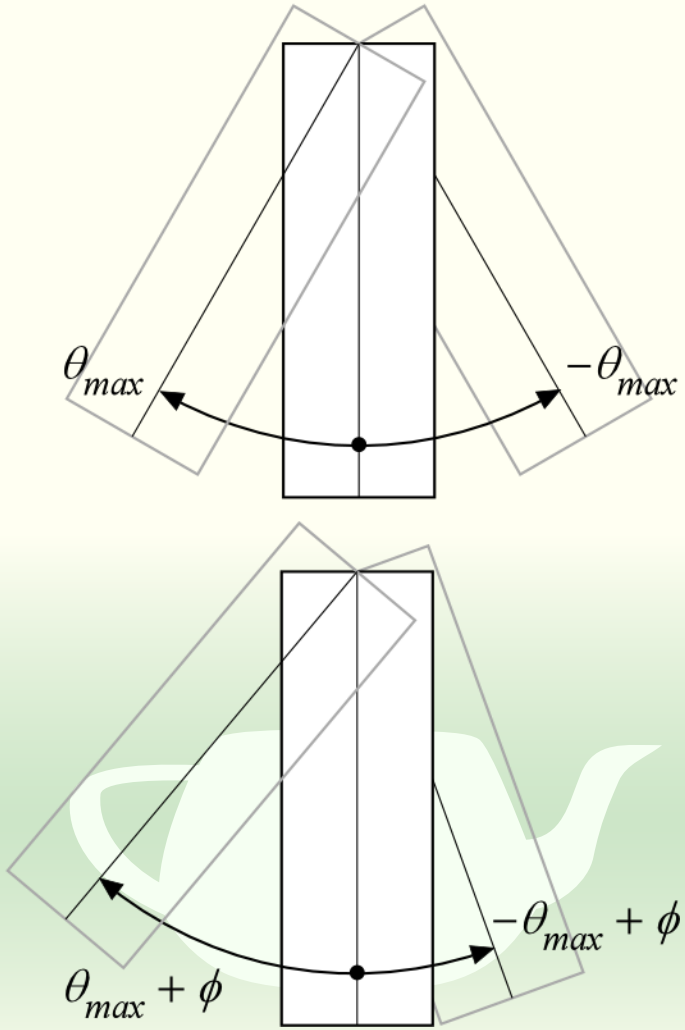
左上腕を配置する

```
const double cycle = 2.0;
const vec3 xAxis{ 1.0f, 0.0f, 0.0f };

 (途中略)

//--------------------------------------------
void ofApp::update(){
  const float t = 2.0 * M_PI
    * fmod(ofGetElapsedTimef(), cycle) / cycle;

  const float t2 = θ2max * sin(t);
  parts[2]->resetTransform();
  parts[2]->move(w / 2 + c, h / 2, 0.0f);
  parts[2]->rotateDeg(t2, xAxis);
```

- x 軸方向のベクトルを多用するので定数 xAxis として用意する
- move() メソッドは現在位置からの平行移動なので update() のたびに resetTransform() で元に戻す
- const float t2 = 30.0f * sin(t);
  - 腕を振る動作を再現するために腕の角度を三角関数で変化させる
  - 最適な θ_{2max} を決めてください
  - $w / 2 + c$ や $h / 2$ は自分で計算してください

腕の振れ角を三角関数で変化させる



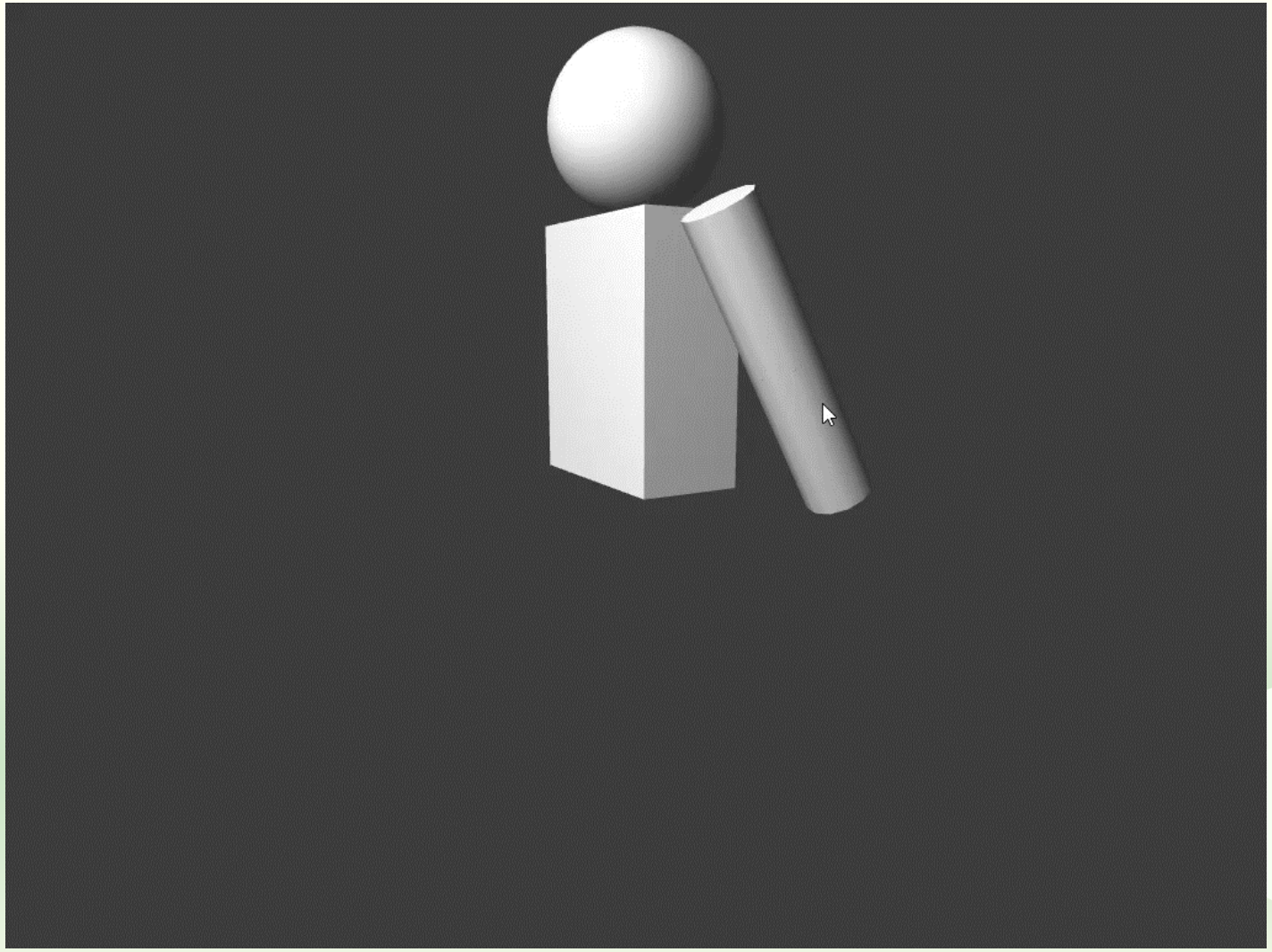


左前腕を配置する

```
//--------------------------------------------
void ofApp::update(){
  const float t = 2.0 * M_PI
    * fmod(ofGetElapsedTimef(), cycle) / cycle;

  const float t2 = θ_max * sin(t);
  parts[2]->resetTransform();
  parts[2]->move(w / 2 + c, w / 2, 0.0f);
  parts[2]->rotateDeg(t2, xAxis);

  const float t3 = θ_3max * sin(t) - φ_3;
  parts[3]->resetTransform();
  parts[3]->move(0.0f, -l_arm, 0.0f);
  parts[3]->rotateDeg(t3, xAxis);
```

- 左前腕は左上腕の振りに追従することに加えて肘の屈伸による振りを追加する
  - θ_{3max} や ϕ_3 を調整して自然な腕の振りを再現してください

課題３－１実行例



課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **3-1.png** というファイル名で保存し、Moodle の第 3 回課題にアップロードしてください

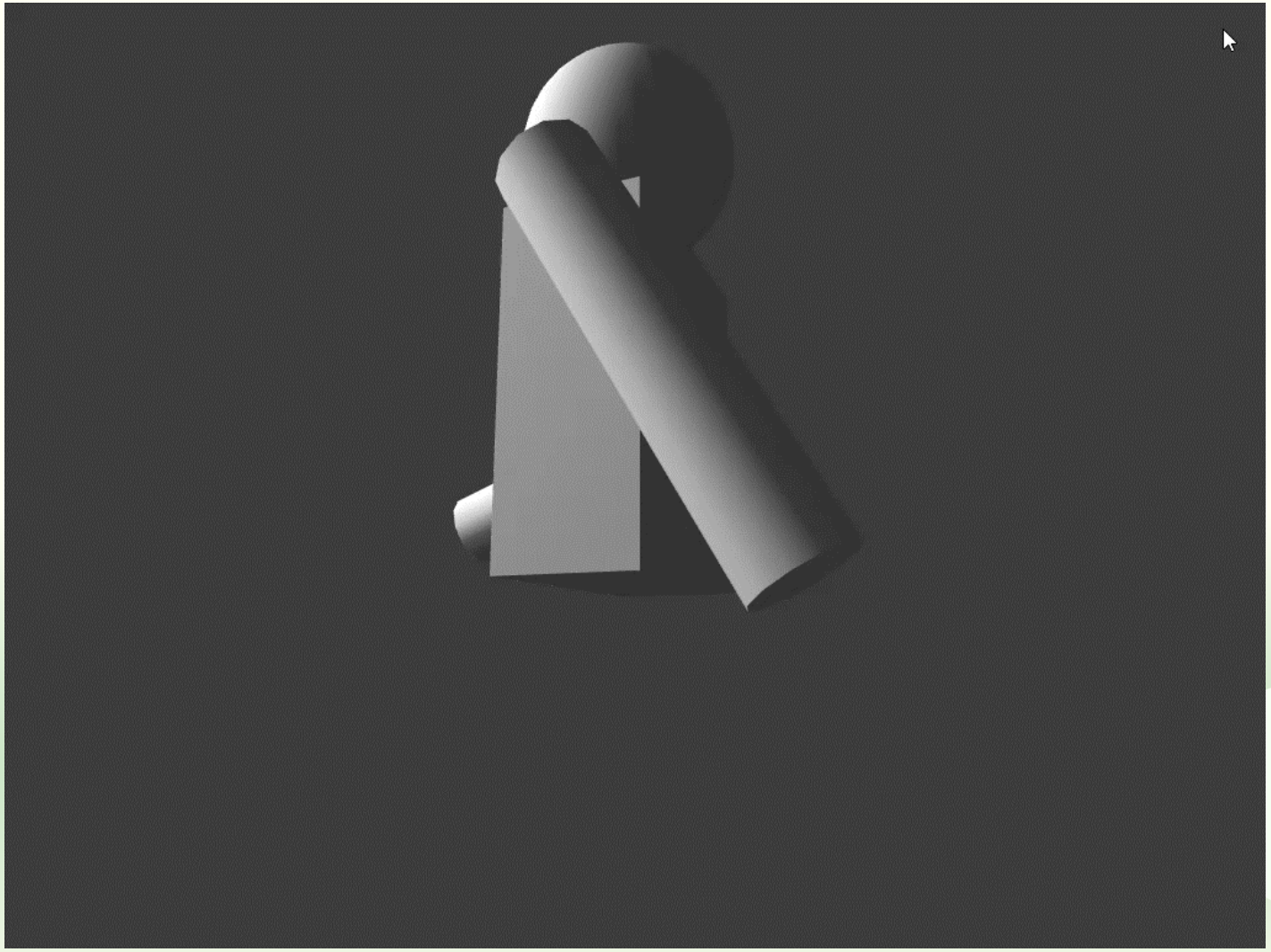
課題３－２

右腕を振る

右腕を振る

- 右腕の上腕と前腕を追加して左腕と反対のタイミングで振るようにしてください
- 右腕の上腕と前腕は左腕と同じ形なので右腕のオブジェクトをコピーして使うとよいでしょう
- このとき右腕の上腕の振りの周期は左腕の上腕と逆になるようにしてください
  - 肘が逆方向に折れ曲がらないように前腕の角度を設定する必要があります

課題３－２実行例



課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **3-2.png** というファイル名で保存し、Moodle の第 3 回課題にアップロードしてください

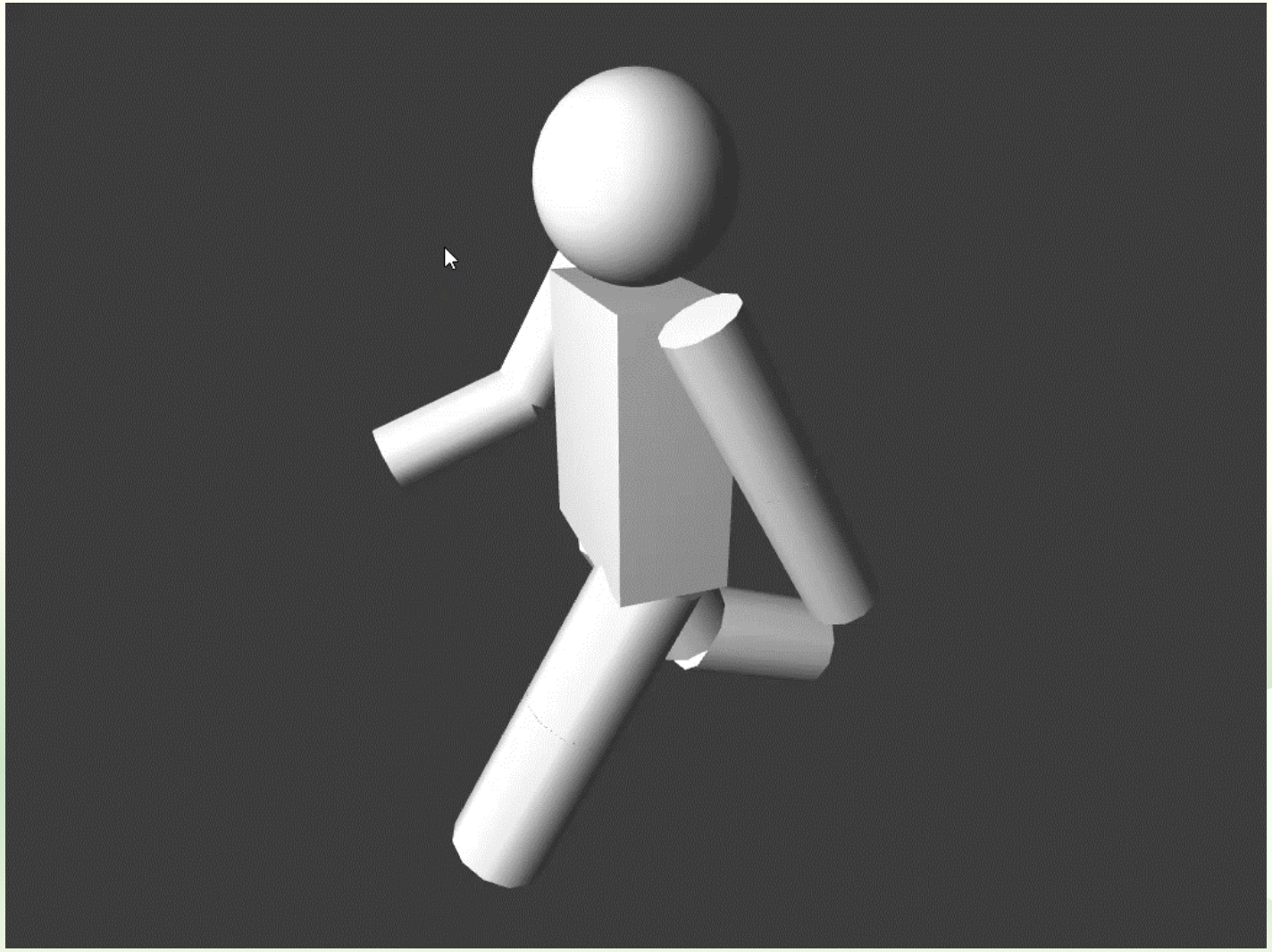
課題３－３

両足を振る

両足を振る

- 右足の腿と脛、左足の腿と脛を追加して、それぞれ右腕、左上と反対のタイミングで振るようにしてください
- 腿と脛は同じ形ですが腕とは形が違うので、腿のオブジェクトを新規に作成して、残りの腿と脛にコピーして使うとよいでしょう
- このとき右足の腿の振りの周期は左足の腿と逆になるようにしてください
  - 膝が逆方向に折れ曲がらないように前腕の角度を設定する必要があります

課題３－３実行例



課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **3-3.png** というファイル名で保存し、Moodle の第 3 回課題にアップロードしてください

課題３－４

円上を歩くようにする

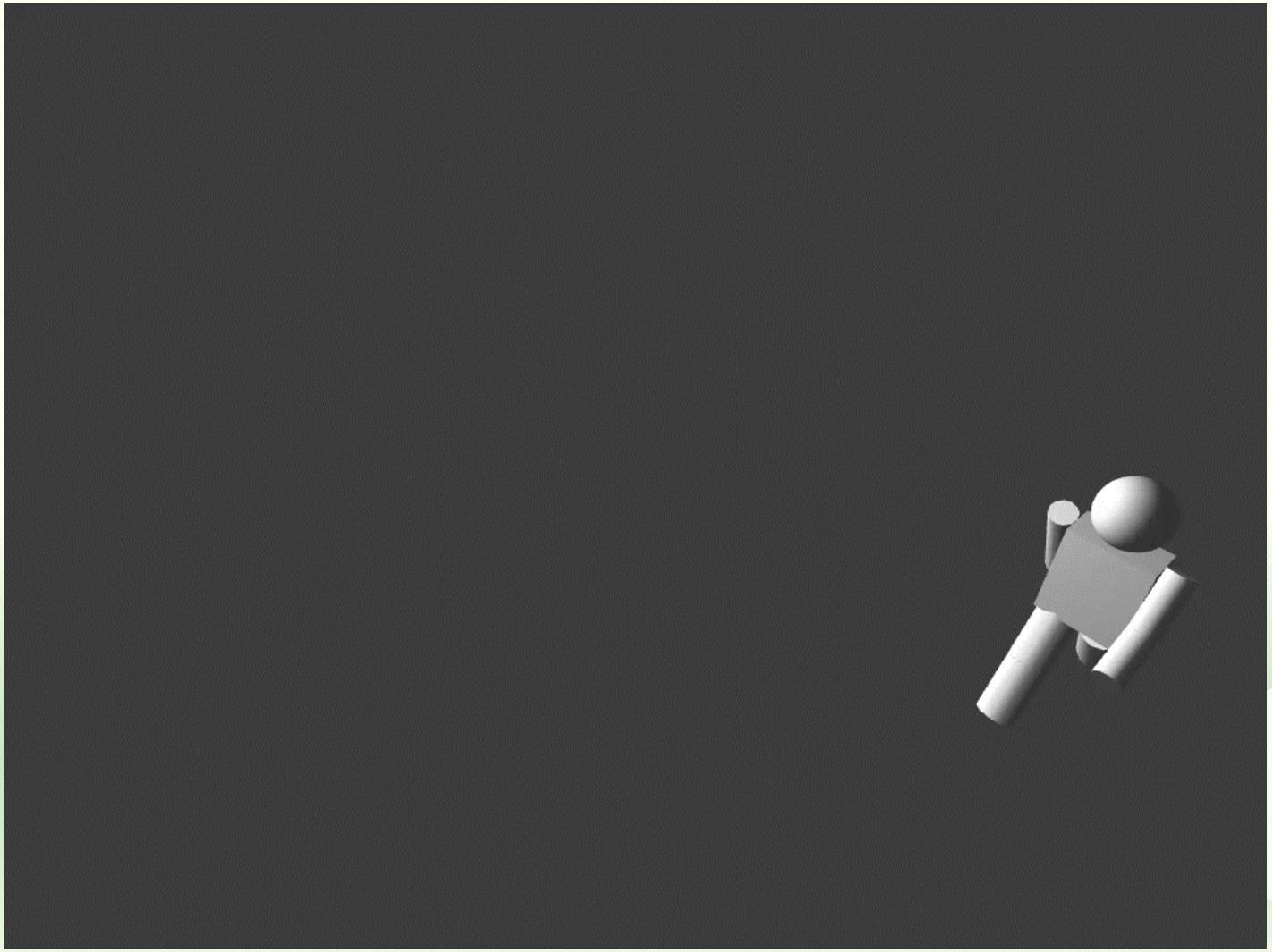
円上を歩く

- 中心を設定して胴のオブジェクトをｙ軸周りに回転させれば円上を歩くようになる

課題のアップロード

- 作成したプログラムの実行中のウィンドウを**５秒以内**で動画キャプチャして、**3-4.mp4** というファイル名で Moodle の第３回課題にアップロードしてください
  - 実行例と同じだと評価できないので形や動きを変えてください
  - 動画のキャプチャができないときはスクリーンショットを撮って 3-4.png というファイル名でアップロードしてください
- ソースプログラム **ofApp.h** と **ofApp.cpp** を Moodle の第３回課題にアップロードしてください

課題３－４実行例



補足

カメラの向き、画角、縦横比、前方面、後方面を設定する

lookAt() メソッド

- void ofNode::lookAt(const glm::vec3 &target)
  - カメラの向きを target の位置に向ける
- void ofNode::lookAt(const glm::vec3 & target, glm::vec3 up)
  - 上方向を up にしてカメラの向きを target の位置に向ける
- void ofNode::lookAt(const ofNode &node)
  - カメラの向きを他の node の位置に向ける
- void ofNode::lookAt(const ofNode &node, const glm::vec3 &up)
  - 上方向を up にしてカメラの向きを他の node の位置に向ける

カメラを移動しても常に１点を向くようにする

```
//--------------------------------------------
void ofApp::update(){
  const vec3 up{ 0.0f, 1.0f, 0.0f };
  const vec3 center{ 0.0f, 0.0f, 0.0f };

  camera.rotateAroundDeg(1.0f, up, center);
  camera.lookAt(center, up);

   (途中略)
}
```

- rotateAroundDeg() メソッドでカメラの位置を回転してもカメラの方向は変わらない
  - rotateAroundDeg() で camera の位置を回転するときカメラは center からずれた位置にないといけない
- lookAt() メソッドでカメラが常に原点を向くようにしている
  - この場合 up を指定しなくても y 方向が上になる

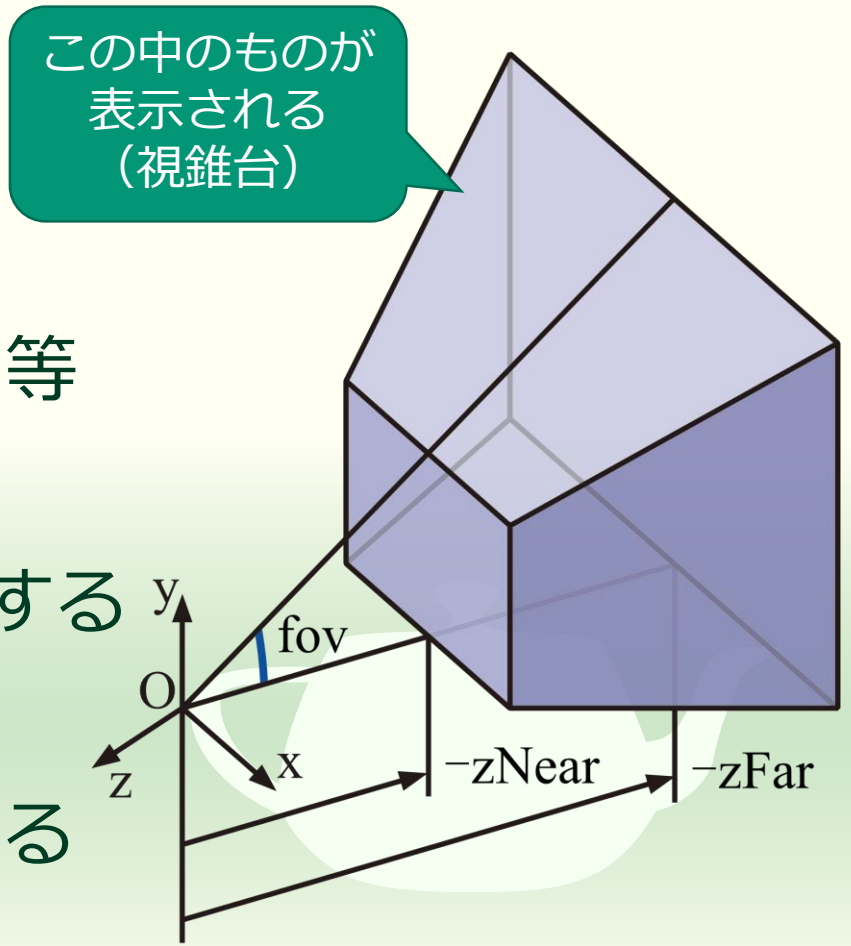
カメラが他の図形を追いかけるようにする

```
//--------------------------------------------
void ofApp::update(){
  const vec3 up{ 0.0f, 1.0f, 0.0f };
  const vec3 center{ 0.0f, 0.0f, 0.0f };

  parts[0]->rotateAroundDeg(-0.3f, up, center);
  parts[0]->rotate(-0.3f, up);

  camera.lookAt(*parts[0]);

   (途中略)
}
```

- lookAt() の目標にはほかの物体 (ofNode) が指定できる
- この camera は常に parts[0] の方向を向く
- lookAt() は ofNode クラスのメソッドなので ofCamera 以外の of3dPrimitive などの向きも制御できる

カメラの画角、縦横比、前方面、後方面

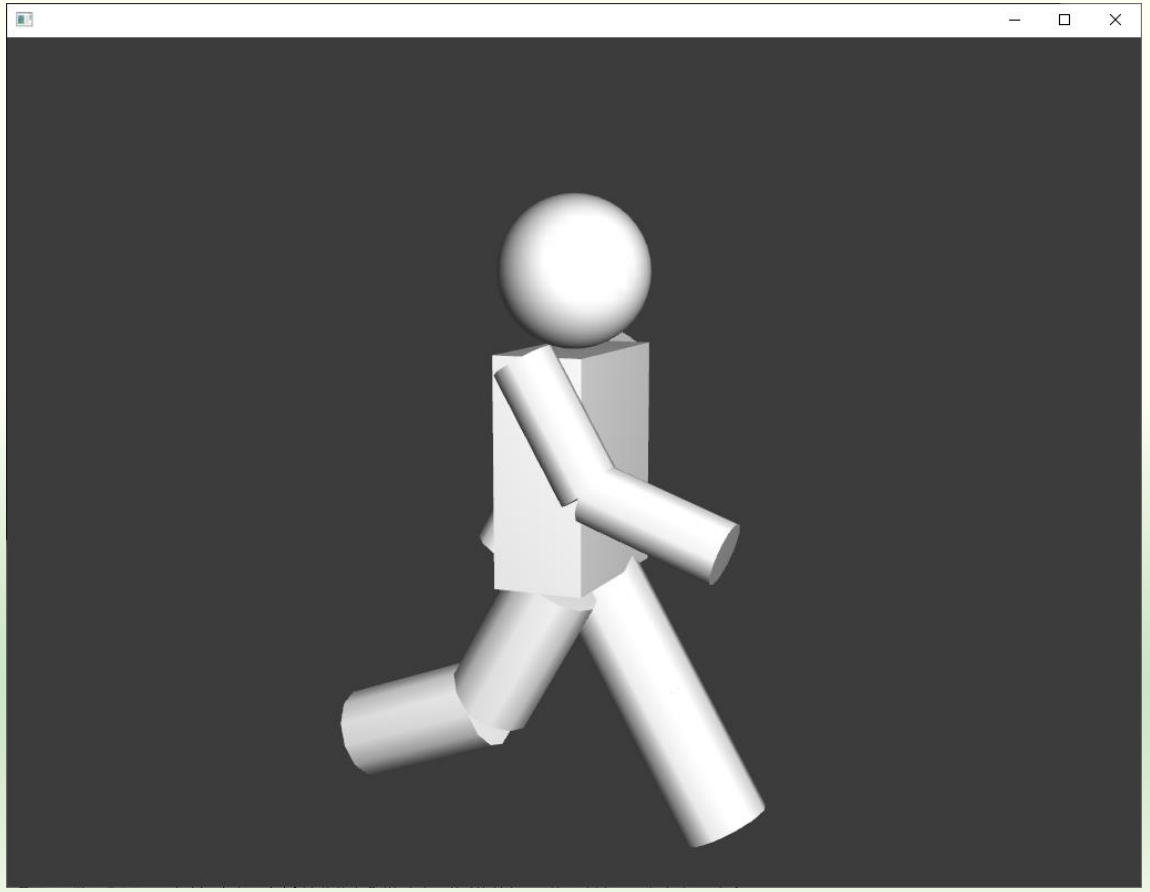
- void ofCamera::setFov(float fov)
  - fov にカメラの画角を度で指定する
- void ofCamera::setAspectRatio(float a)
  - a にウィンドウの縦横比を指定する
  - a は float(ofGetWidth()) / float(ofGetHeight()) 等
- void ofCamera::setNearClip(float zNear)
  - zNear に前方面のカメラからの距離を指定する
- void ofCamera::setFarClip(float zFar)
  - zFar に後方面のカメラからの距離を指定する



画角 fov が小さいほど望遠になる

camera.setFov(60.0f);

- **camera.setFov(30.0f);**

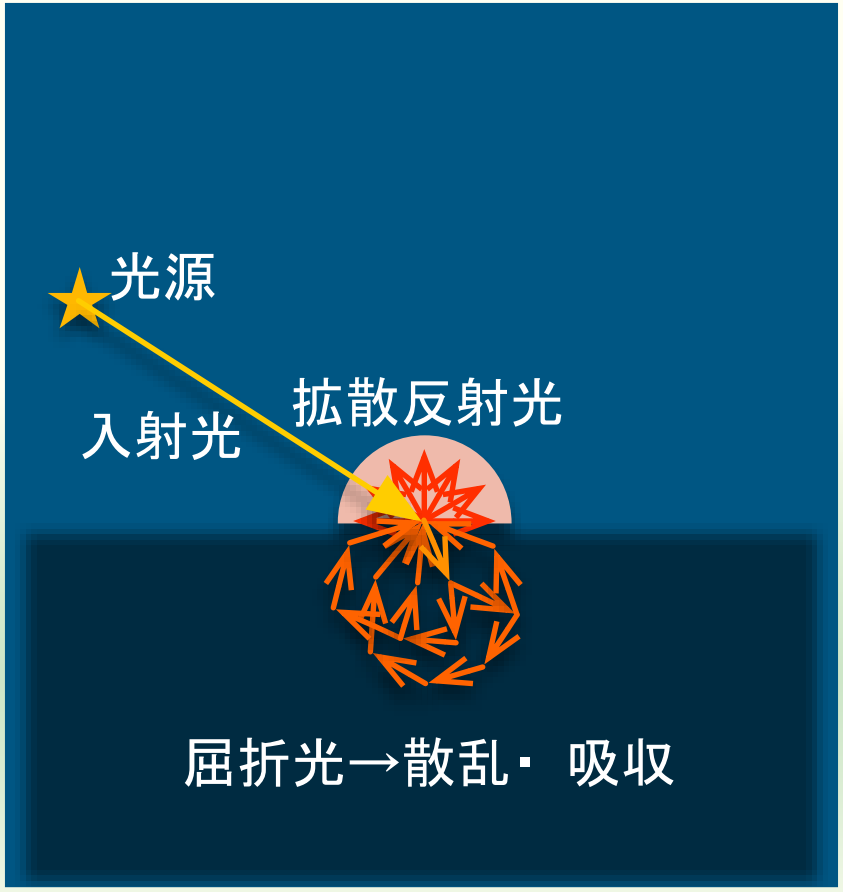


補足

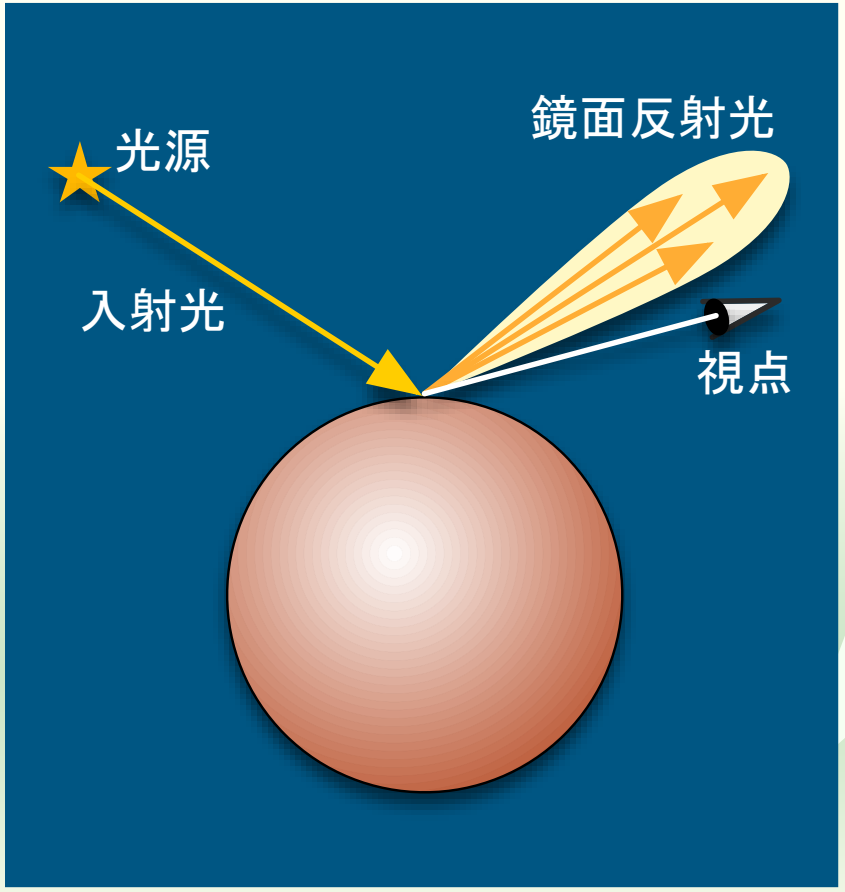
材質

二色性陰影付けモデル

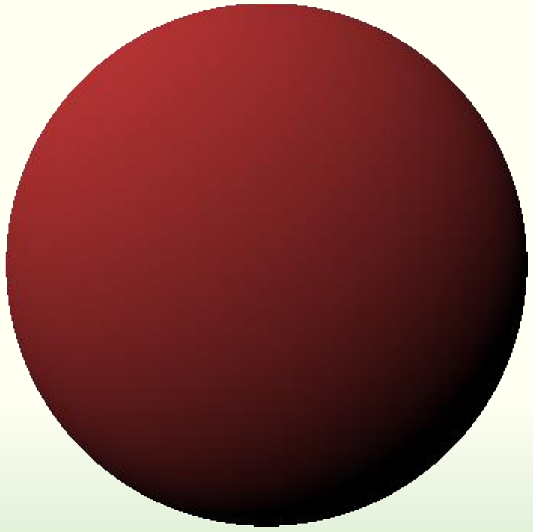
拡散反射光 (diffuse)



鏡面反射光 (specular)

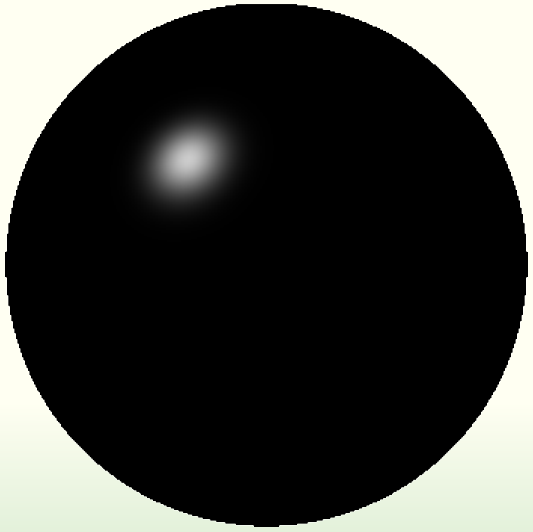


陰影付け方程式



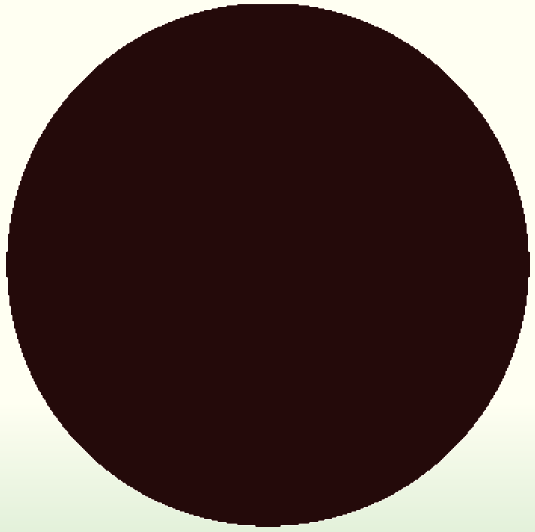
拡散反射光

I_{diff}



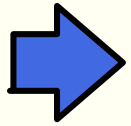
鏡面反射光

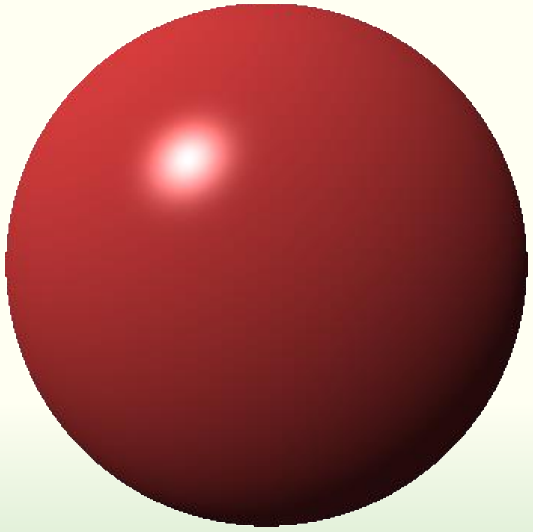
I_{spec}



環境光の反射光

I_{amb}





反射光強度

$I_{tot} = I_{diff} + I_{spec} + I_{amb}$

材質の設定と描画

ofApp.h 等

```
// 材質
ofMaterial material;
```

```
// 拡散反射係数と鏡面反射係数
const ofFloatColor diffuse{ 0.8f, 0.2f, 0.4f };
const ofFloatColor specular{ 0.3f, 0.3f, 0.3f };

// 輝き係数
const float shininess = 30.0f;

// 材質の設定
material.setAmbientColor(diffuse);
material.setDiffuseColor(diffuse);
material.setSpecularColor(specular);
material.setShininess(shininess);
```

エネルギー保存の法則に従うなら上下を足して 1 を超えたらまずい

diffuse と ambient の色は同じにするのが基本

setup() 等

```
// 描画
material.begin();
part->draw();
material.end();
```

draw() 等

- ofMaterial クラスのオブジェクトを生成する
- 拡散反射係数、鏡面反射係数、および輝き係数を決定する
  - 輝き係数が大きいほどハイライトは小さくなる
  - 非金属の材質では鏡面反射係数はグレーにしておくのが基本
  - 金属・半導体では鏡面反射光にも色が付く場合がある
- begin()～end() の間で描画する

補足

std::unique_ptr と std::shared_ptr

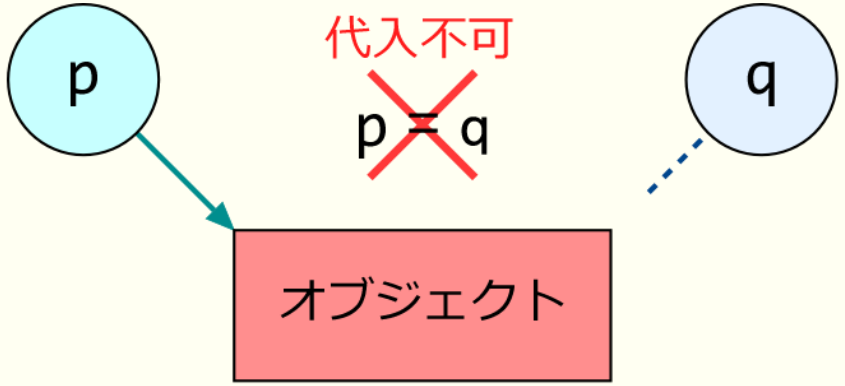
unique_ptr を shared_ptr に替えても動く

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{
  ofEasyCam camera;
  ofLight light;
  vector<shared_ptr<of3dPrimitive>> parts;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
   (以下略)
```

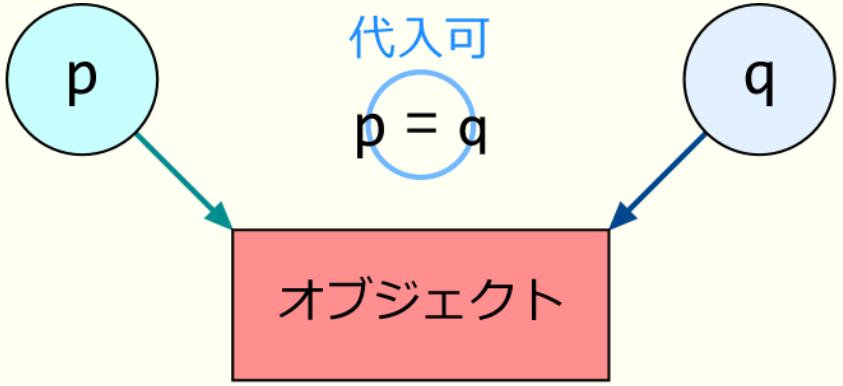
- unique_ptr は複数の変数で同じポインタを保持できない
  - その分性能は良い
- **shared_ptr** は単一のオブジェクトのポインタを複数の変数で保持できるスマートポインタ
  - オブジェクトはすべての変数が削除されたときに削除される
  - unique_ptr のポインタは複数の変数で保持不可

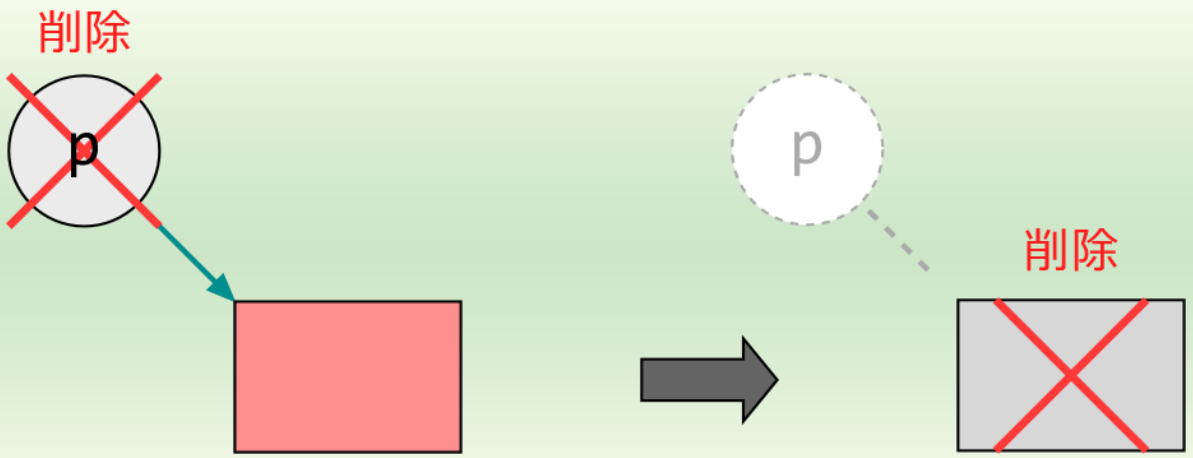
unique_ptr と shared_ptr

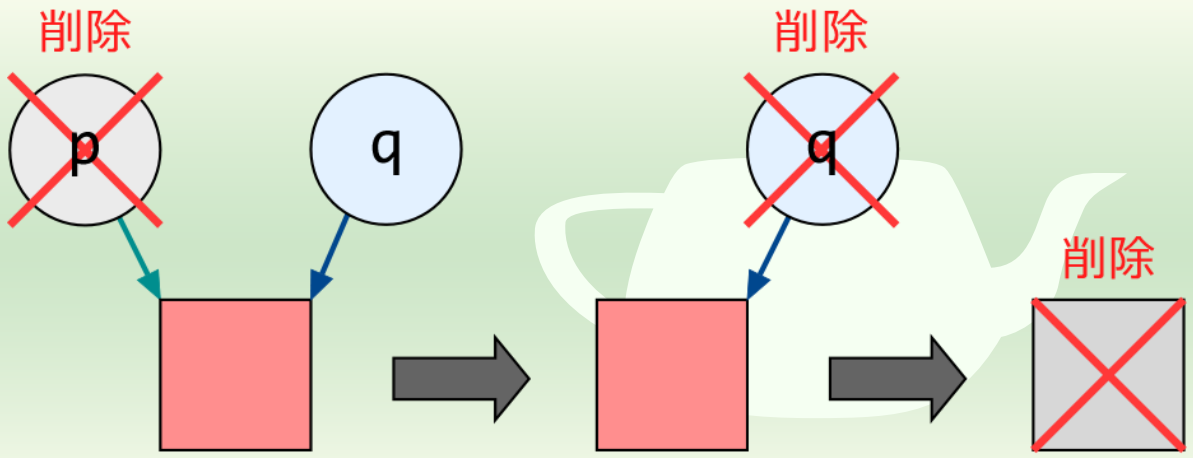
unique_ptr



shared_ptr





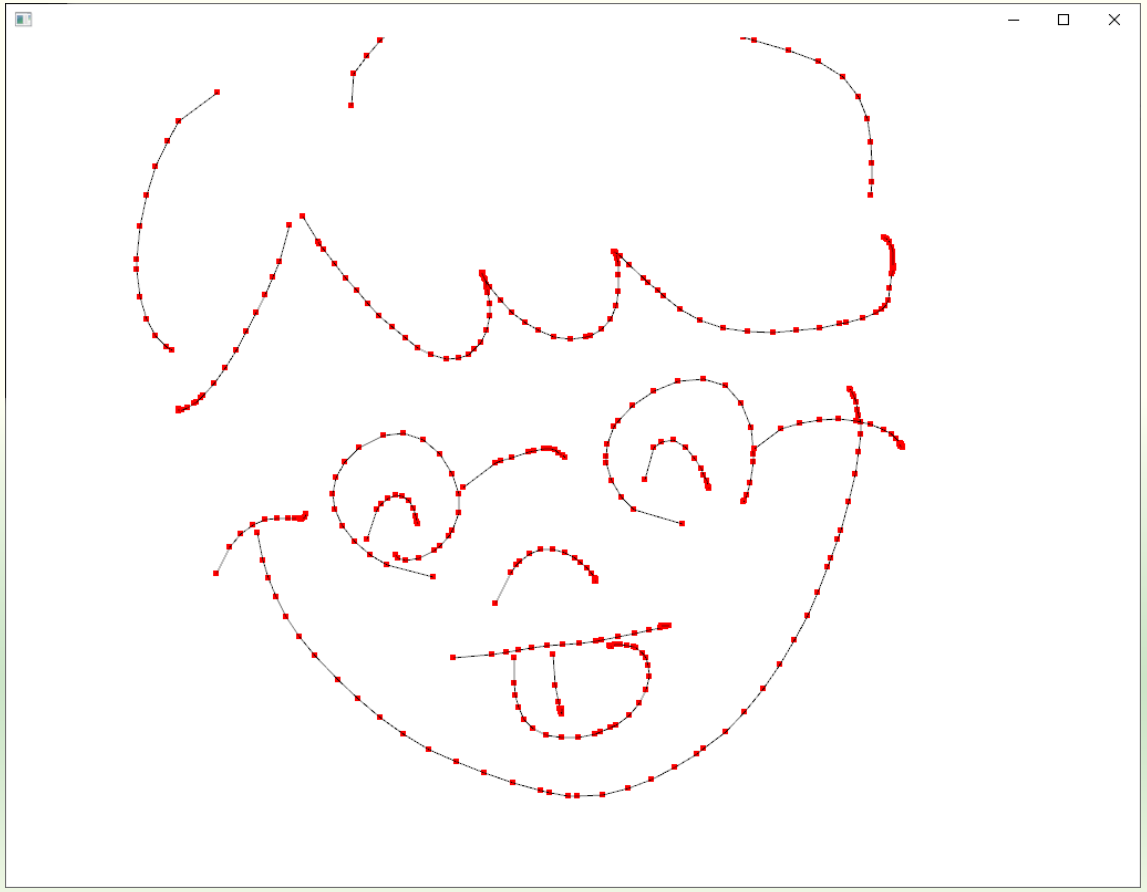


メディアプログラミング演習

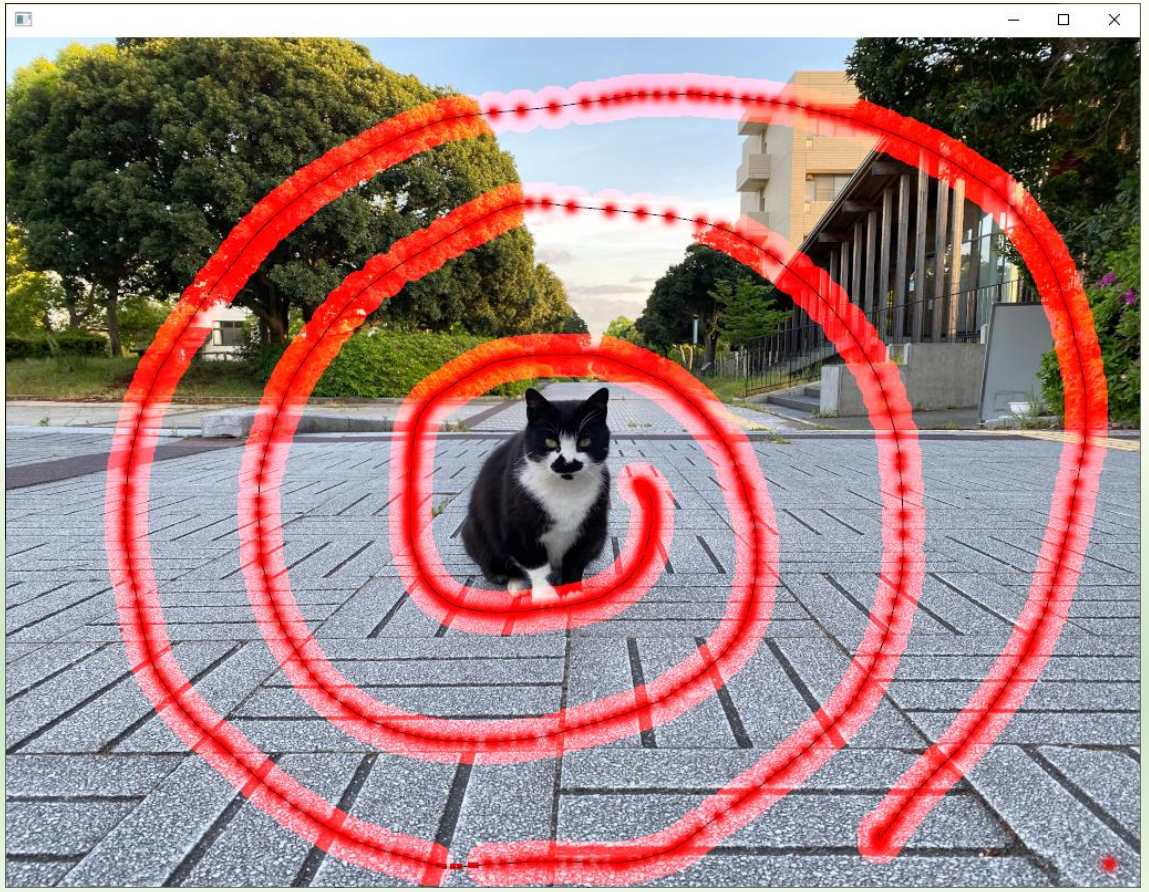
第4回

本日は作図・作画っぽいアプリの作成

ドロー風



ペイント風

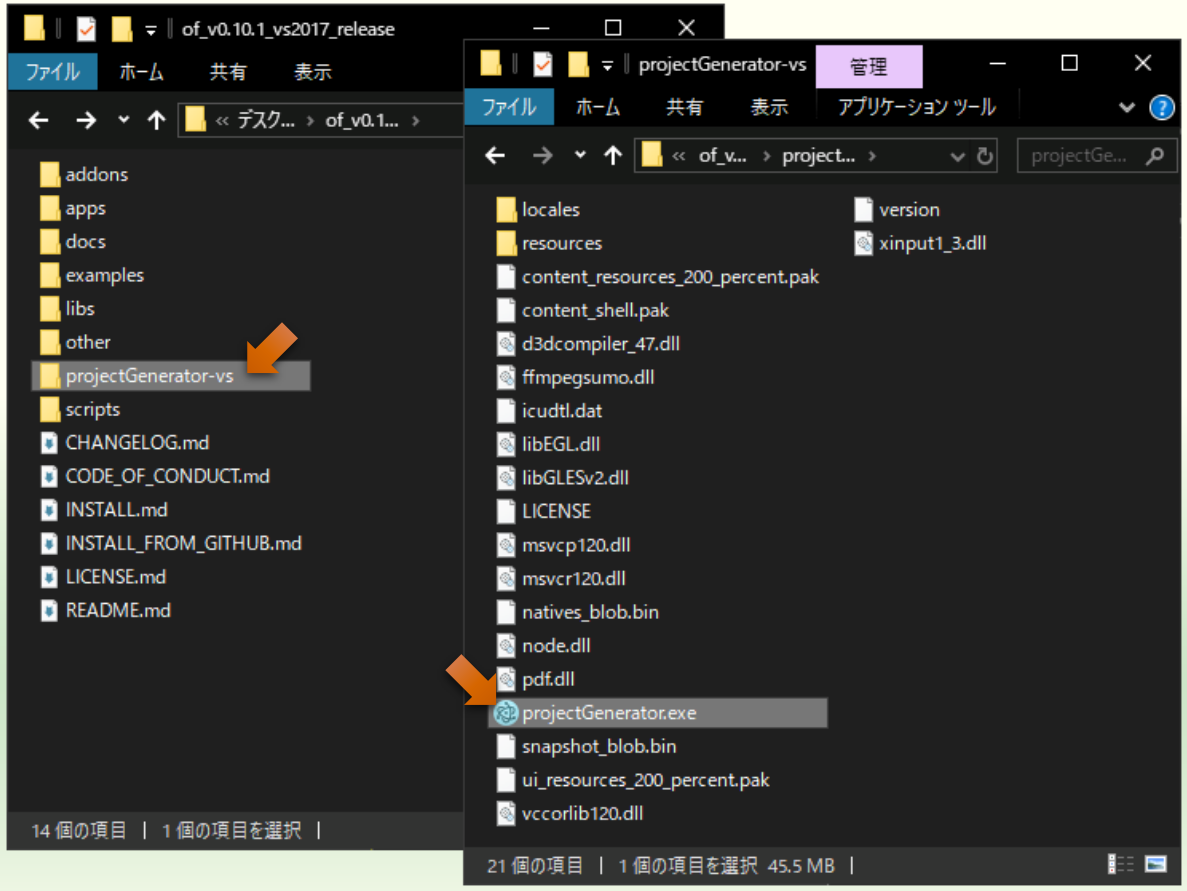


準備

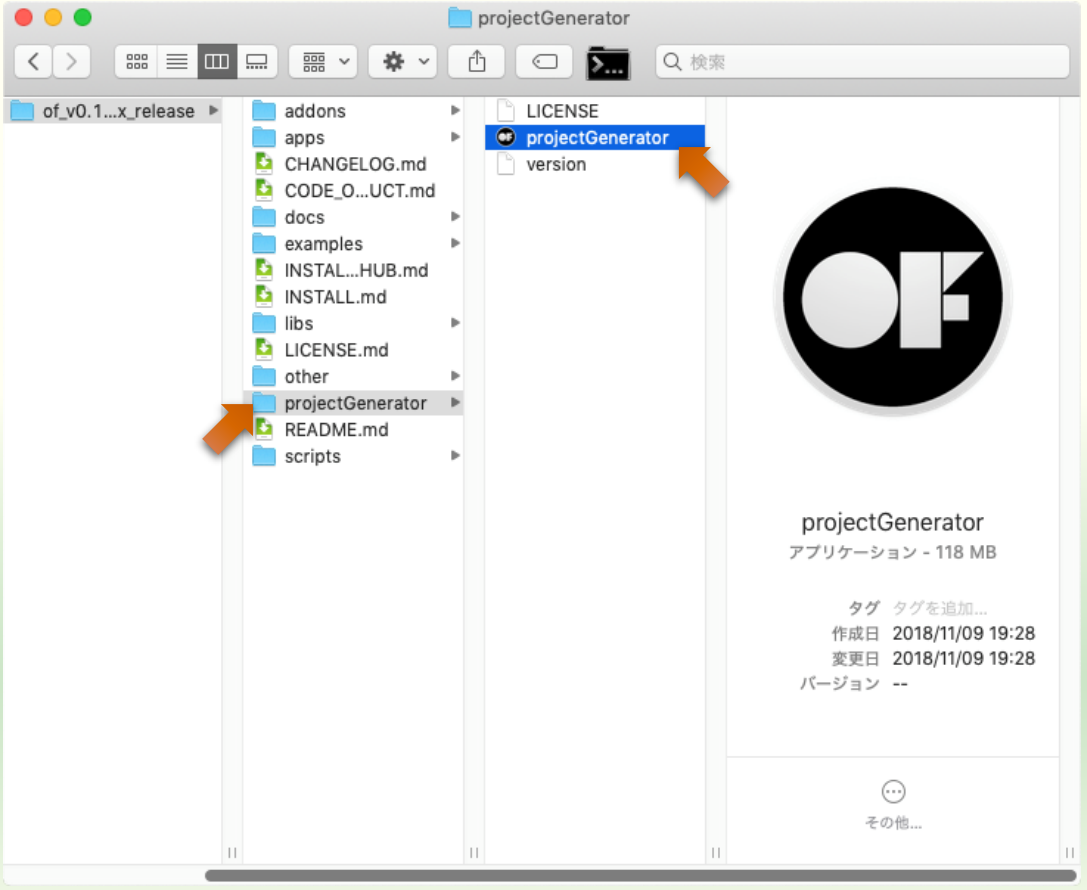
プロジェクトの作成

projectGenerator を起動する

windows 版のパッケージ



macOS 版のパッケージ

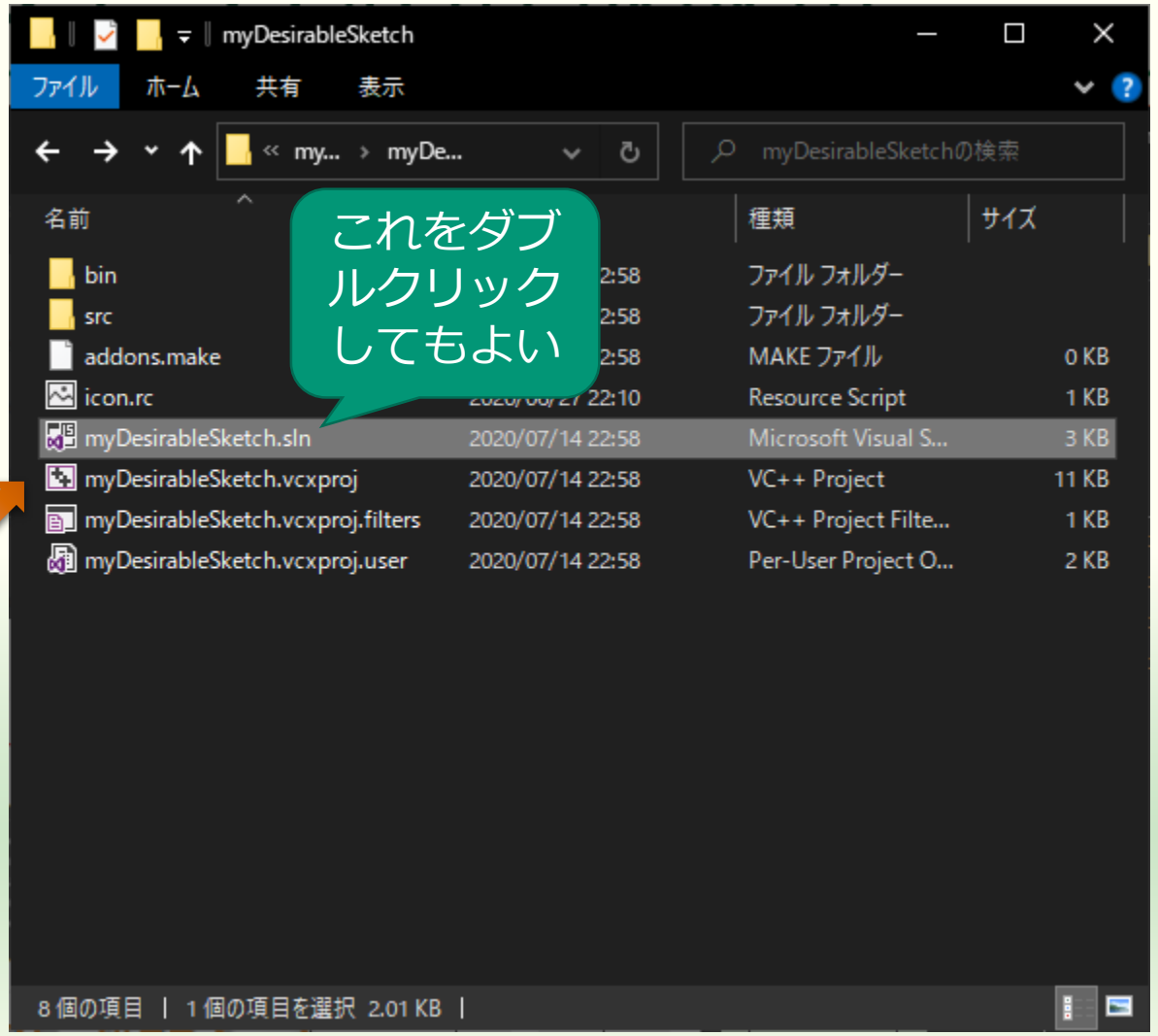


空のプロジェクトの作成

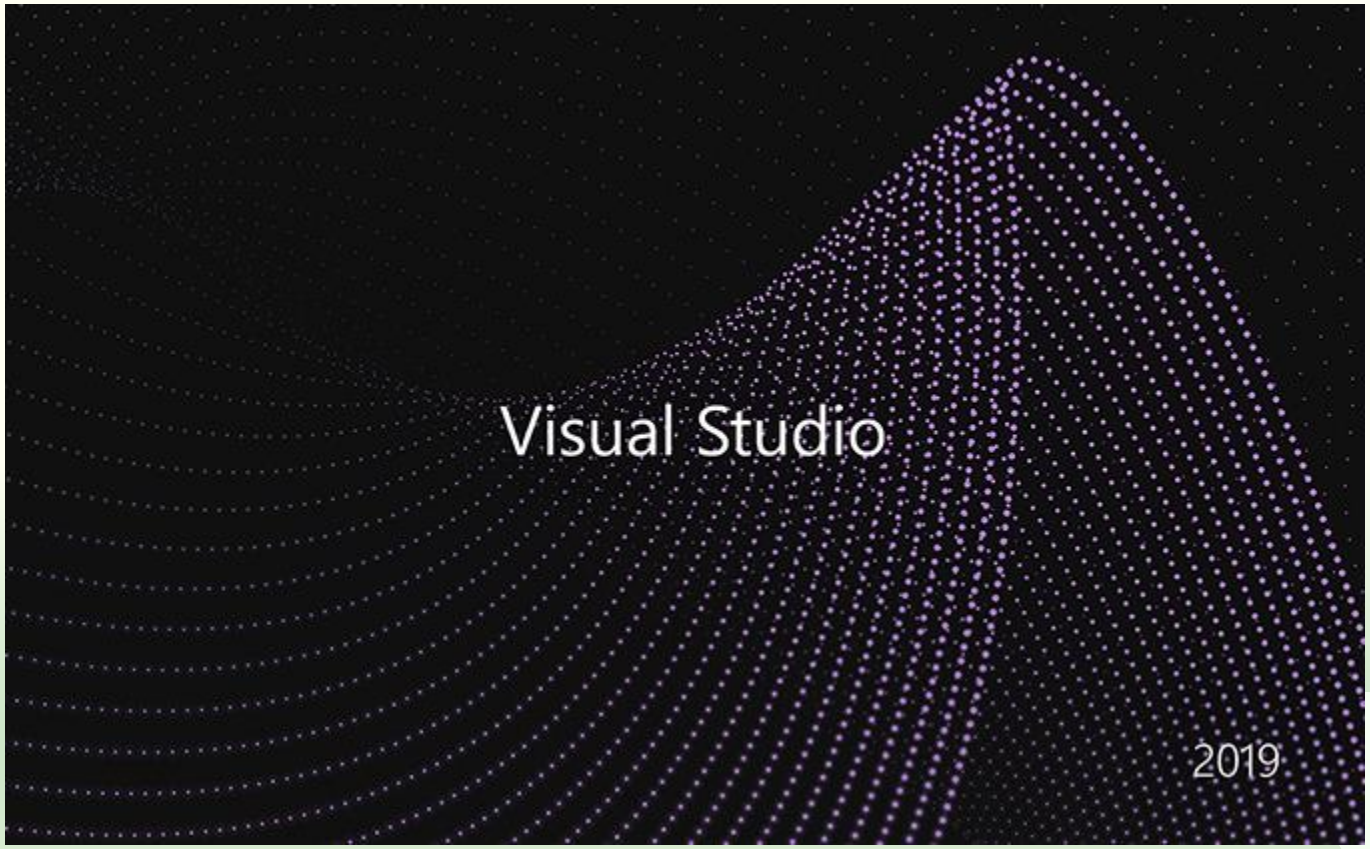
- Project name:
  - 作成するプロジェクト（プログラム）の名前
- Project path:
  - 作成するプロジェクトのファイルを置く場所
  - openFrameworks のパッケージを展開した場所の中の apps¥myApps

プロジェクトの作成成功

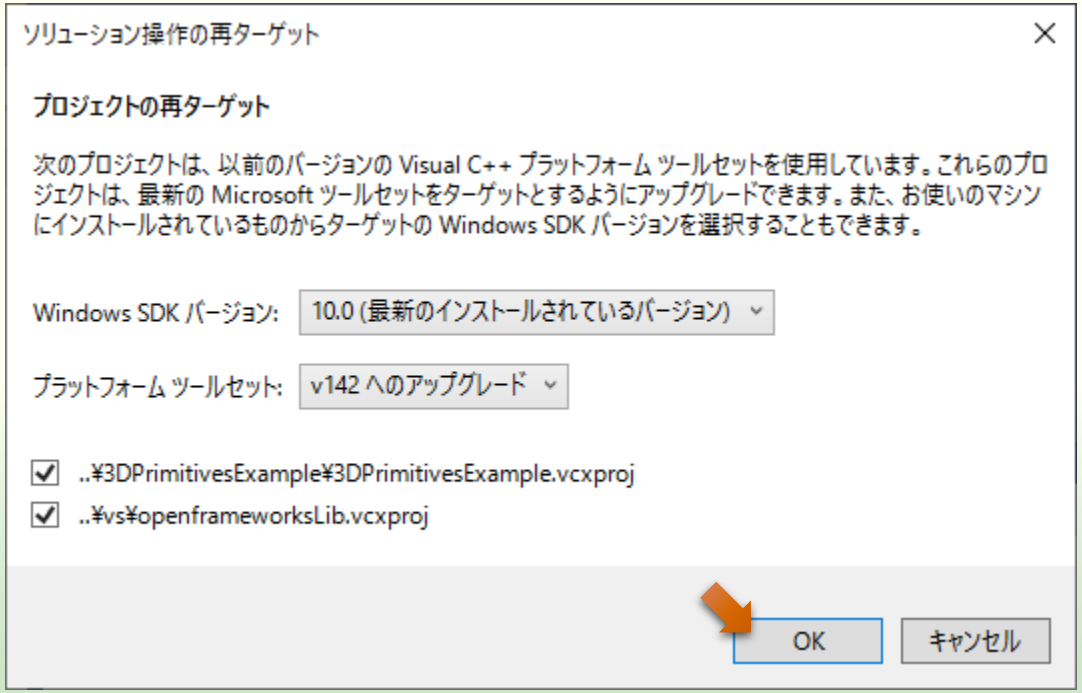
クリックすると開く



Visual Studio 2019 が起動する

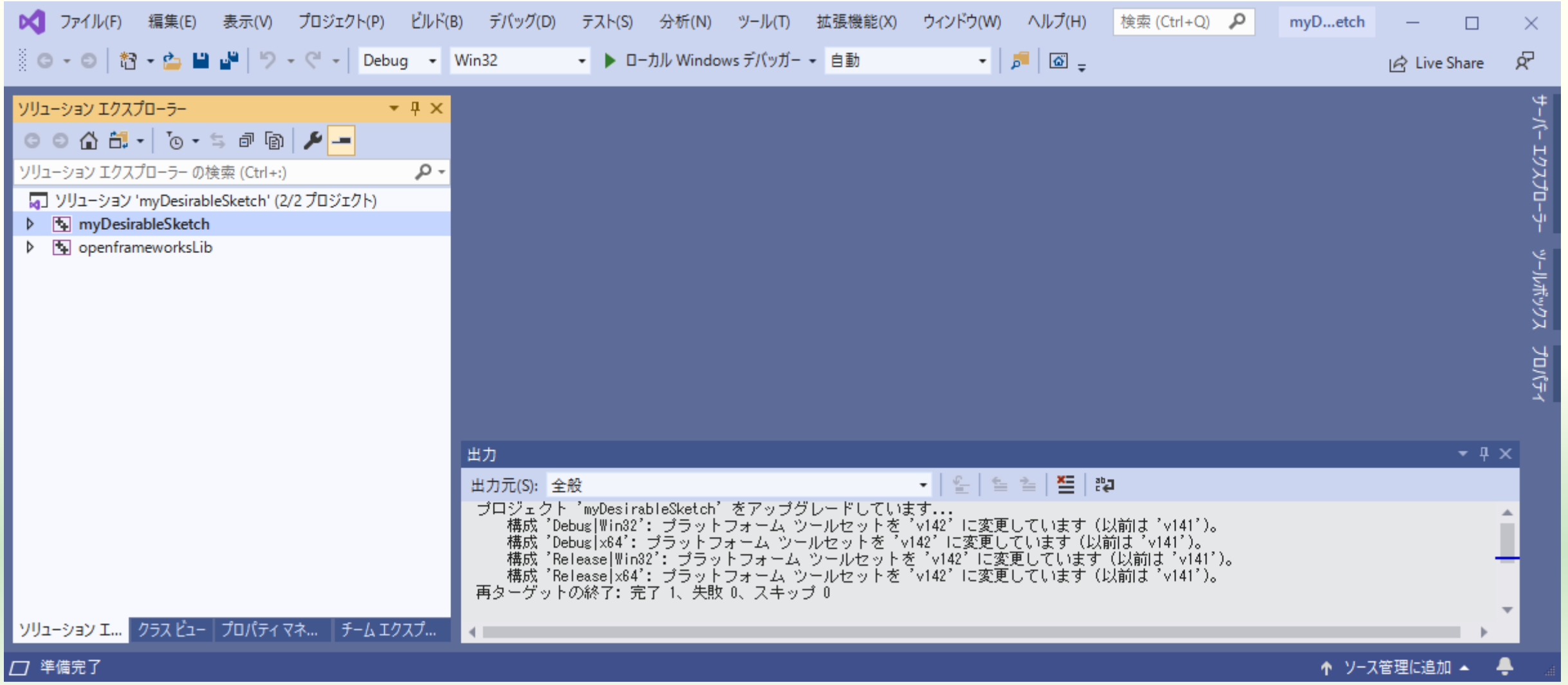


ソリューションの再ターゲット



Visual Studio は頻繁に更新しているので皆さんがお使いの Visual Studio SDK のバージョンと合わない場合がある

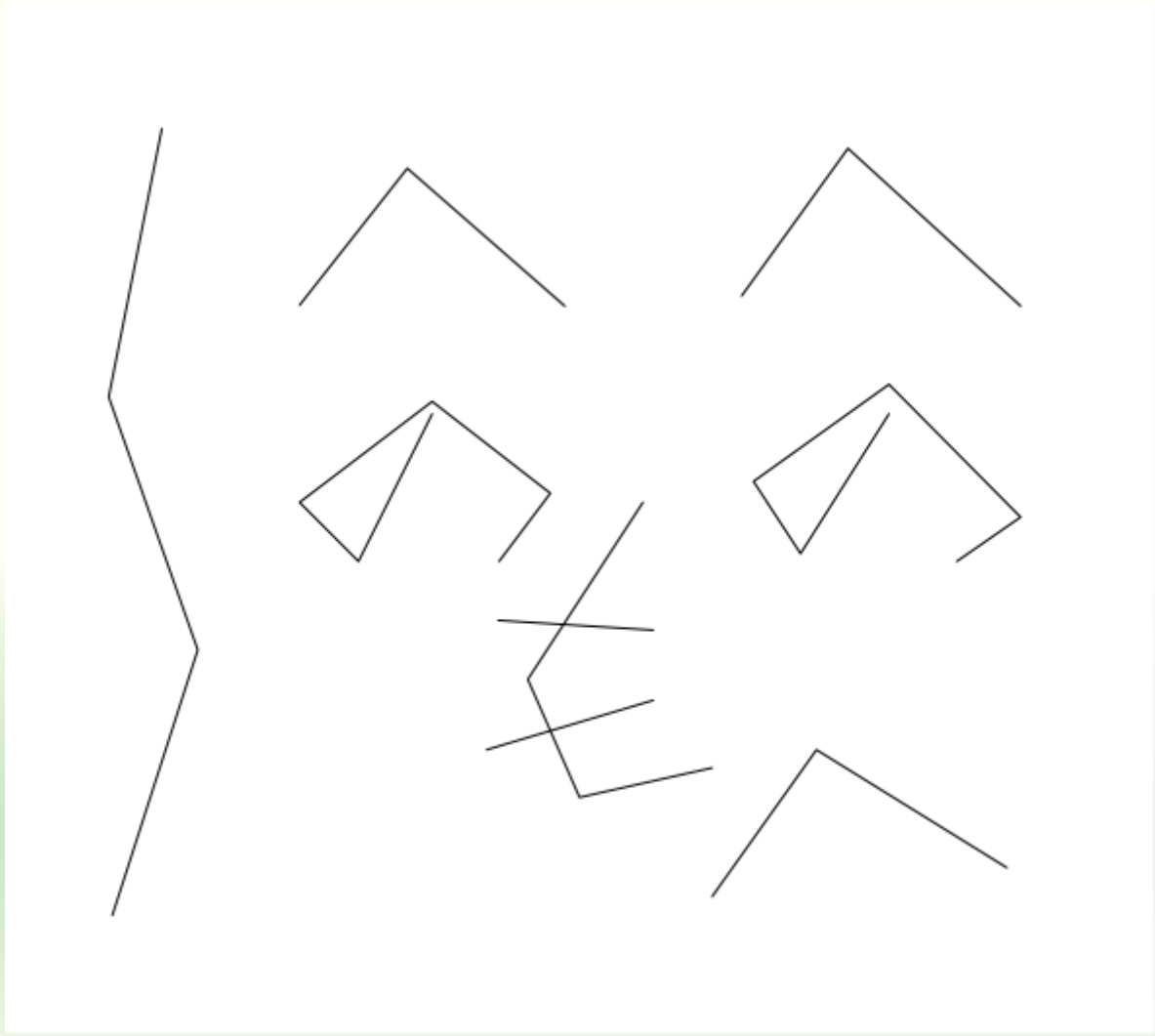
Visual Studio 起動



マウスによる線の描画

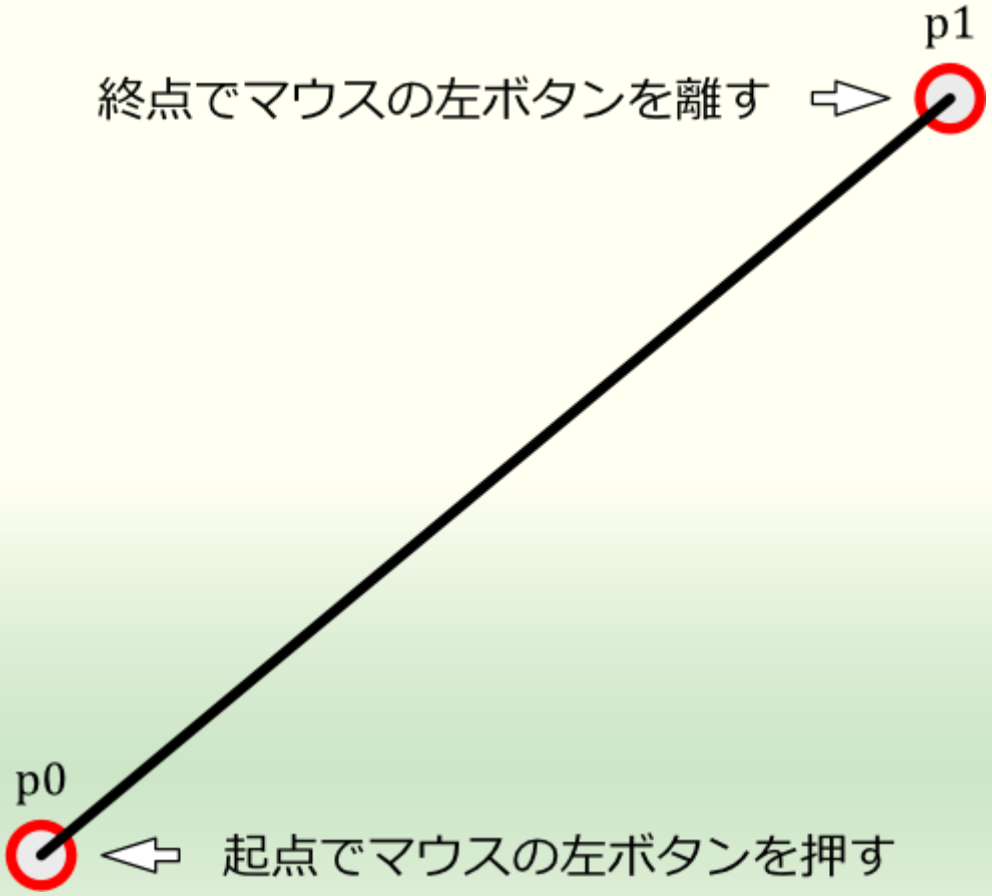
ドローツールっぽいもの

ウィンドウ上にマウスで折れ線を描く



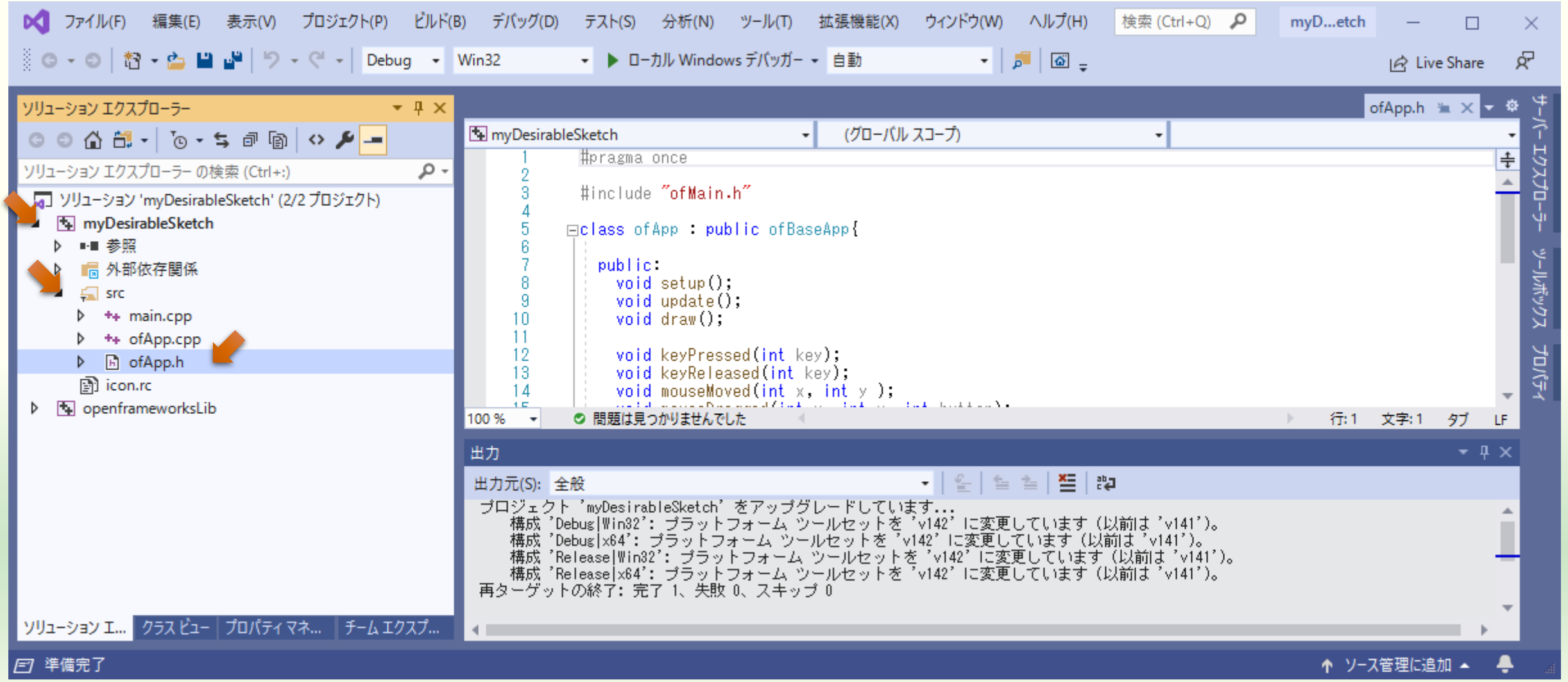
- 仕様
  - マウスの左ボタンをクリックしたところを折れ線で結ぶ
  - マウスの右ボタンをクリックしたところで折れ線は終わる
  - 次に左ボタンをクリックしたら新しい折れ線を描き始める
  - 背景色は白
  - 折れ線の色は黒
- 話の都合上 ofPath や ofPolyline などは知ってても使わない

最初に 2 点間に線分を引くことを考える



- 起点でマウスの左ボタンを押す
  - 押した位置を p0 とする
- 終点でマウスの右ボタンを離す
  - 離した位置を p1 とする
- p0 から p1 に線分を描く

ofApp.h を開く

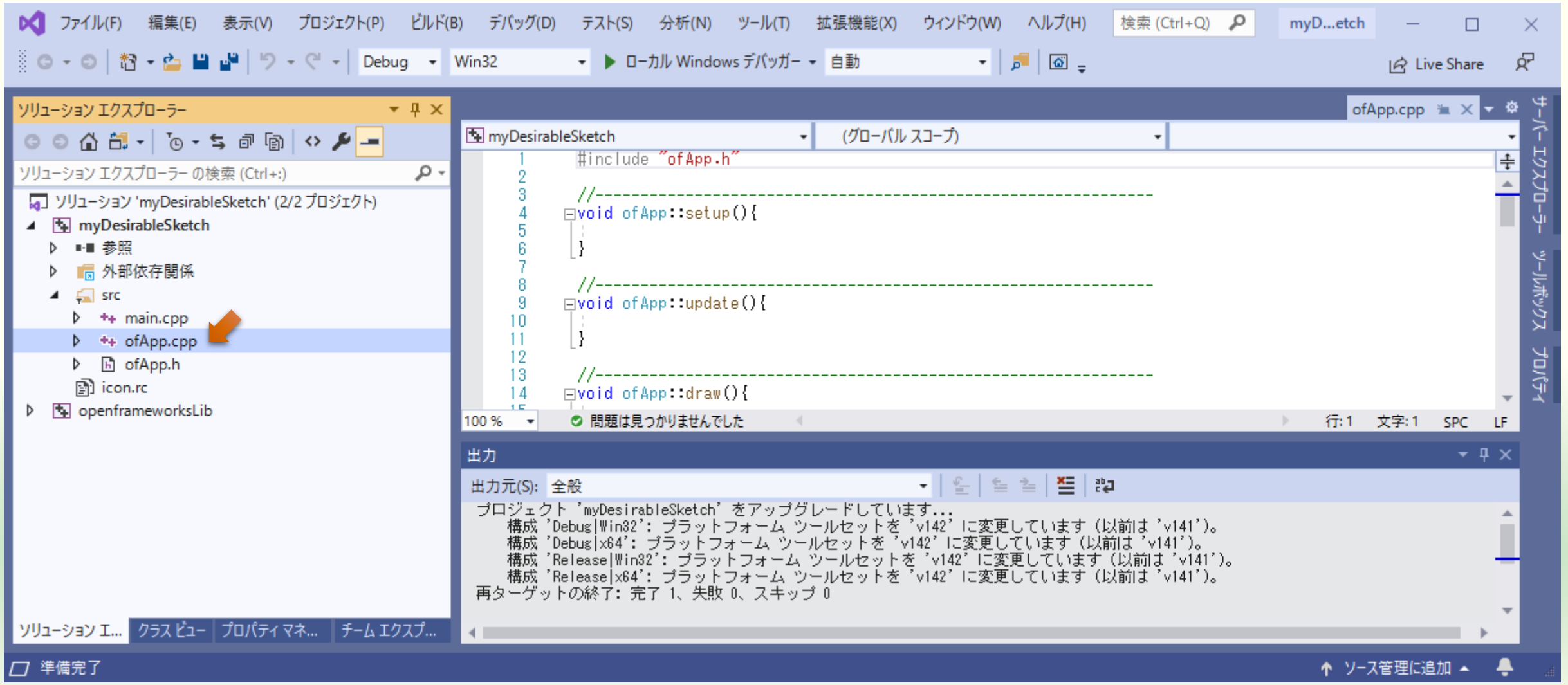

ファイル(F)
編集(E)
表示(V)
プロジェクト(P)
ビルド(B)
デバッグ(D)
テスト(S)
分析(N)
ツール(T)
拡張機能(X)
ウィンドウ(W)
ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myDesirableSketch' (2/2 プロジェクト)
myDesirableSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.h
myDesirableSketch
(グローバル スコープ)
#pragma once
#include "ofMain.h"
class ofApp : public ofBaseApp{
public:
void setup();
void update();
void draw();
void keyPressed(int key);
void keyReleased(int key);
void mouseMoved(int x, int y);
100 %
問題は見つかりませんでした
行: 1
文字: 1
タブ
LF
出力
出力元(S): 全般
プロジェクト 'myDesirableSketch' をアップグレードしています...
構成 'Debug|Win32': プラットフォーム ツールセットを 'v142' に変更しています (以前は 'v141')。
構成 'Debug|x64': プラットフォーム ツールセットを 'v142' に変更しています (以前は 'v141')。
構成 'Release|Win32': プラットフォーム ツールセットを 'v142' に変更しています (以前は 'v141')。
構成 'Release|x64': プラットフォーム ツールセットを 'v142' に変更しています (以前は 'v141')。
再ターゲットの終了: 完了 1、失敗 0、スキップ 0
ソリューション エ...
クラス ビュー
プロパティ マネ...
チーム エクスプ...
サーバー エクスプローラー
ツールボックス
プロパティ
準備完了
ソース管理に追加

ofApp クラスにメンバ変数 p0, p1 追加する

```
#pragma once

#include "ofMain.h"

using namespace glm;

class ofApp : public ofBaseApp{
  vec2 p0, p1;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  （以下略）
```

- p0, p1 は glm::vec2 クラスにする
  - “glm::” という名前空間の指定を省略するために using namespace glm; を入れておく

ofApp.cpp を開く



p0 と p1 にマウスの現在位置を代入する

```
//--------------------------------------------------------
void ofApp::mouseDragged(int x, int y, int button){

}

//--------------------------------------------------------
void ofApp::mousePressed(int x, int y, int button){
  if (button == 0){
    p0 = vec2{ x, y };
  }
}

//--------------------------------------------------------
void ofApp::mouseReleased(int x, int y, int button){
  if (button == 0){
    p1 = vec2{ x, y };
  }
}
```

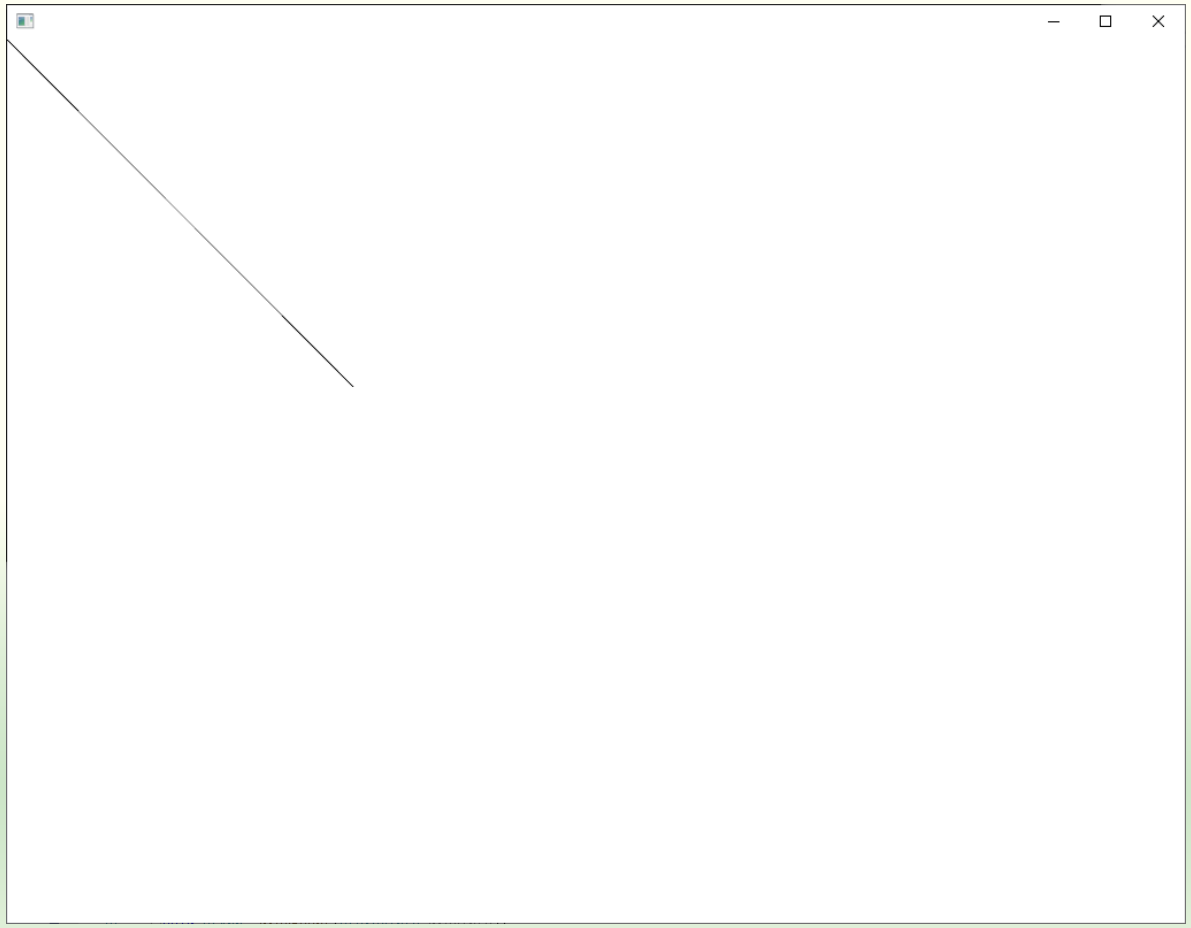
- mousePressed(x, y, button)
  - マウスのボタンを押したときに実行される
  - **もし**押されたボタン button が 0（左）なら起点の位置 p0 にマウスの現在位置 (x, y) を代入する
- mouseReleased(x, y, button)
  - マウスのボタンを離したときに実行される
  - **もし**離したボタン button が 0（左）なら終点の位置 p1 にマウスの現在位置 (x, y) を代入する

p0 から p1 に線分を描く

```
#include "ofApp.h"

//--------------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
}

//--------------------------------------------------
void ofApp::update(){

}

//--------------------------------------------------
void ofApp::draw(){
  ofSetColor(0, 0, 0);
  ofDrawLine(p0, p1);
}
```

- setup() で背景色を白にする
  - ofBackground(int r, int g, int b, int a = 255);
    - r = g = b = 255 は白, a（不透明度）は省略すると 255（不透明）
- draw() で黒い線を描く
  - ofSetColor(int int r, int g, int b), ofSetColor(int int r, int g, int b, int a)
    - r = g = b = 0 は黒, 引数が 3 つのときは不透明度 a に 255 が設定される
  - drawLine(vec2 p0, vec2 p1)
    - p0 から p1 に線分を描く

ボタンを離す前に線が引かれる

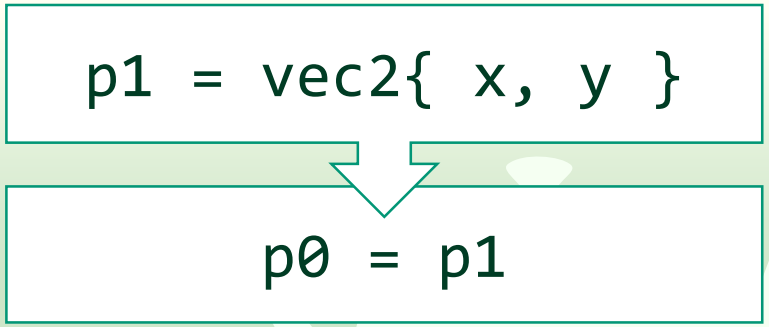


- 左ボタンを押した瞬間に原点から線が引かれてしまう
  - p0, p1 の初期値は vec2{ 0, 0 } すなわち原点なので
  - 左ボタンを押して p0 に値を入れたことで p1 の原点まで線が引かれてしまう
- 実は何もしない状態でも原点に点を描いている
  - p0, p1 が初期値の vec2{ 0, 0 } のままだから

原点までの線が出ないようにする

```
//--------------------------------------------------------
void ofApp::mouseDragged(int x, int y, int button){

}

//--------------------------------------------------------
void ofApp::mousePressed(int x, int y, int button){
  if (button == 0){
    p0 = p1 = vec2{ x, y };
  }
}

//--------------------------------------------------------
void ofApp::mouseReleased(int x, int y, int button){
  if (button == 0){
    p1 = vec2{ x, y };
  }
}
```

- mousePressed(x, y, button)
  - マウスの左ボタンを押したときに終点の p1 にもボタンを押した位置を入れてれば原点までの線は描かれない
  - 代入 = は**右から**順番に実行される



ドラッグ中には何も出ない

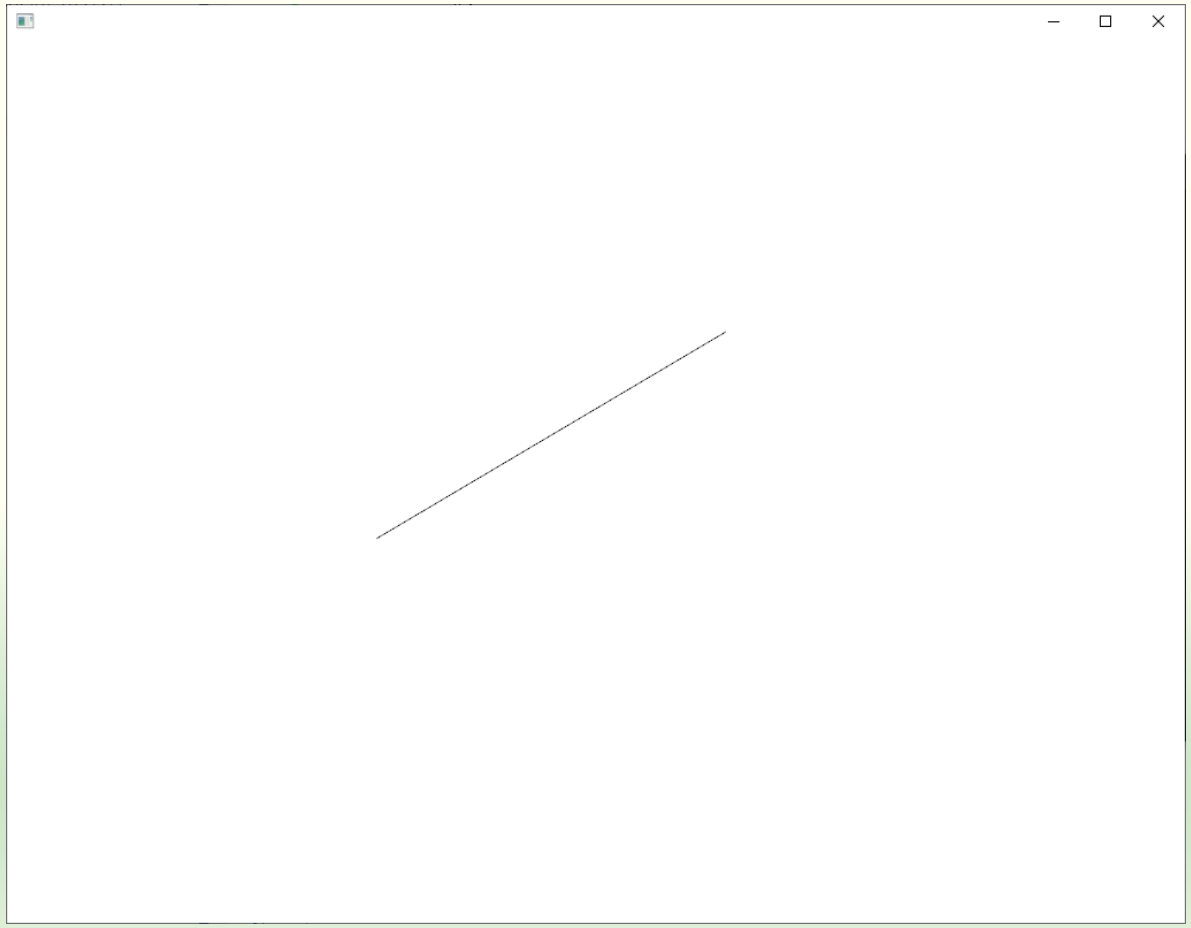
- ドラッグ中は p0 と p1 が等しい
  - マウスの左ボタンを押した位置が入っている
- ボタンを離すと p1 に終点の位置が入る
  - 線が描かれる

ドラッグ中に線を引く

```
//--------------------------------------------------------
void ofApp::mouseDragged(int x, int y, int button){
  if (button == 0){
    p1 = vec2{ x, y };
  }
}

//--------------------------------------------------------
void ofApp::mousePressed(int x, int y, int button){
  if (button == 0){
    p0 = p1 = vec2{ x, y };
  }
}

//--------------------------------------------------------
void ofApp::mouseReleased(int x, int y, int button){
  if (button == 0){
    p1 = vec2{ x, y };
  }
}
```

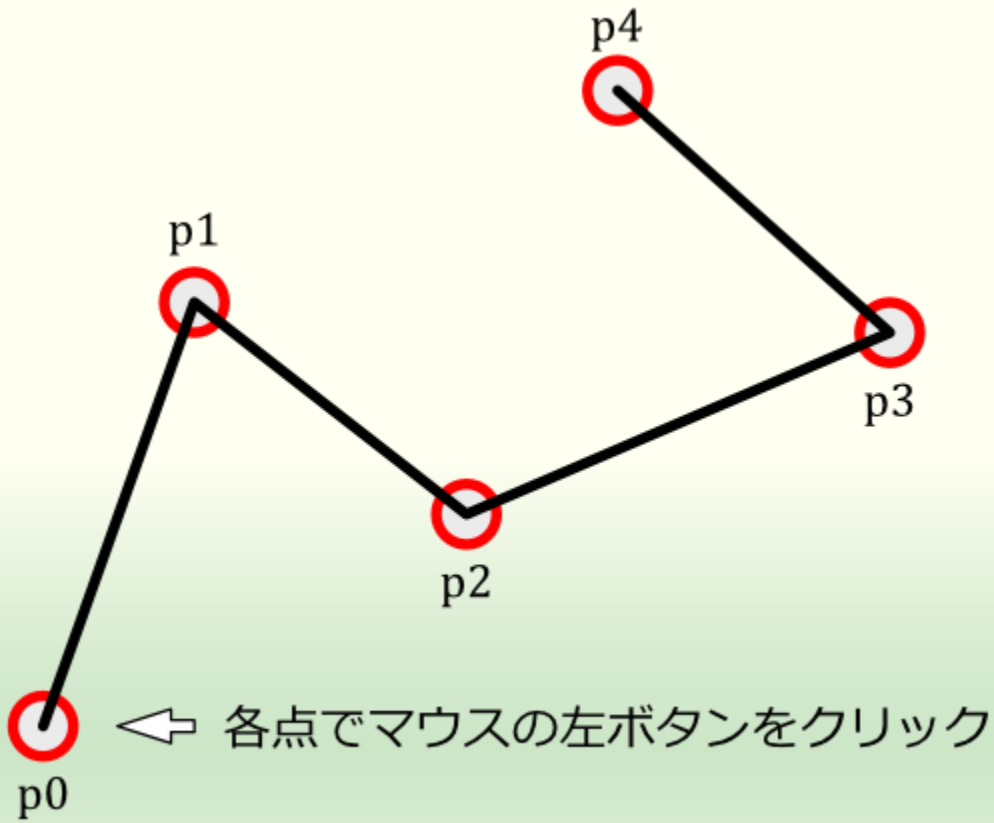
- mouseDragged(x, y, button)
  - マウスのドラッグ中に実行される
  - **もし**ドラッグ中に押されているマウスのボタン button が 0（左）なら終点の位置 p1 にマウスの現在位置 (x, y) を代入する

ドラッグ中も線が引かれる



- マウスのボタンを離すと線が固定される
- しかし次にマウスのボタンを押すと消える

折れ線を描くことを考える



- 左ボタンをクリックするたびにマウスの位置を記録する
  - ドラッグはしない
  - std::vector を使う

ofApp.h でメンバ変数を vector にする

```
#pragma once

#include "ofMain.h"

using namespace glm;

class ofApp : public ofBaseApp{
  vector<vec2> points;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  （以下略）
```

- 変数名は points とかにする

points は空の vector

ofApp.cpp のマウス操作の処理を修正する

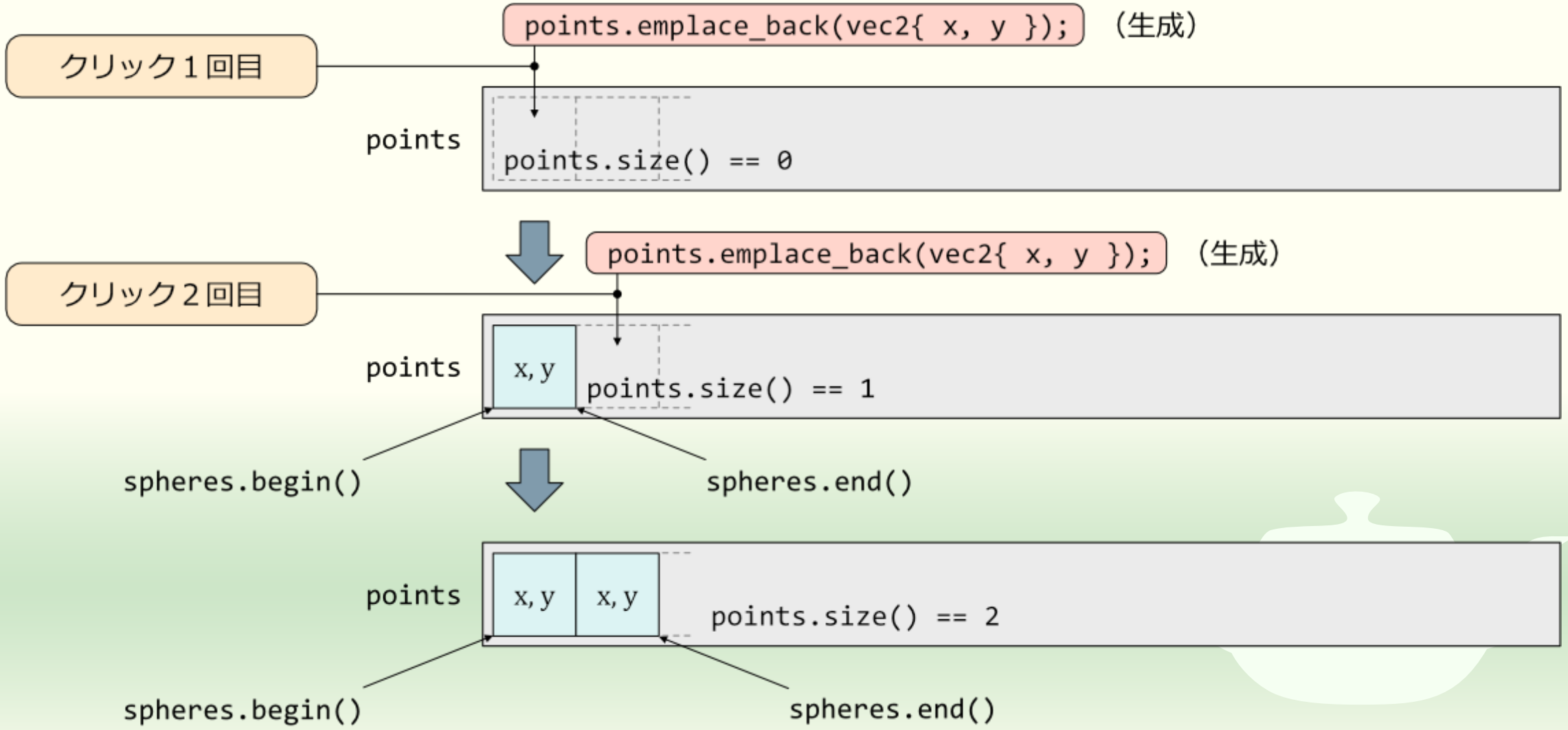
```
//--------------------------------------------------------
void ofApp::mouseDragged(int x, int y, int button){
  if (button == 0){
    p1 = vec2{ x, y };
  }
}

//--------------------------------------------------------
void ofApp::mousePressed(int x, int y, int button){
  if (button == 0){
    points.emplace_back(vec2{ x, y });
  }
}

//--------------------------------------------------------
void ofApp::mouseReleased(int x, int y, int button){
  if (button == 0){
    p1 = vec2{ x, y };
  }
}
```

削除

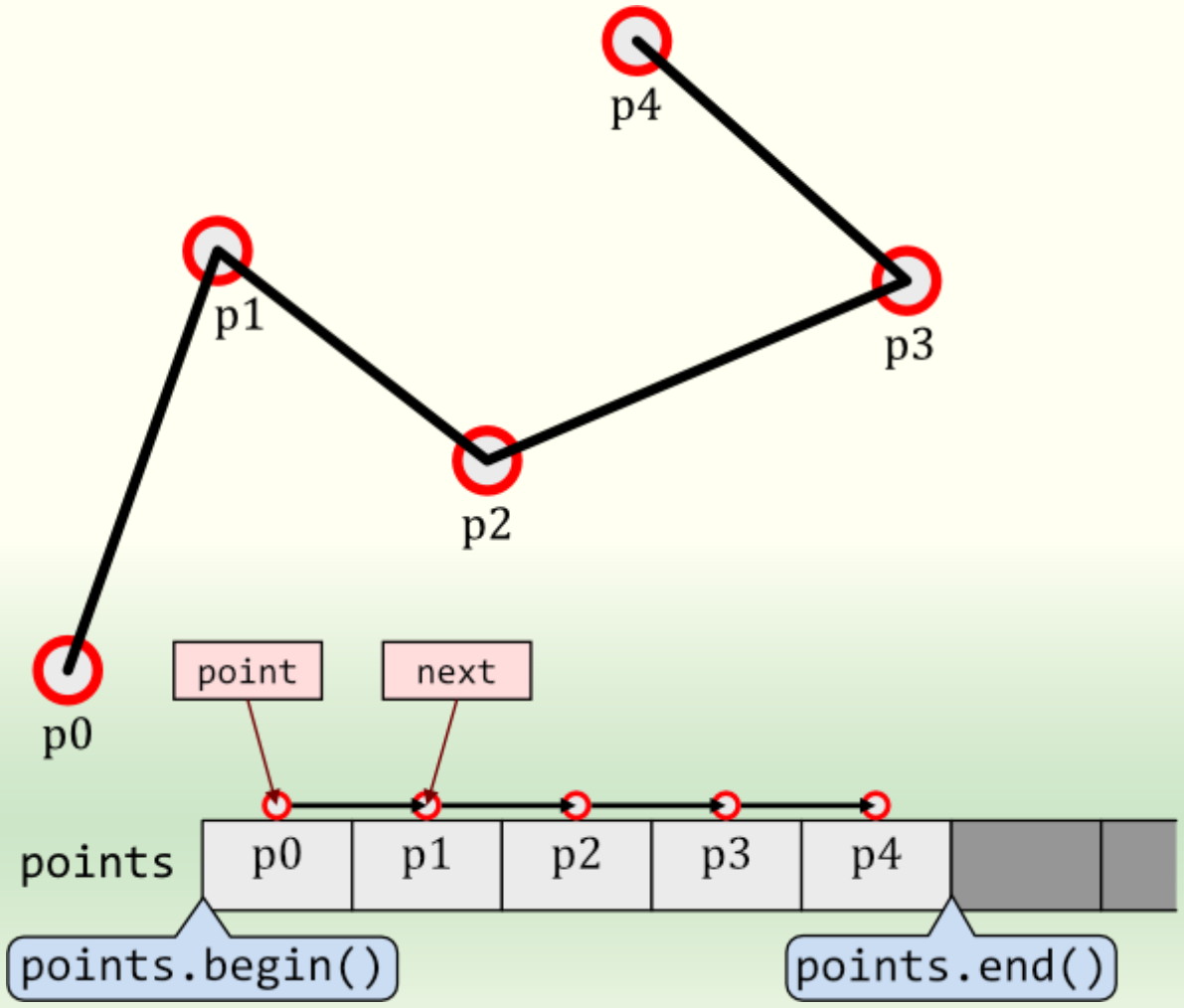
削除

- mousePressed(x, y, button)
  - マウスのボタンが押されたとき押されたボタン button が 0（左）なら points にマウスの現在位置 (x, y) を追加する
- mouseDragged(x, y, button)
  - マウスのドラッグ中は何もしない
- mouseReleased(x, y, button)
  - マウスのボタンを離したときも何もしない

emplace_back() で vector の末尾に要素を生成

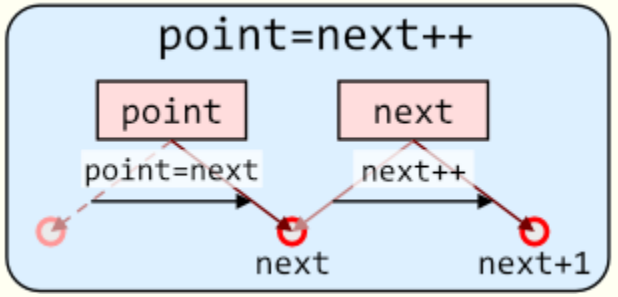


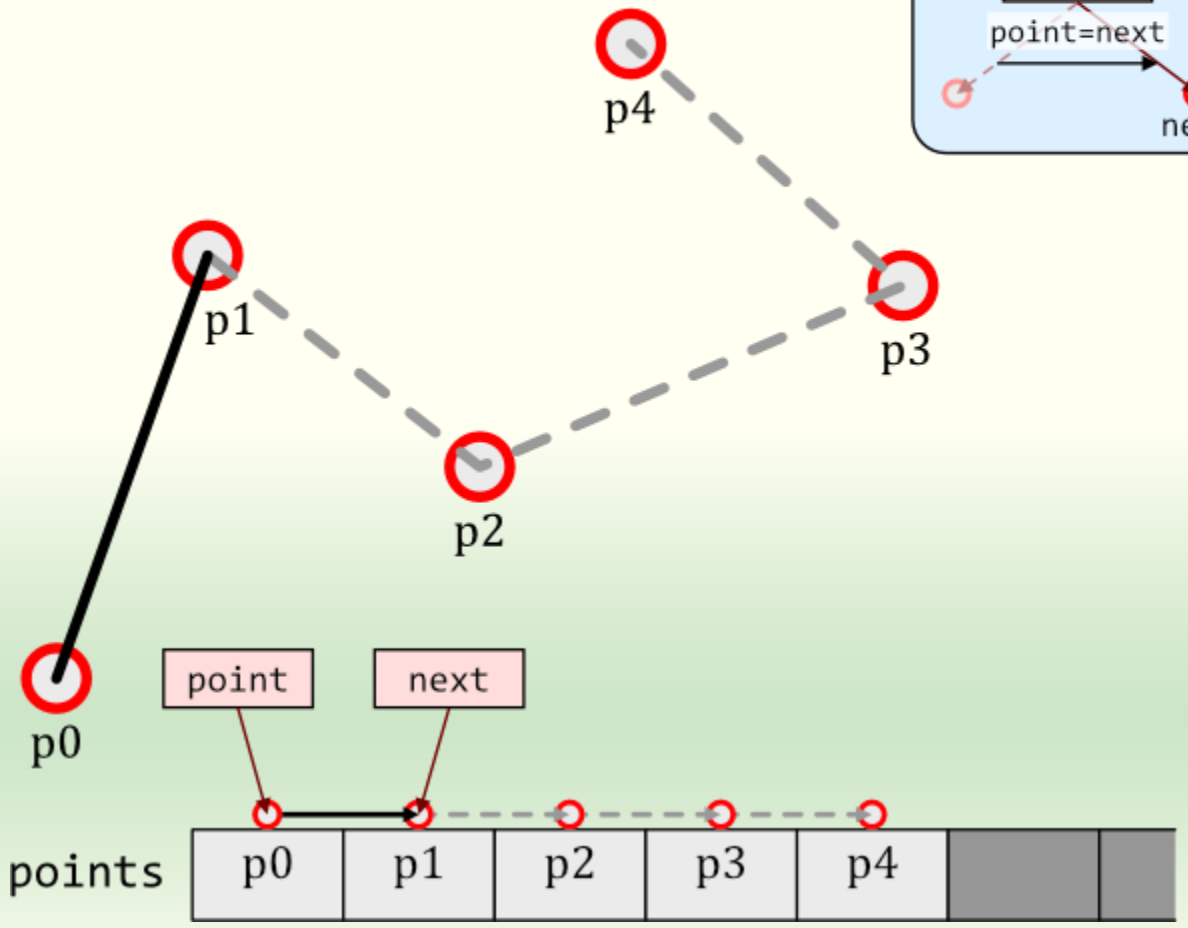
クリックしたところを線分で結ぶ

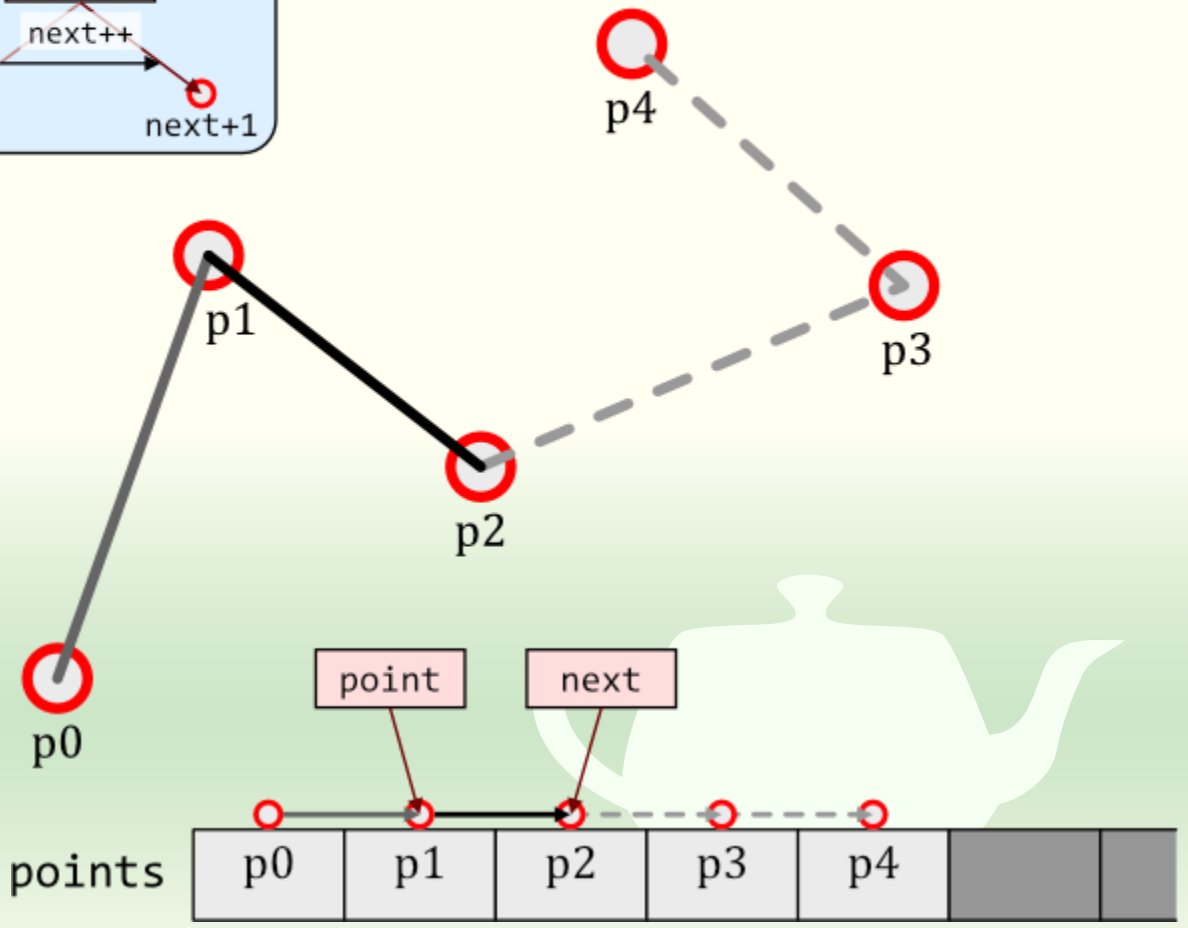


- クリックした位置は points という vector に p0 → p1 → p2 → … という順に入っている
- 最初 point に p0 の**場所**、next に p1 の**場所**を入れておく
- point が指すデータから next が指すデータに線分を引く
- point に next を代入し、next に 1 を足して次のデータに進む
- next がデータの終わりを超えていたら終わる

折れ線の描画


point=next++
point
next
point=next
next++
next
next+1


p4
p1
p3
p2
p0
point
next
points
p0
p1
p2
p3
p4


p4
p1
p3
p2
p0
point
next
points
p0
p1
p2
p3
p4

vector が空でなければ最初のデータを取り出す

```
#include "ofApp.h"

//--------------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
}

//--------------------------------------------------
void ofApp::update(){

}

//--------------------------------------------------
void ofApp::draw(){
  if (points.empty()) return;
  auto point = points.begin();
```

（次のページに続く）

- if (points.empty()) return;
  - points にデータが入っていなければ何もせずに**戻る**
    - **empty()** メソッドは vector の中に何も入っていなければ true になる
    - if の { } 内に ;（セミコロン）がひとつしかないときは { } を省略できる
- auto point = points.begin();
  - points の最初のデータ（p0）の場所を point に代入する
    - **begin()** メソッドは vector の最初のデータの場所（**イテレータ**）を取り出す
    - begin() メソッドは vector<vec2>::iterator というデータ型だが長いので point は **auto** を使って型推論して宣言している

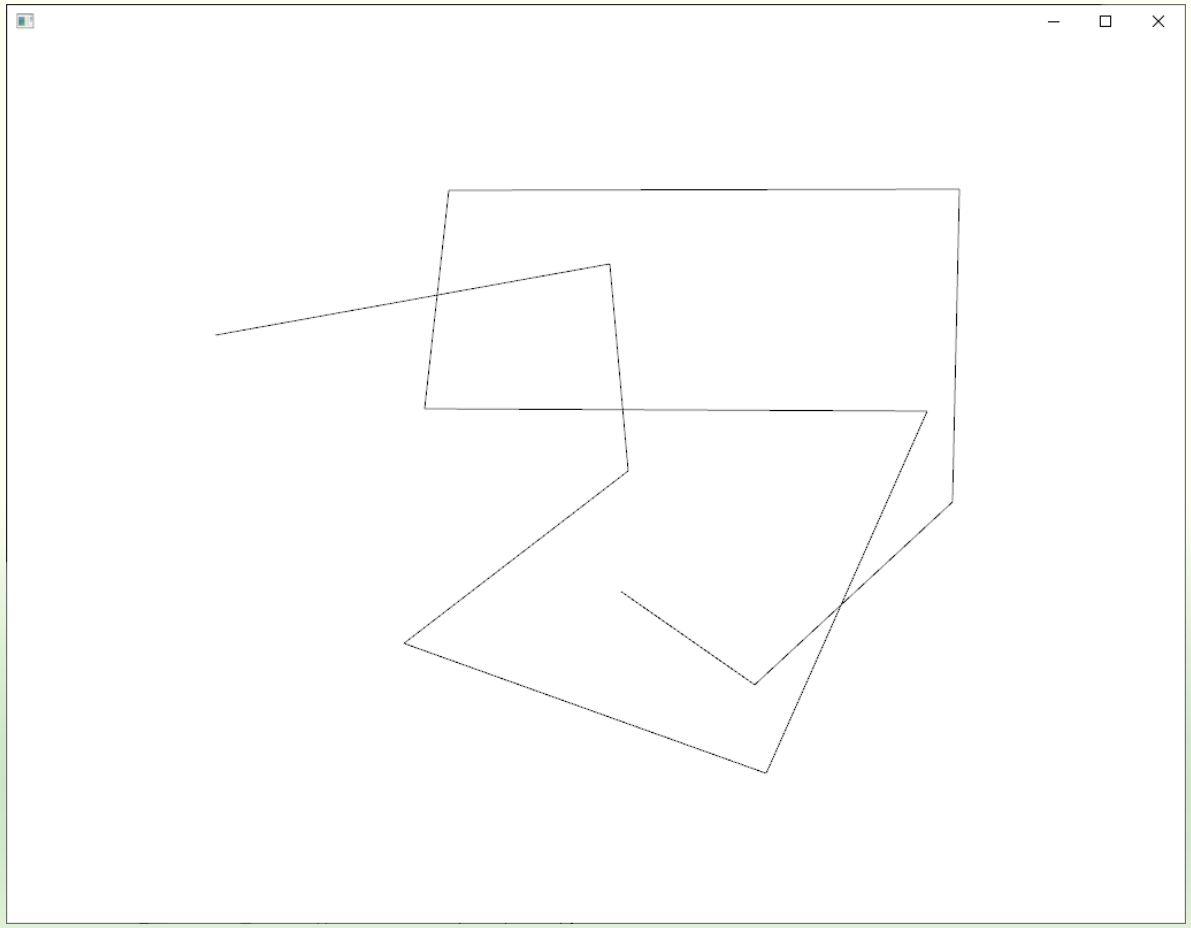
2つずつ点を取り出して線分で結ぶ

```
    （前のページの続き）
  for (auto next = point + 1;
    next != points.end(); point = next++){
    ofSetColor(0, 0, 0);
    ofDrawLine(*point, *next);
  }
}
```

間接参照演算子 (*) を使って
イテレータが指している
データの実体を取り出す

- auto next = point + 1
  - point の次のデータ（最初は p1）の場所を next に代入する
    - イテレータに 1 を足すと次のデータの場所が得られる
- next != points.end()
  - next が points の最後のデータの次でなければ { } 内を実行する
    - **end()** メソッドは vector の最後のデータの次の場所を取り出す
- point = next++
  - point に next を代入した**後**に next に 1 を足して次の場所に進める

折れ線が描ける

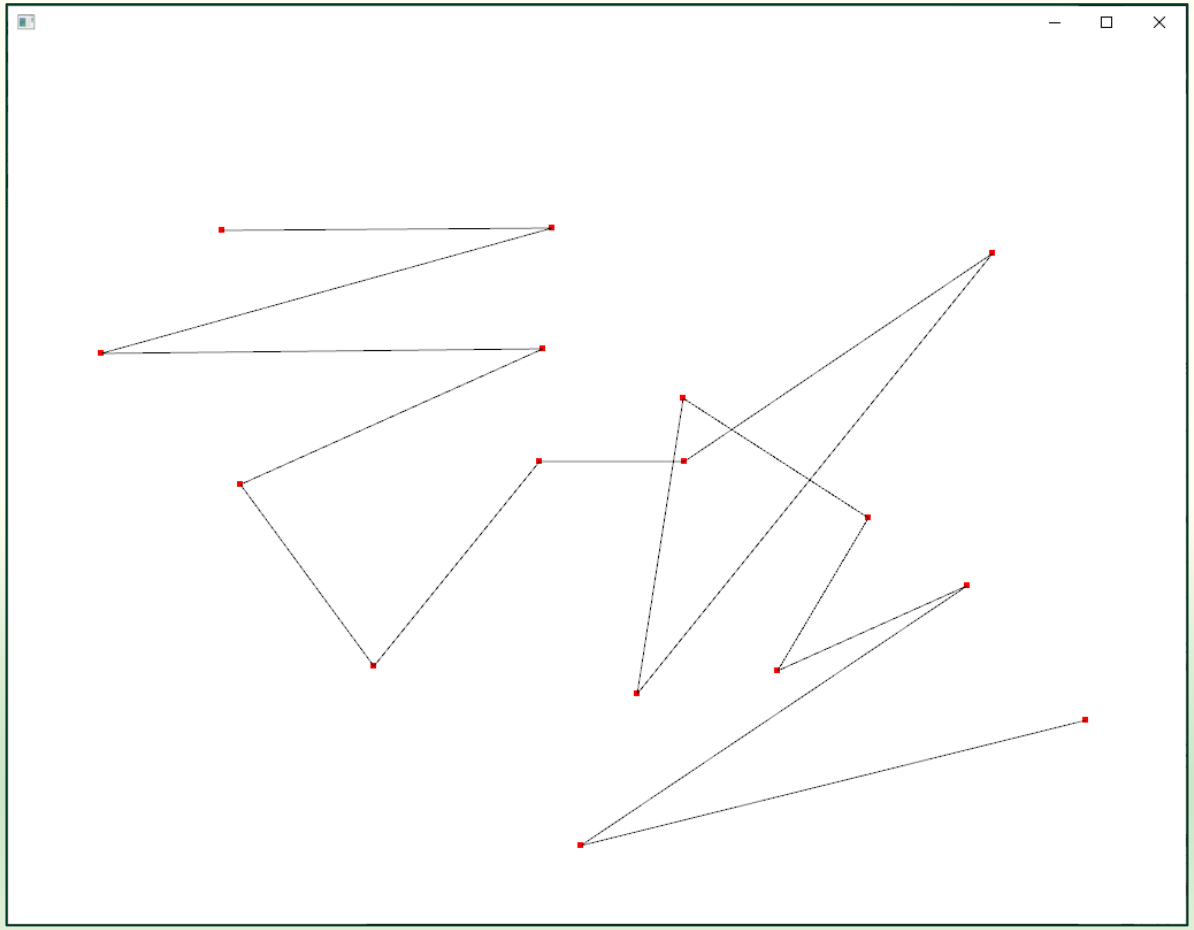


- ひと続きの折れ線しか描けない

課題４－１

クリックしたところに点を打つ

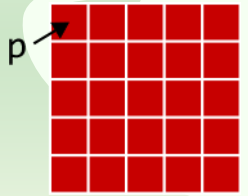
クリックしたところに点を打つ



- クリックしたところに点を打つようにしてください
- 縦横５ドットの赤い点を打つとしたら次のようになります

```
ofSetColor(255, 0, 0);
ofDrawRectangle(p, 5, 5);
```

  - ofDrawRectangle()は矩形を描く
  - p は左上の位置で vec2 型の値



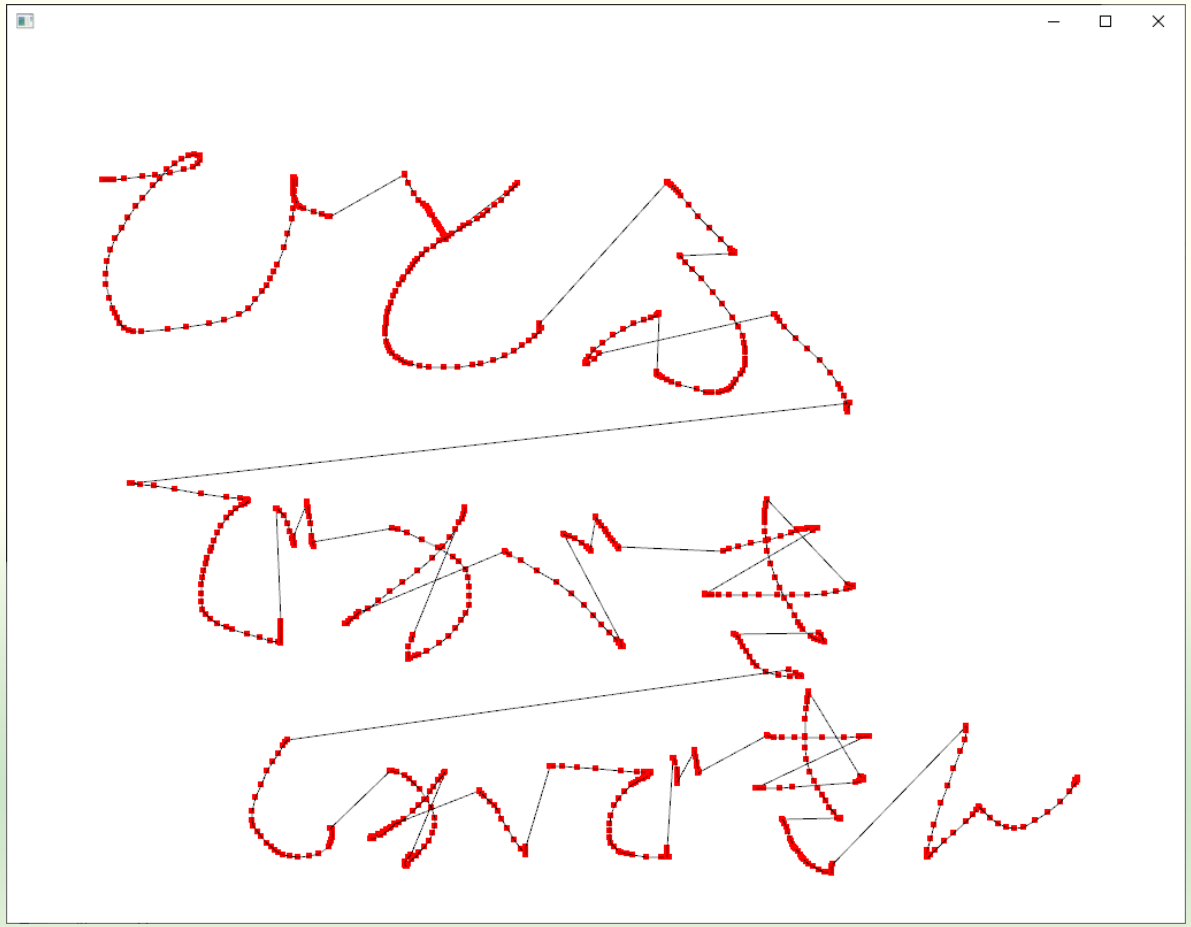
課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **4-1.png** というファイル名で保存し、Moodle の第４回課題にアップロードしてください

課題４－２

ドラッグ中も線を引く

ドラッグ中も線を引く



- マウスの左ボタンを押しながらドラッグしているときも線を引くようにしてください

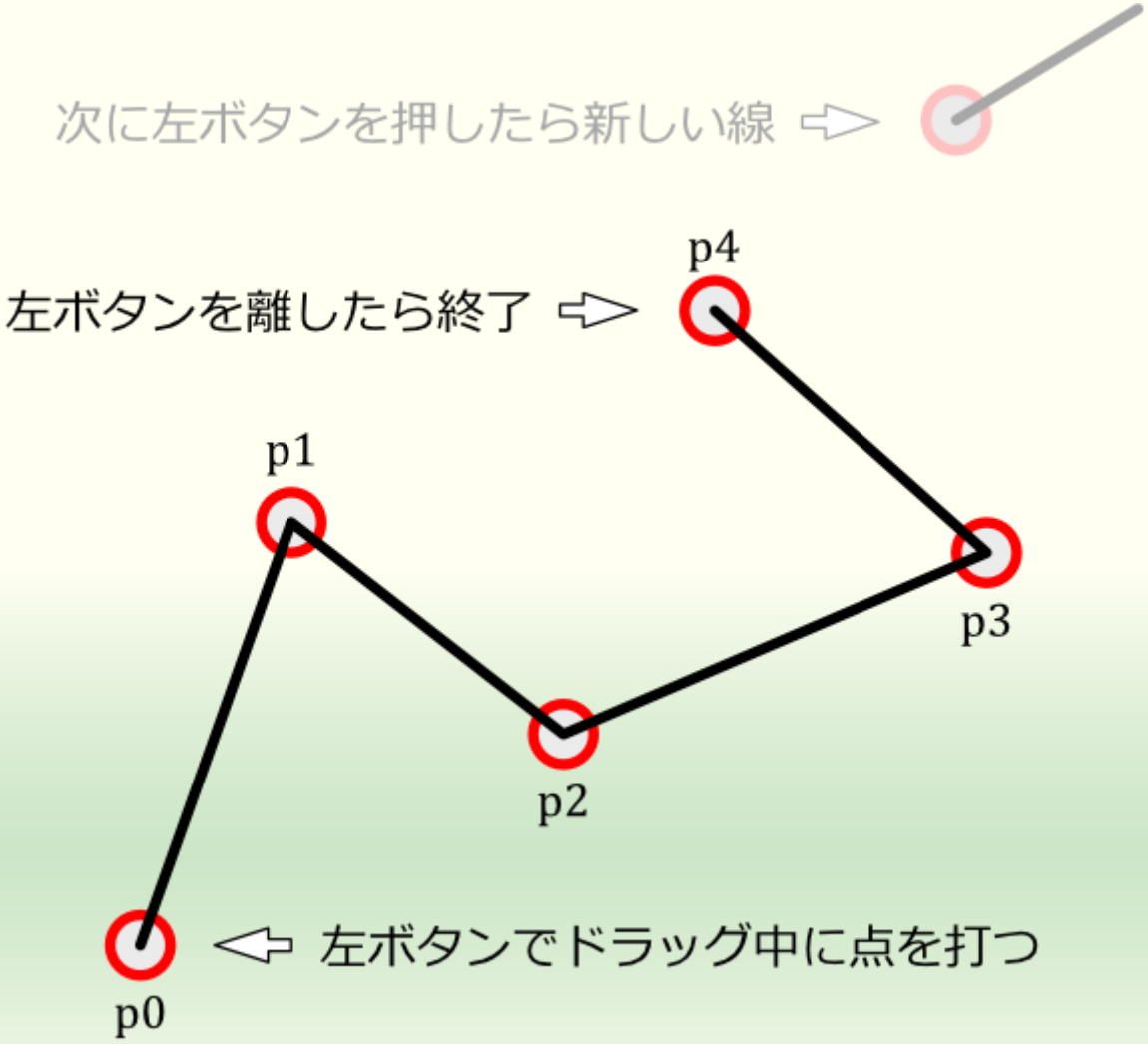
課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **4-2.png** というファイル名で保存し、Moodle の第４回課題にアップロードしてください

課題４－３

複数の折れ線を描く

複数の折れ線を描く



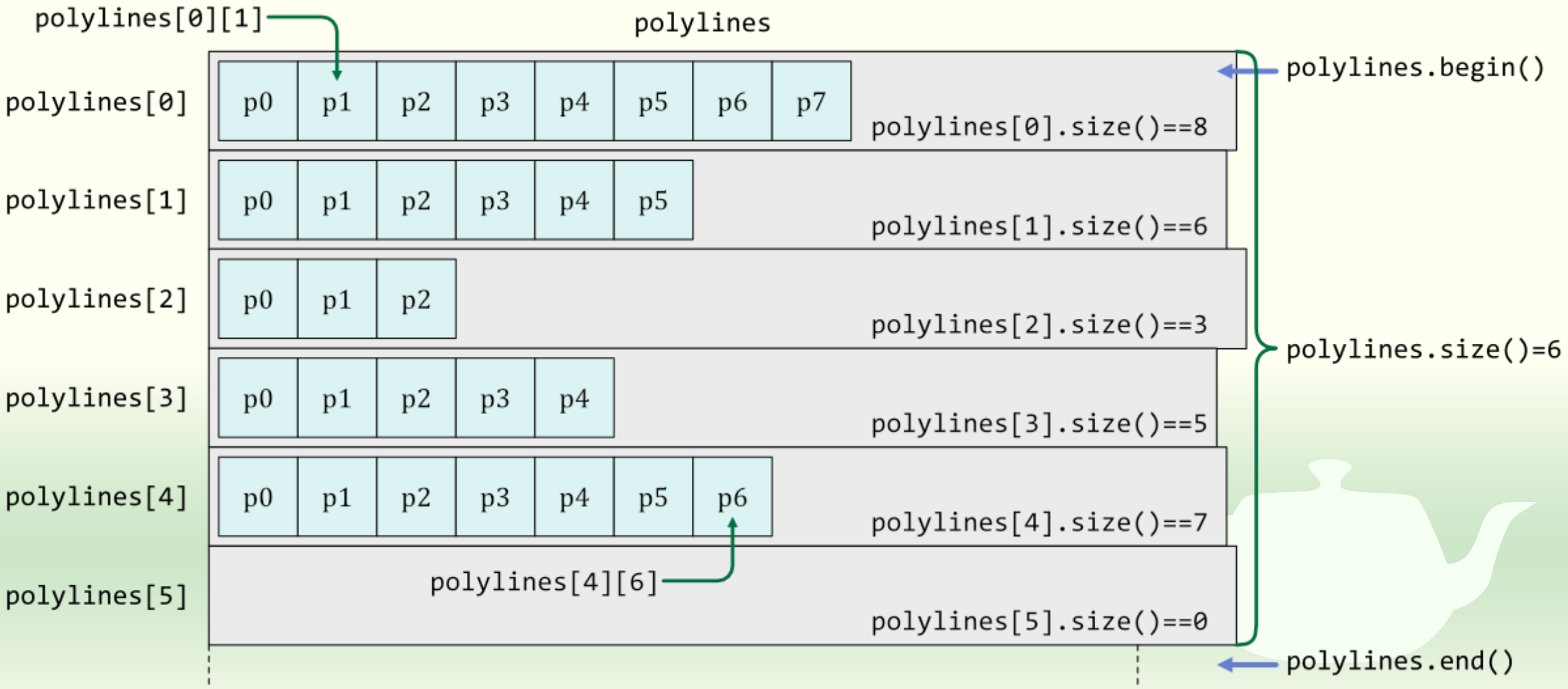
- 左ボタンを押しながらマウスをドラッグしている間に点を打つ
- 左ボタンを離したら折れ線の作成を終わる
  - 次に左ボタンを押したら新しい折れ線を描き始める

ofApp.h で折れ線の vector を作る

```
#pragma once

#include "ofMain.h"

using namespace glm;

class ofApp : public ofBaseApp{
  vector<vector<vec2>> polylines;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  （以下略）
```

- vector<vec2>
  - 点のコンテナ（vec2）の vector
  - 折れ線のコンテナ
- **vector<**vector<vec2>**>** polylines;
  - 折れ線のコンテナ（vector<vec2>）の vector
- vector の vector

vector<vector<vec2>> polylines; のイメージ



polylines に空のコンテナを追加する

```
#include "ofApp.h"

//--------------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
  polylines.emplace_back();
}

//--------------------------------------------------
void ofApp::update(){

}
```

（次のページに続く）

- polylines は空
  - 折れ線を格納する vector<vec2> のコンテナが一つもない
    - 点が追加できない
- 最初に polylines に vector<vec2> のコンテナを追加する
  - 内容は空
    - マウスのドラッグしたときに点のデータをこのコンテナに追加する
    - 点のデータは常に polylines の最後のコンテナに追加するようにする

全部の折れ線を描画する

```
//--------------------------------------------
void ofApp::draw(){
  for (auto &points : polylines){
    if (points.empty()) return;
    auto point = points.begin();
    for (auto next = point + 1;
      next != points.end(); point = next++){
      ofSetColor(0, 0, 0);
      ofDrawLine(*point, *next);
    }
  }
}
```

- polylines の一つ一つの要素（折れ線のコンテナ）を取り出す
- 取り出したコンテナに格納されている折れ線を描画する
- この処理を繰り返す

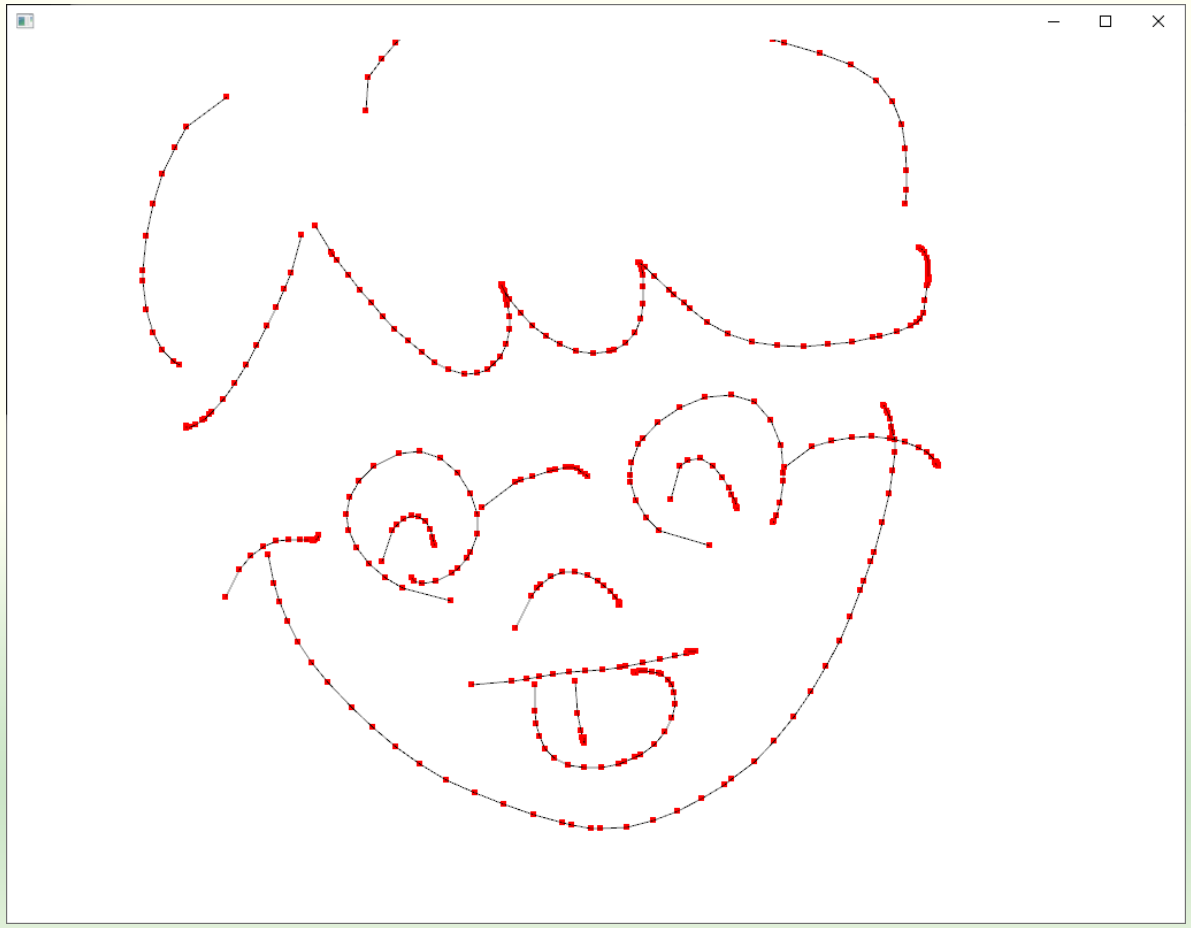
このプログラムには点を描く処理を含めていませんが点も描くようにしてください

左ボタンでドラッグした時に点を追加する

```
//--------------------------------------------------------
void ofApp::mouseDragged(int x, int y, int button){
  if (button == 0){
    auto &points = polylines.back();
    points.emplace_back(vec2{ x, y });
  }
}

//--------------------------------------------------------
void ofApp::mousePressed(int x, int y, int button){
  if (button == 0){
    auto &points = polylines.back();
    points.emplace_back(vec2{ x, y });
  }
}

//--------------------------------------------------------
void ofApp::mouseReleased(int x, int y, int button){

}
```

- polylines の最後の要素のコンテナを取り出す
- 取り出したコンテナに点を追加する

複数の折れ線を描く



- 複数の折れ線を描くようにしなさい
  - マウスのドラッグを終了して左ボタンを**離したら**１本の線分の作成を終える
  - 次にマウスの左ボタンを押してドラッグを開始したら新しい折れ線を描く
- polylines の最後に空のコンテナを１つ**追加する**だけ

課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **4-3.png** というファイル名で保存し、Moodle の第４回課題にアップロードしてください

画像データの変更

ペイントツールっぽいもの

ofApp クラスに画像のメンバ変数を追加する

```
#pragma once

#include "ofMain.h"

using namespace glm;

class ofApp : public ofBaseApp {
  vector<vector<vec2>> polylines;
  ofImage image;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  (以下略)
```

- ofImage
  - 画像の読み込み、保存、描画を行うクラス
    - 画面に画像を描画する
    - 画像のピクセルデータを操作する
    - 画像ファイルを読み込む
    - OpenGL テクスチャを作成する

ofApp.cpp の setup() で image に画像を読み込む

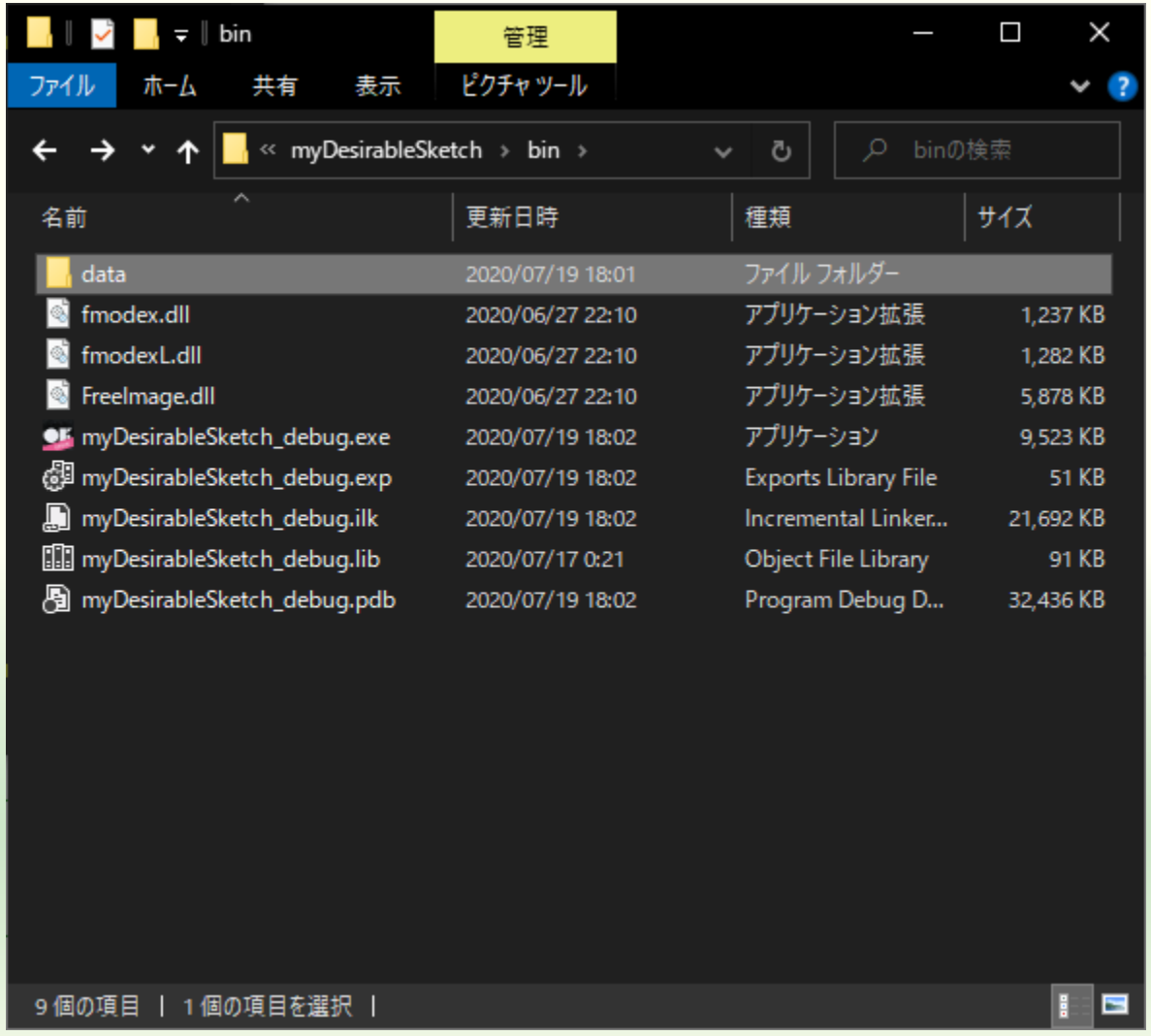
```
#include "ofApp.h"

//--------------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
  polylines.emplace_back();
  image.load("image.jpg");
}
```

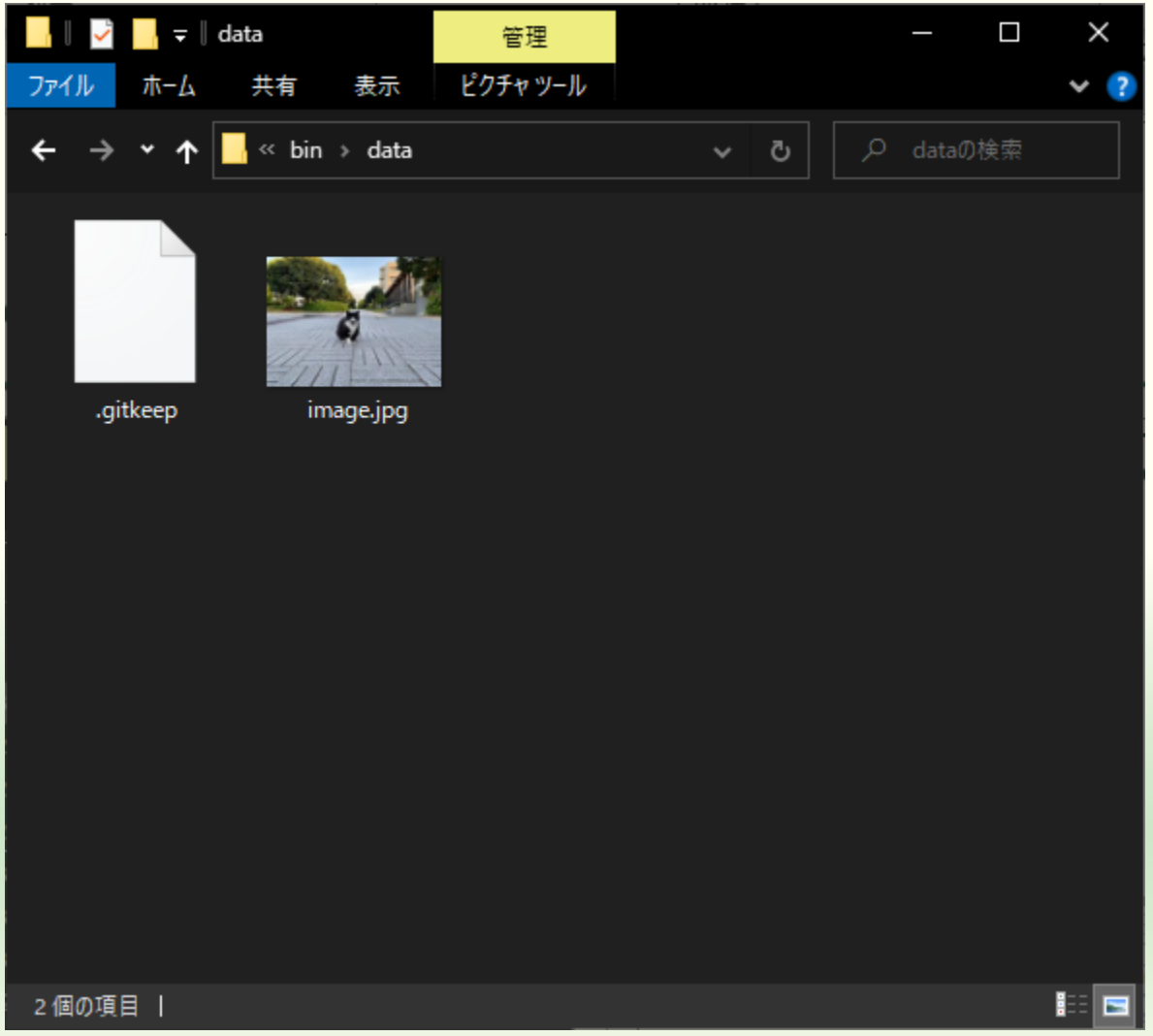
- image.load("image.jpg");
  - プロジェクトのフォルダの bin の data の中にある image.jpg という画像ファイルを image に読み込む
- "image.jpg" は画像ファイル名
  - JPEG, PNG, GIF 画像が読み込める

load() メソッド

- bool ofImage_::load(const filesystem::path &fileName, const ofImageLoadSettings &settings)
  - filename: 画像ファイルのパス
  - settings: 画像ファイルの読み込み設定
    - メンバ: bool accurate, bool exifRotate, bool grayscale, bool separateCMYK
    - 省略可能
- fileName に指定した画像ファイルから画像を読み込む
  - JPEG, PNG, GIF 形式のファイルが読み込み可能
  - パスを省略したときはプロジェクトのフォルダの **bin/data** の中
  - 戻り値は読み込みに成功したとき true、失敗したとき false を返す

画像は bin フォルダの中の data に配置する





draw() で image を描画する

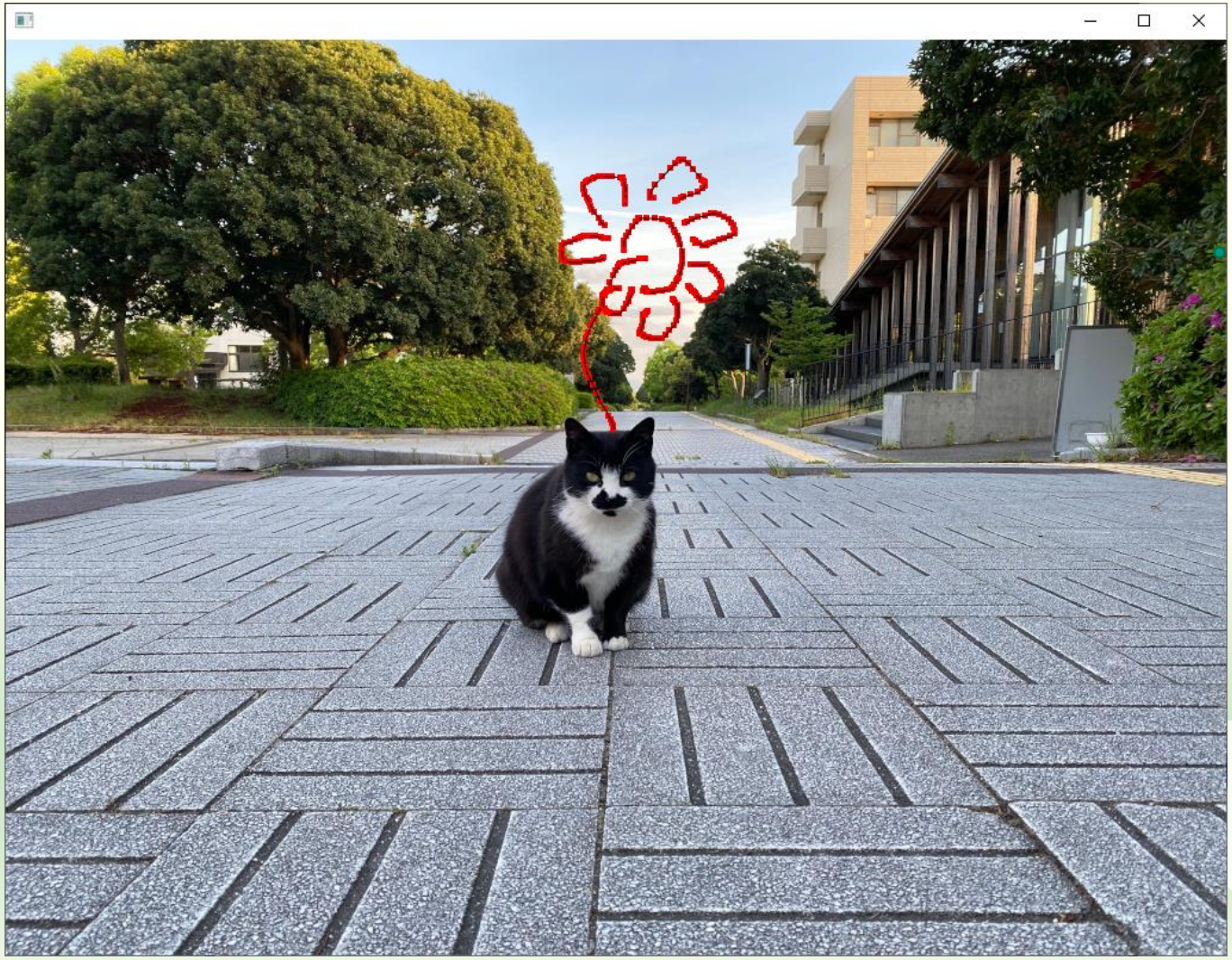
```
//--------------------------------------------
void ofApp::draw(){
  ofSetColor(255, 255, 255);
  image.draw(0, 0);
  for (auto &points : polylines){
    if (points.empty()) return;
    auto point = points.begin();
    ofSetColor(255, 0, 0);
    ofDrawRectangle(points[0] - 3.0f, 5, 5);
    for (auto next = point + 1;
      next != points.end(); point = next++){
      ofSetColor(255, 0, 0);
      ofDrawRectangle(*next - 3.0f, 5, 5);
      ofSetColor(0, 0, 0);
      ofDrawLine(*point, *next);
    }
  }
}
```

- image.draw(0, 0);
  - image をウィンドウの (0, 0) の位置に描画する
- 表示は image の色と ofSetColor() で設定された色との積になる
  - ofSetColor(255, 0, 0); とすると画像の赤成分だけが表示される
  - ofSetColor(127, 127, 127); とすると明度が半分になる
  - ofSetColor(0, 0, 0); とすると画像の内容にかかわらず黒になる

draw() メソッド

- void ofImage_::draw(float x, float y)
  - ウィンドウの (x, y) の位置を左上として画像を描画する
  - ウィンドウからはみ出た部分は表示されない
- void ofImage_::draw(float x, float y, float w, float h)
  - ウィンドウの (x, y) の位置を左上として幅 w 高さ h の領域に画像を拡大縮小して描画する
    - image.draw(0, 0, ofGetWidth(), ofGetHeight()); とすると画像全体が常にウィンドウいっぱいに表示される
  - ウィンドウからはみ出た部分は表示されない

実行結果



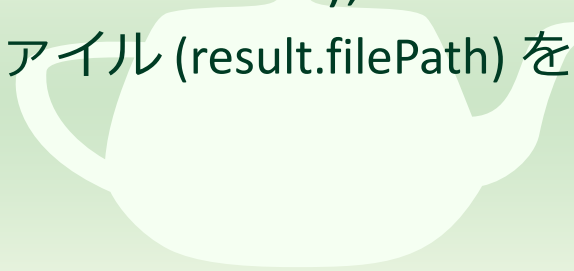
折れ線も
描ける

ファイル選択ダイアログを使う

```
//--------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
  polylines.emplace_back();
  image.load("image.jpg");
  auto result = ofSystemLoadDialog();
  if (result.bSuccess){
    image.load(result.filePath);
  }
}
```

注意
ofSystemLoadDialog() は日本語を含むファイルパスに対応していない（日本語が？になってしまう）

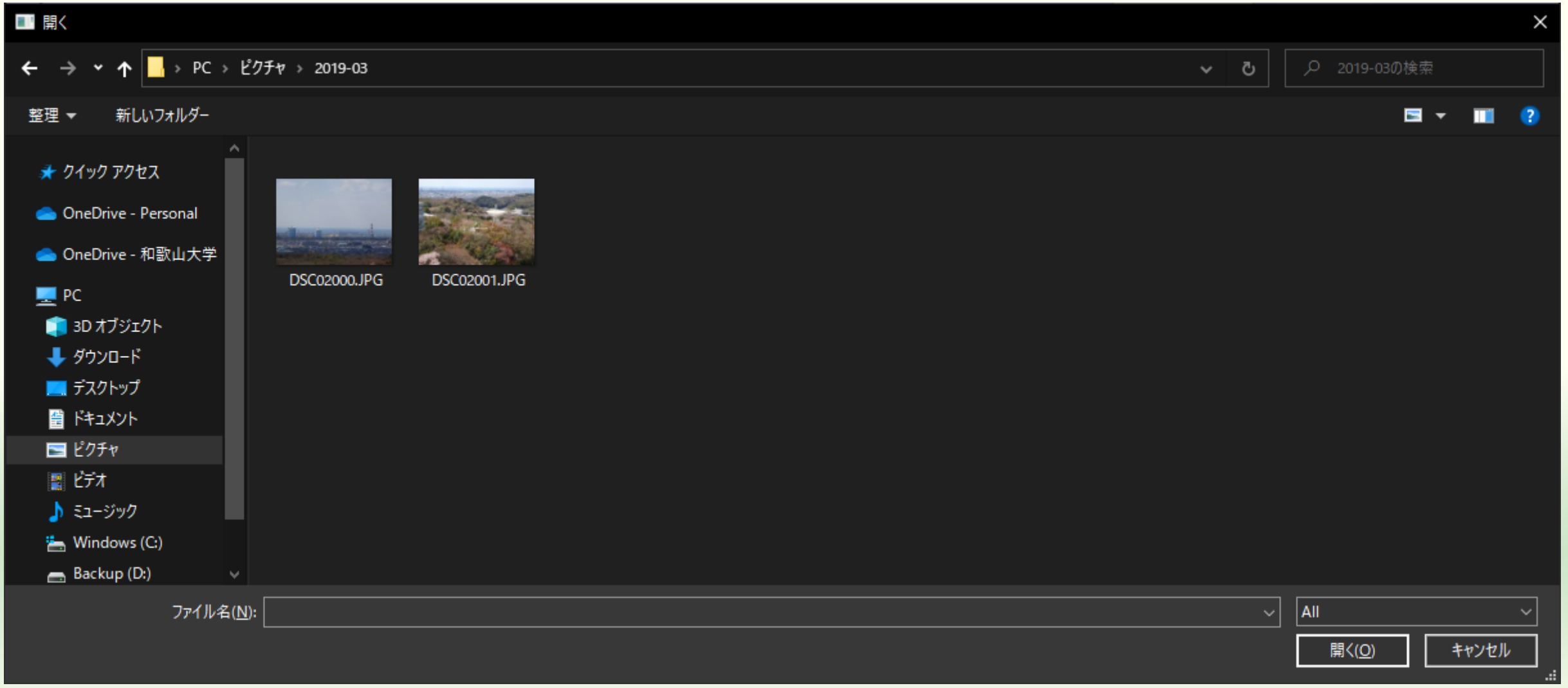
- 好きなところからファイルを読み込みたい場合
- auto result = ofSystemLoadDialog();
  - ファイル選択ダイアログを開く
- **if** (result.bSuccess){
  - もしファイルの選択に成功していれば
  - image.load(result.filePath);
    - 選択したファイル (result.filePath) を読み込む



ofSystemLoadDialog()

- ofFileDialogResult ofSystemLoadDialog(string windowTitle, bool bFolderSelection=false, string defaultPath)
  - windowTitle: ダイアログウィンドウのタイトルバーに表示する文字列
  - bFolderSelection: true ならフォルダの選択, false ならファイルの選択
  - defaultPath: ファイルパスを省略したときのデフォルトのパス名
- ofFileDialogResult
  - bSuccess: ファイルの選択がキャンセルされずに成功したら true
  - getName(), fileName: ファイル名
  - getPath(), filePath: ファイル名のフルパス

ファイル選択ダイアログウィンドウ



読み込みに失敗したらエラーを表示する

```
//--------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
  polylines.emplace_back();
  image.load("image.jpg");
  auto result = ofSystemLoadDialog();
  if (result.bSuccess){
    if (!image.load(result.filePath)){
      ofSystemAlertDialog("Can't load file: "
        + result.filePath);
    }
  }
}
```

注意
ofSystemAlertDialog() は日本語を含むメッセージに対応していない（日本語が文字化けする）

- **if** (!image.load(result.filePath)){
  - **もし** result.filePath の読み込みに成功**しなかったら**
    - ! は論理反転演算子、true→false, false→true
  - ofSystemAlertDialog("message")
    - message を表示する
- "Can't load file: " + result.filePath
  - "Can‘t load file: " に result.filePath の内容を連結する
  - このときの + は文字列 (string) を連結する演算子

‘o’ か ‘O’ キーのタイプでファイルを読み込む

```
//--------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
  polylines.emplace_back();
  image.load("image.jpg");

}

 (途中略)

//--------------------------------------------
void ofApp::keyPressed(int key){
  if (key == 'o' || key == 'O'){
    auto result = ofSystemLoadDialog();
    if (result.bSuccess){
      if (!image.load(result.filePath)){
        ofSystemAlertDialog("Can't load file: "
          + result.filePath);
      }
    }
  }
}
```

移動

- プログラム実行時に毎回ファイル選択ダイアログを表示したくない
- keyPressed()
  - キーボードのキーを押したときに実行される
- **if** (key == 'o' || key == 'O'){
  - **もし**押されたキー key が ‘o’ または ‘O’ なら
    - ファイル選択ダイアログを表示する
    - 選択された画像ファイルを読み込む

Windows で日本語のファイル名を選択する

```
//--------------------------------------------------------------
#include <commdlg.h>

void ofApp::keyPressed(int key){
  if (key == 'o' || key == 'O'){
    TCHAR filePath[MAX_PATH] = TEXT("");
    OPENFILENAME ofn;

    memset(&ofn, 0, sizeof(OPENFILENAME));
    ofn.lStructSize = sizeof(OPENFILENAME);
    ofn.lpstrFilter = TEXT("JPEG (*.jpg)¥0*.jpg¥0PNG (*.png)¥0*.png¥0GIF (*gif)¥0All (*.*)¥0*.*¥0¥0");
    ofn.lpstrFile = filePath;
    ofn.nMaxFile = sizeof(filePath);
    ofn.Flags = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;
    ofn.lpstrDefExt = TEXT("jpg");

    if (GetOpenFileName(&ofn) && !image.load(filePath)) {
      MessageBox(NULL, filePath, TEXT("ファイルが開けません"), MB_ICONERROR);
    }
  }
}
```

WIN32（Windows の機能）を直接使って
ファイル選択ダイアログを表示する

ofApp クラスにブラシのメンバ変数を追加する

```
#pragma once

#include "ofMain.h"

using namespace glm;

class ofApp : public ofBaseApp {
  vector<vector<vec2>> polylines;
  ofImage image, brush;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  (以下略)
```

- polyline は image に読み込んだ画像とは別に表示している
  - image の表示に重ねて表示しているが image の内容を変更しているわけではない
- 追加した画像の ofImage のインスタンス brush は image に格納されている画像そのものを修正するために使う
  - そのため、ここでは brush（ブラシ）という変数名にした

brush にメモリを割り当てる

```
//--------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
  polylines.emplace_back();
  brush.allocate(50, 50, OF_IMAGE_COLOR);
}
```

- brush には画像を読み込むのではなくメモリだけを割り当てる
- brush.allocate(50, 50, OF_IMAGE_COLOR);
  - brush のサイズは縦 50 画素（ピクセル）、横 50 画素
  - OF_IMAGE_COLOR は R, G, B の 3 つのチャネルをもつフルカラー画像

allocate() メソッド

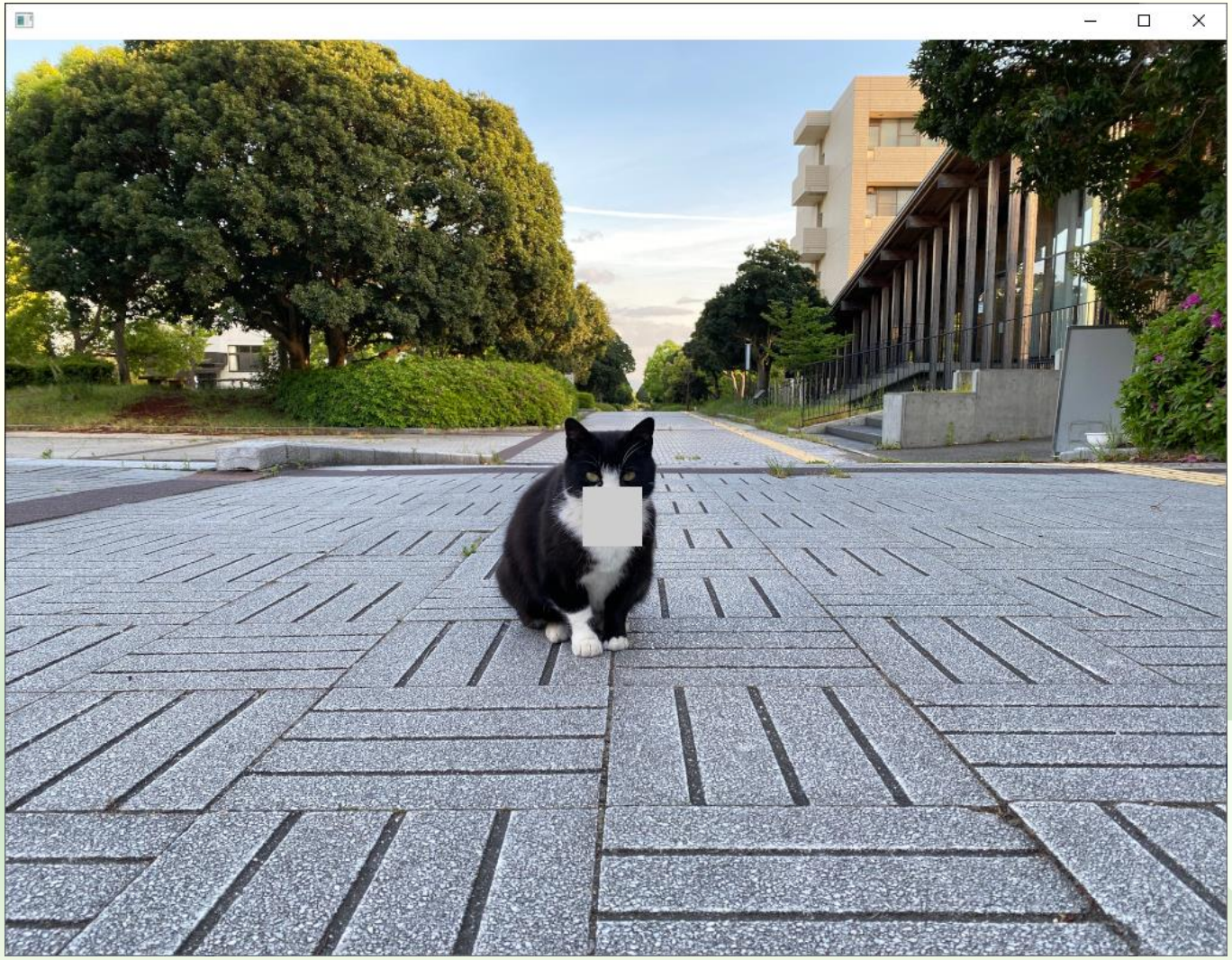
- void ofPixels_::allocate(size_t w, size_t h, ofImageType imageType)
- void ofPixels_::allocate(size_t w, size_t h, ofPixelFormat pixelFormat)
- void ofPixels_::allocate(size_t w, size_t h, size_t channels)
  - w, h: メモリを割り当てる画像のサイズ
  - imageType: OF_IMAGE_GRAYSCALE, OF_IMAGE_COLOR, OF_IMAGE_COLOR_ALPHA
  - pixelFormat: OF_PIXELS_RGB, OF_PIXELS_RGBA, OF_PIXELS_BGRA, OF_PIXELS_MONO
  - channels: １画素当たりのチャネル数

brush を image に重ねて描画する

```
//--------------------------------------------
void ofApp::draw(){
  ofSetColor(255, 255, 255);
  image.draw(0, 0);
  brush.draw(mouseX, mouseY);
  for (auto &points : polylines){
    if (points.empty()) return;
    auto point = points.begin();
    ofSetColor(255, 0, 0);
    ofDrawRectangle(points[0] - 3.0f, 5, 5);
    for (auto next = point + 1;
      next != points.end(); point = next++){
      ofSetColor(255, 0, 0);
      ofDrawRectangle(*next - 3.0f, 5, 5);
      ofSetColor(0, 0, 0);
      ofDrawLine(*point, *next);
    }
  }
}
```

- brush.draw() を image.draw() の後に置く
  - mouseX, mouseY は現在のマウスカーソルの位置
  - brush の画像がマウスカーソルと一緒に動く
  - brush の色は初期値のグレー

グレーの正方形がマウスカーソルと一緒に動く



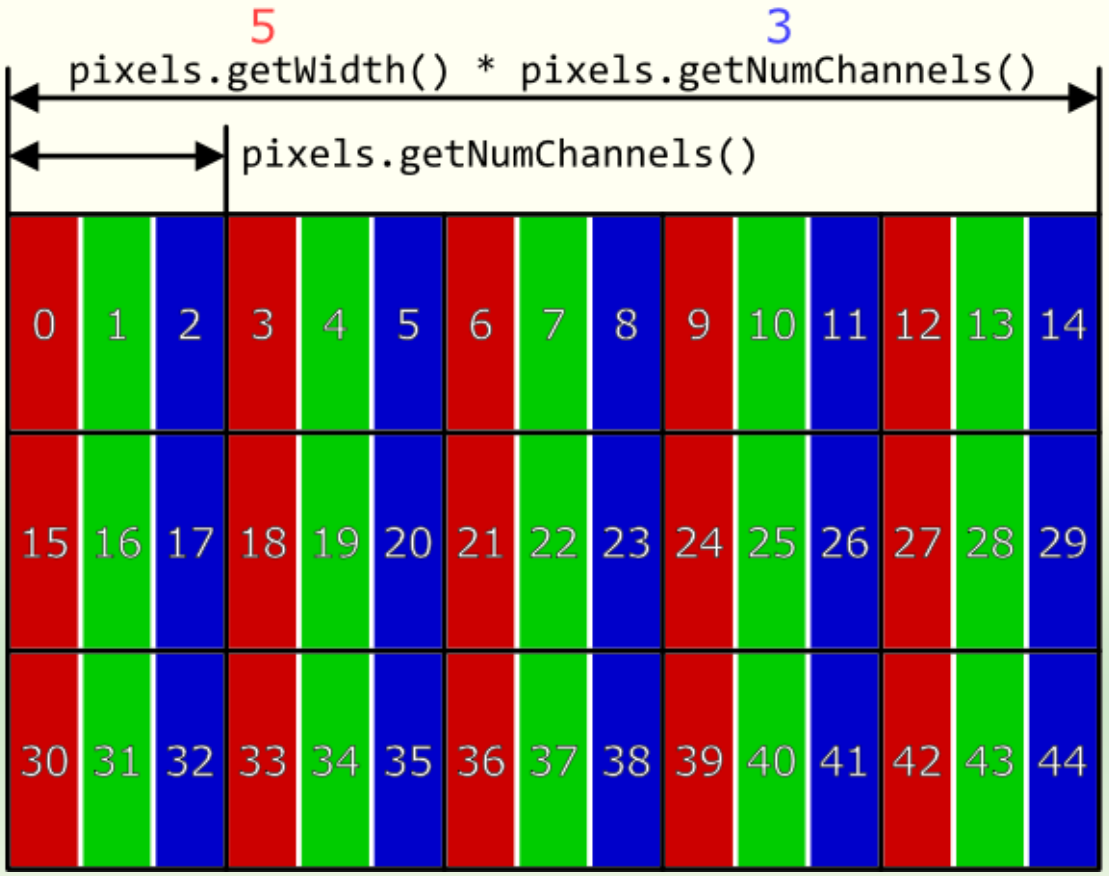
画素データを取り出す

```
//--------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
  polylines.emplace_back();
  image.load("image.jpg");
  brush.allocate(50, 50, OF_IMAGE_COLOR);
  ofPixels &pixels = brush.getPixels();
}
```

- getPixels() メソッド
  - ofImage クラスのオブジェクト（画像）の画素データを取り出す
- ofPixels クラス
  - 画像の画素データのクラス

ofPixels クラスの画素データ

```
brush.allocate(5, 3, OF_IMAGE_COLOR);
ofPixels &pixels = brush.getPixels();
```



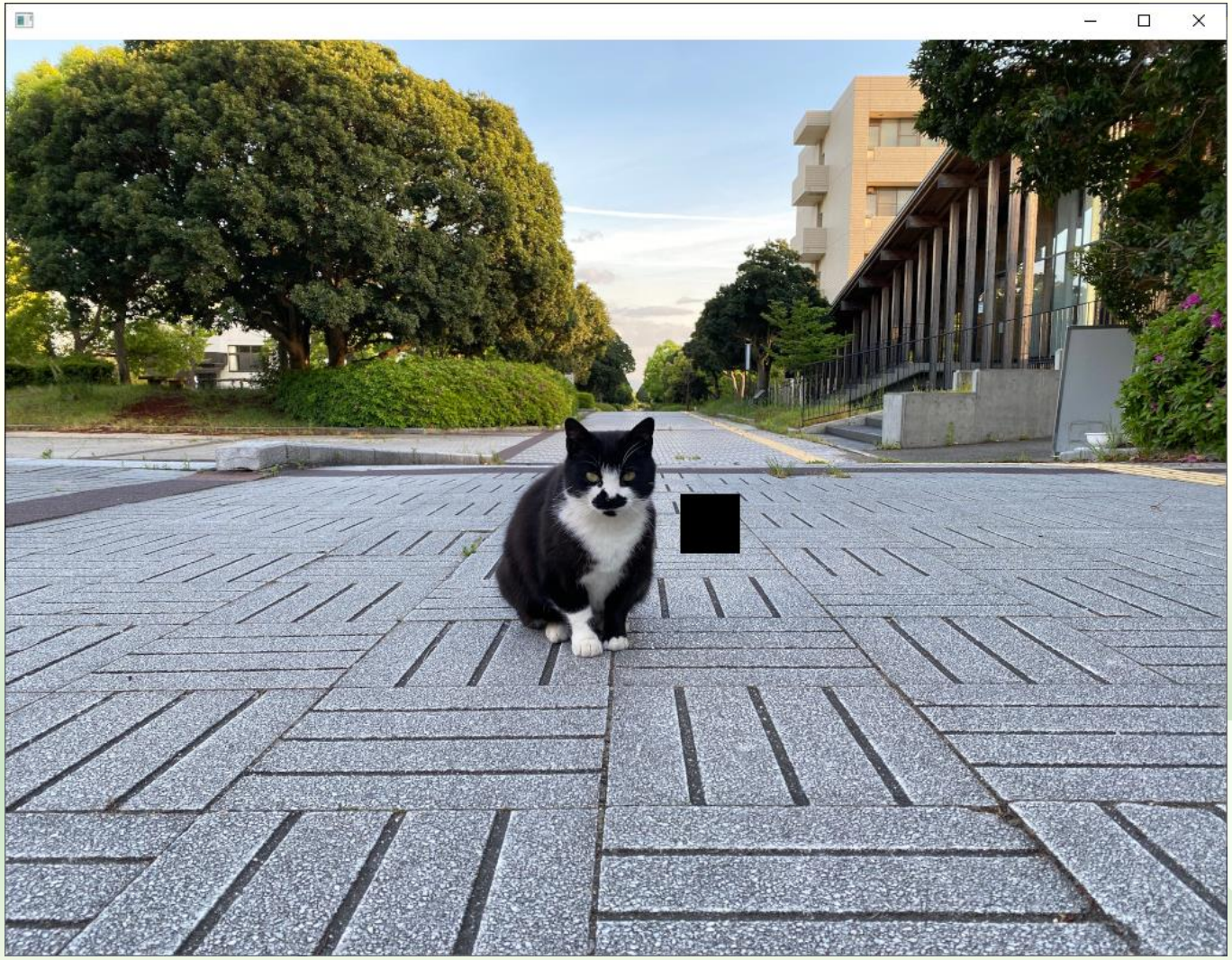
- 例えば brush.allocate(5, 3, OF_IMAGE_COLOR); で割り当てた画素データは左のようになる
  - pixels.getWidth() == 5
  - pixels.getHeight() == 3
  - pixels.getNumChannerl() == 3
  - pixels.size() == 45
- １画素分のデータは pixels の３つの要素に入っている
  - pixels[0]: 赤, pixels[1]: 緑, pixels[2]: 青, pixels[3]: 赤, pixels[4]: 緑, …

brush を黒く塗りつぶす

```
//--------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
  polylines.emplace_back();
  image.load("image.jpg");
  brush.allocate(50, 50, OF_IMAGE_COLOR);
  ofPixels &pixels = brush.getPixels();
  for (size_t i = 0; i < pixels.size(); i += 3){
    pixels[i] = 0;
    pixels[i + 1] = 0;
    pixels[i + 2] = 0;
  }
  brush.update();
}
```

- size() メソッド
  - 画素データの数を返す
  - データ型は size_t
- OF_IMAGE_COLOR の場合は3つごとに 1 画素のカラーデータ
  - それぞれ R, G, B
- pixels の各要素に値を代入すれば画素の色が変えられる
- 最後に update() メソッドを実行すれば brush に反映される

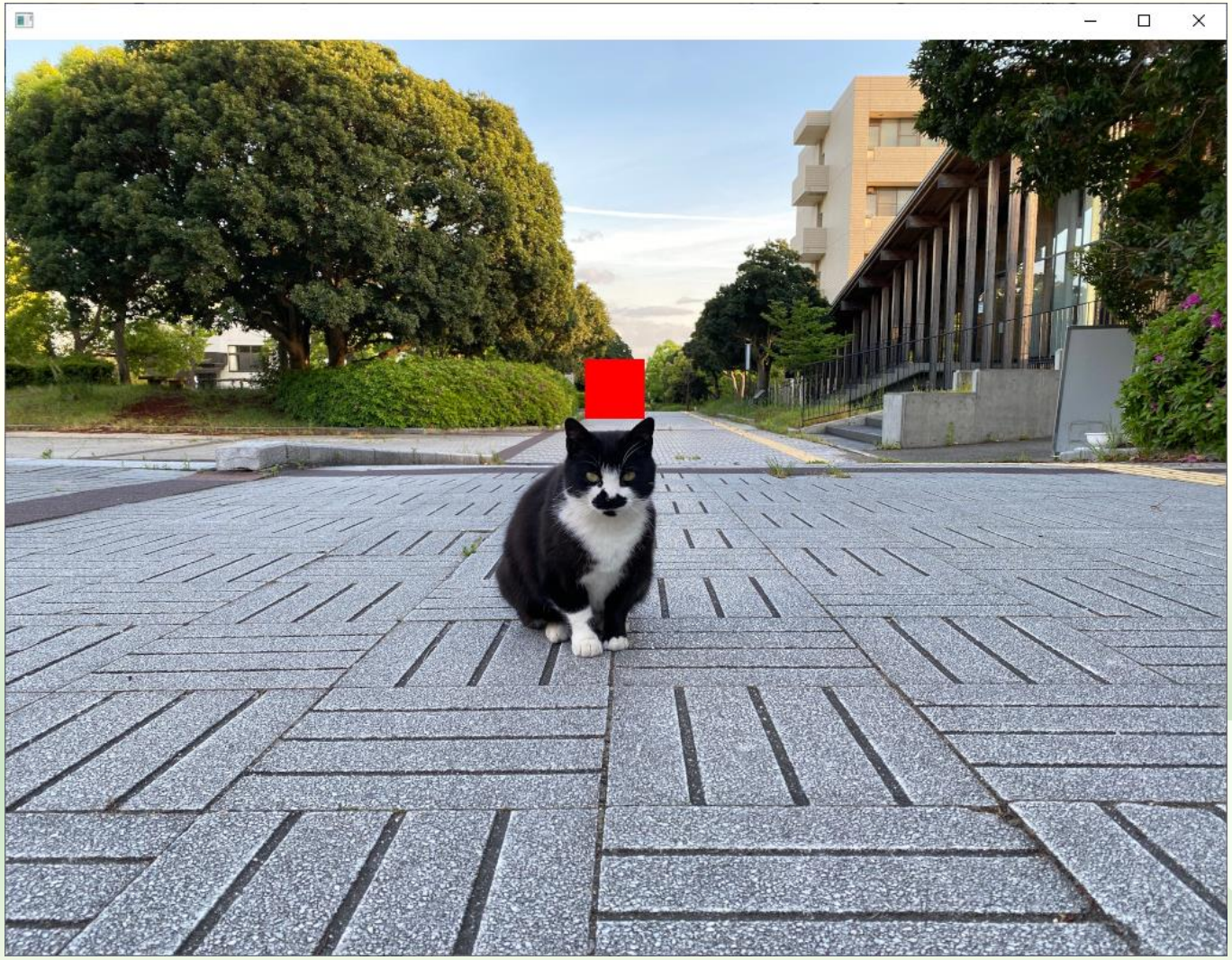
brush が黒くなる



課題４－４

赤いブラシを作る

brush を赤くしてください



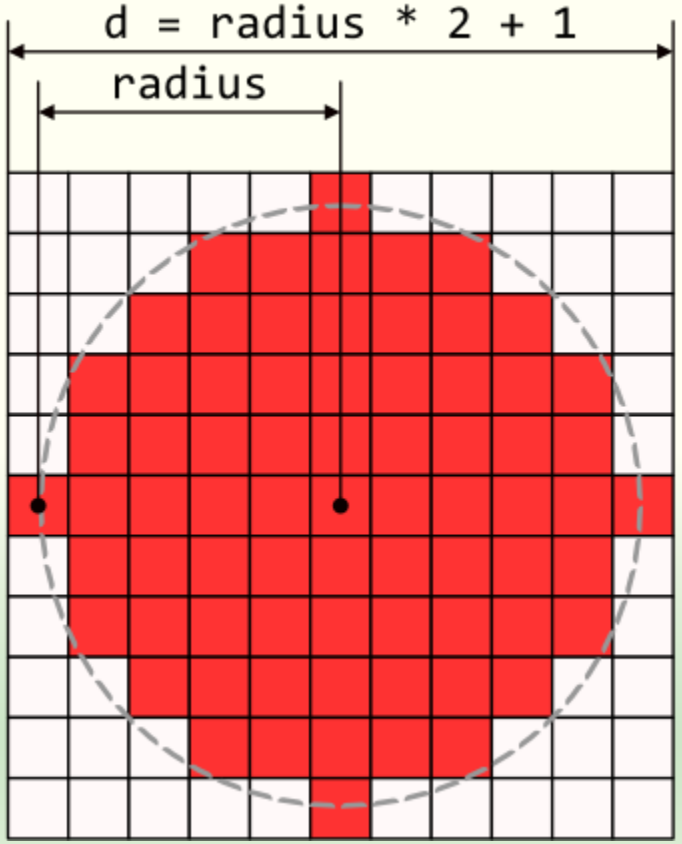
課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **4-4.png** というファイル名で保存し、Moodle の第４回課題にアップロードしてください

課題４－５

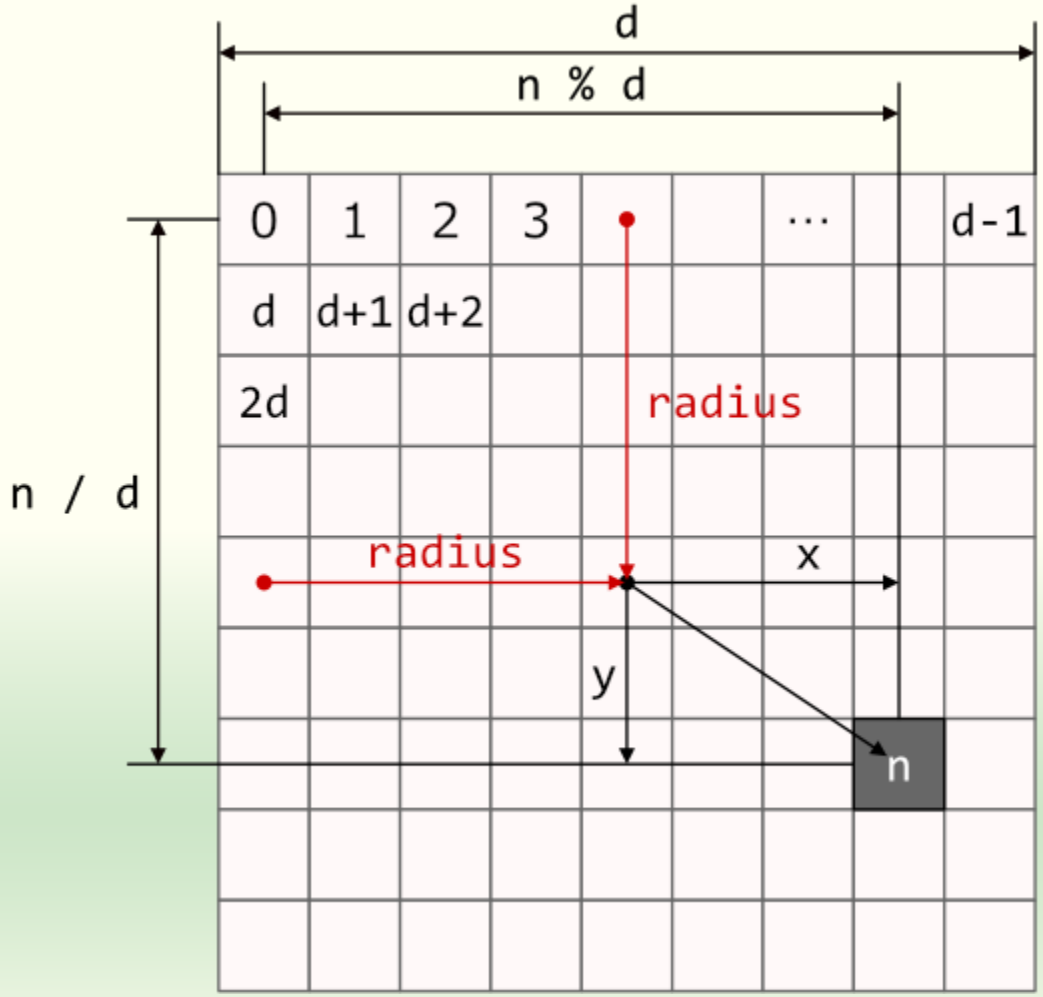
ブラシを円形にする

brush を丸くする



- 半径が radius の円を塗りつぶす
- この円がぴったり収まる領域は辺の長さ d が d = radius * 2 + 1 の正方形である

画素の番号 n と座標値 x, y



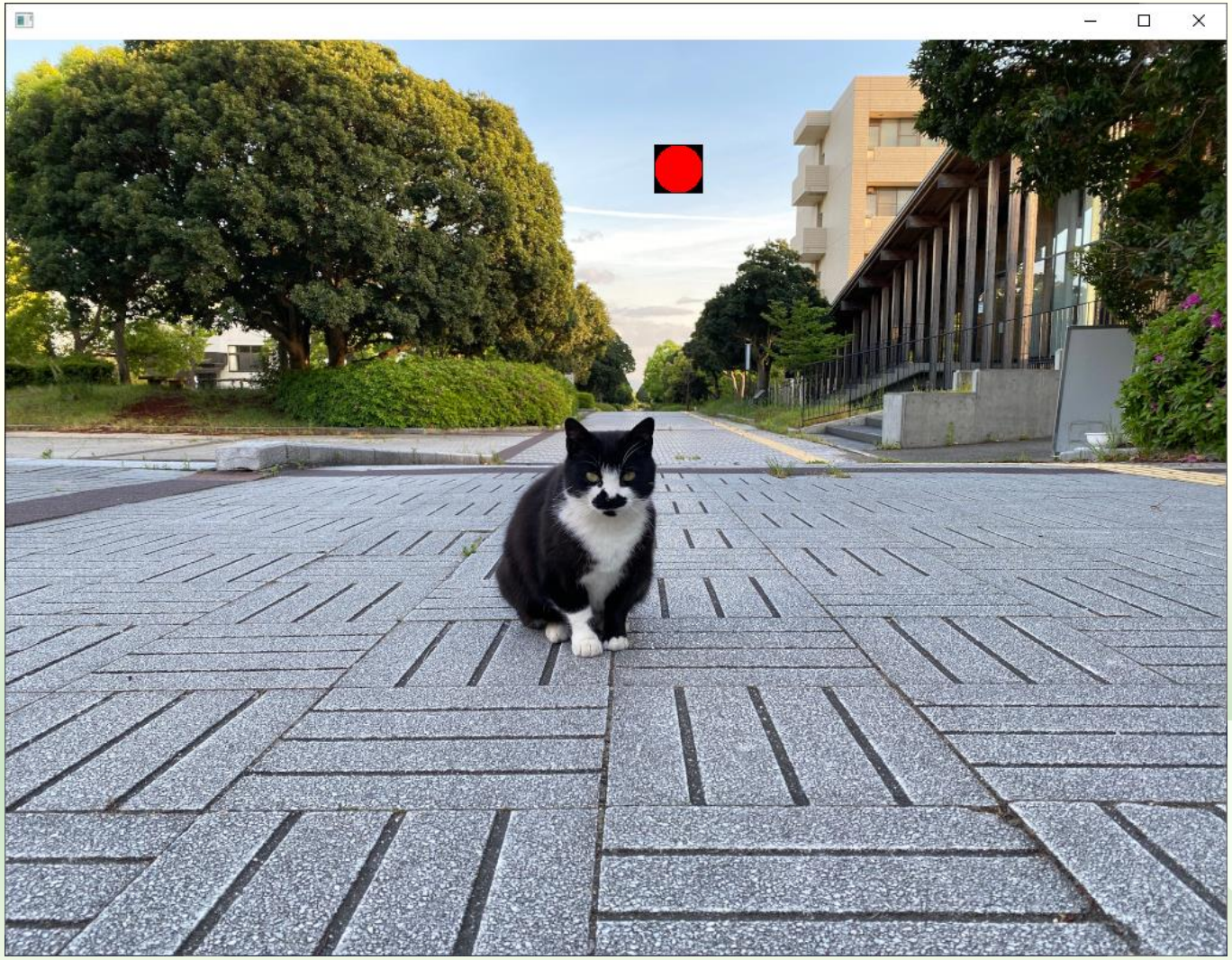
- 画像の幅（横方向の画素数）を d とする
- 画像の左上の画素を起点として、その画素の番号を 0 とする
- 番号 n の画素の画像の左端からの画素数は n % d
- 番号 n の画素の画像の上端からの画素数は n / d
- それらから半径 radius を引いて中心からの画素位置 x, y を得る

brush のサイズを求める

```
#include "ofApp.h"

const int radius = 20;

//--------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
  polylines.emplace_back();
  image.load("image.jpg");
  const int d = radius * 2 + 1;
  brush.allocate(d, d, OF_IMAGE_COLOR);
  ofPixels &pixels = brush.getPixels();
  for (size_t i = 0; i < pixels.size(); i += 3){
    const int n = i / 3;
    const int x = n % d - radius;
    const int y = n / d - radius;
  (途中略)
  }
}
```

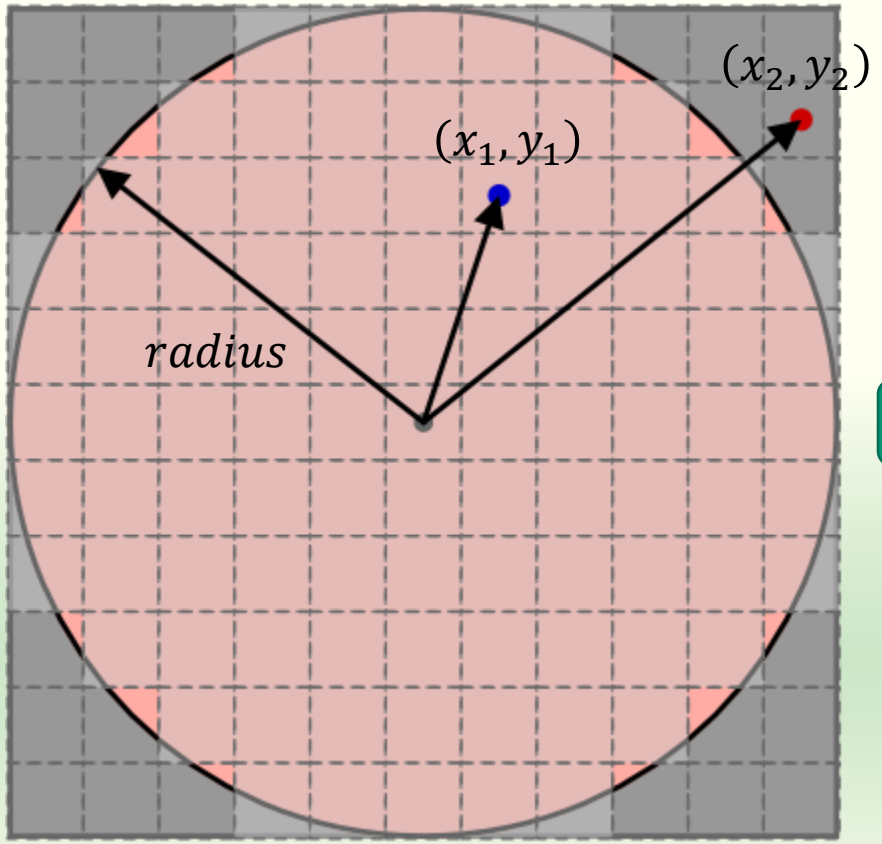
- const int n = i / 3;
  - i は 1 画素ごとに 3 進むから 3 で割れば画素の番号 n が得られる
- const int x = n % d - radius;
  - n を領域の幅 d で割った剰余は画素の領域の左端からの位置となる
  - これから radius を引けば領域の中心からの x 座標値が得られる
- const int y = n / d - radius;
  - n を領域の幅 d で割った商の整数部は画素の領域の上端からの位置となる
  - これから radius を引けば領域の中心からの y 座標値が得られる

brush を円形にしてください



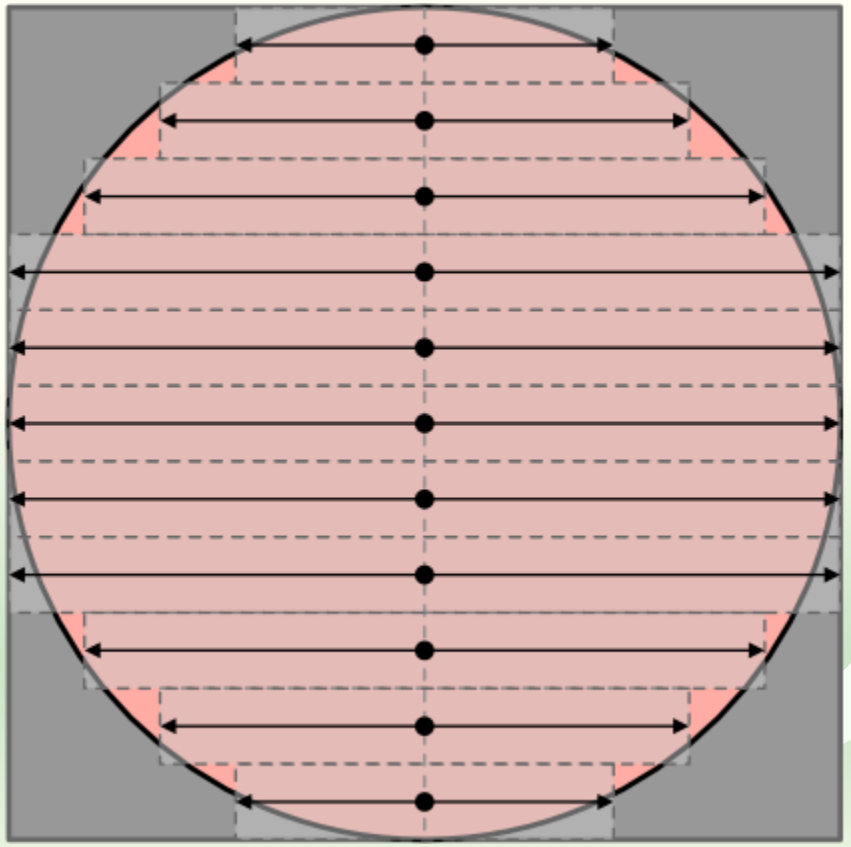
円の内部にある画素に色を付ける

画素が円の内部にあるか判定する



やり方は色々

画素の高さごとに幅を求める



マウスカーソルの先に円の中心を置く

```
//--------------------------------------------
void ofApp::draw(){
  ofSetColor(255, 255, 255);
  image.draw(0, 0);
  brush.draw(mouseX - radius, mouseY - radius);
  for (auto &points : polylines){
    if (points.empty()) return;
    auto point = points.begin();
    ofSetColor(255, 0, 0);
    ofDrawRectangle(points[0] - 3.0f, 5, 5);
    for (auto next = point + 1;
      next != points.end(); point = next++){
      ofSetColor(255, 0, 0);
      ofDrawRectangle(*next - 3.0f, 5, 5);
      ofSetColor(0, 0, 0);
      ofDrawLine(*point, *next);
    }
  }
}
```

- マウスカーソルの位置から半径 radius を引く

課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **4-5.png** というファイル名で保存し、Moodle の第４回課題にアップロードしてください

課題４－６

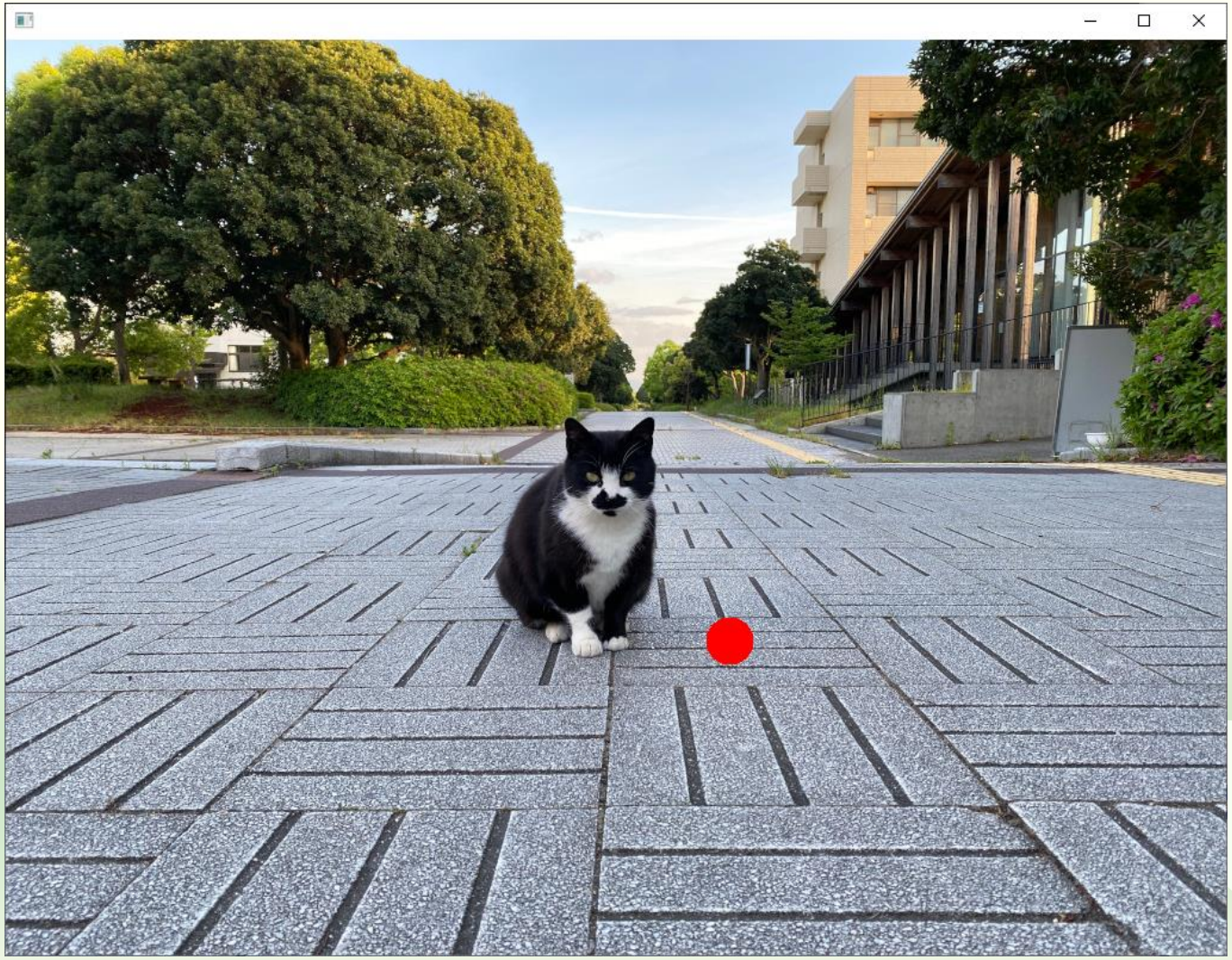
ブラシの範囲外を透明にする

円の範囲外を透明にする

```
//--------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
  polylines.emplace_back();
  image.load("image.jpg");
  const int d = radius * 2 + 1;
  brush.allocate(d, d, OF_IMAGE_COLOR_ALPHA);
  ofPixels &pixels = brush.getPixels();
  for (size_t i = 0; i < pixels.size(); i += 4){
    const int t = i / 4;
    const int x = t % d - radius;
    const int y = t / d - radius;
 (途中略)
  }
}
```

- allocate() メソッドの imageType として OF_IMAGE_COLOR の代わりに **OF_IMAGE_COLOR_ALPHA** を用いる
  - アルファチャンネルを設ける
- １画素当たり４チャネル用いる
  - 円の範囲外ではアルファチャネル（第４チャネル）を 0 にする
  - 円の範囲内ではアルファチャネルを 255 にする

ブラシの範囲外を透明にする



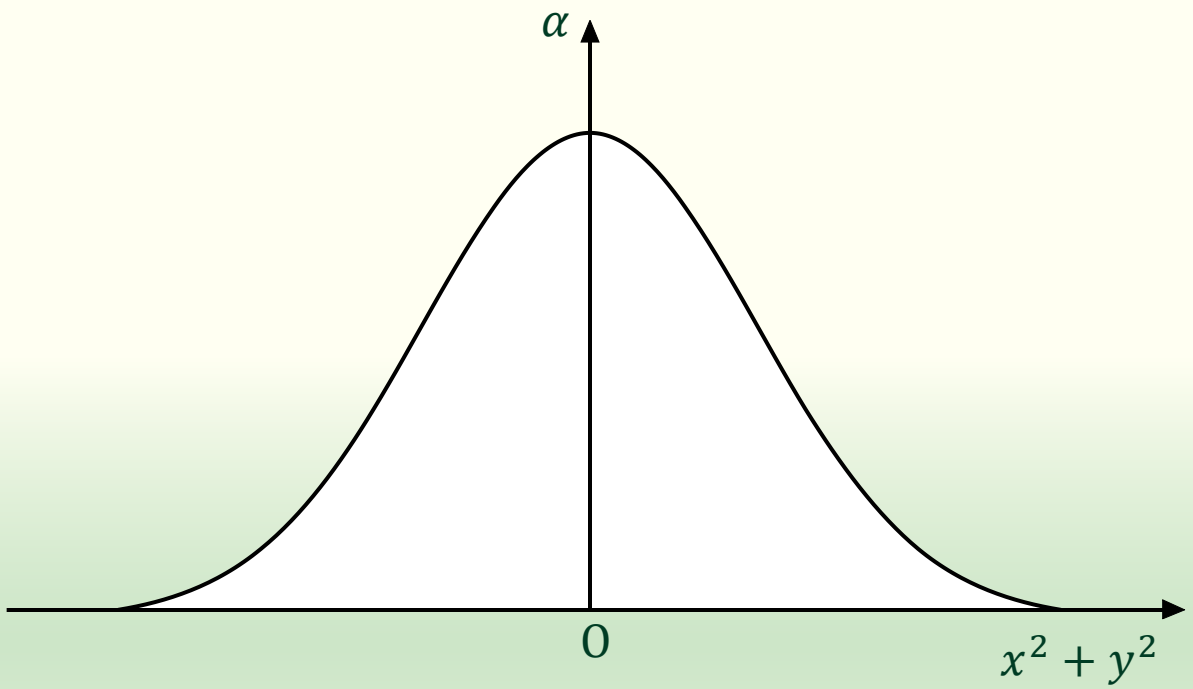
課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **4-6.png** というファイル名で保存し、Moodle の第４回課題にアップロードしてください

課題４－７

ブラシの中心から円周に向かって透明にする

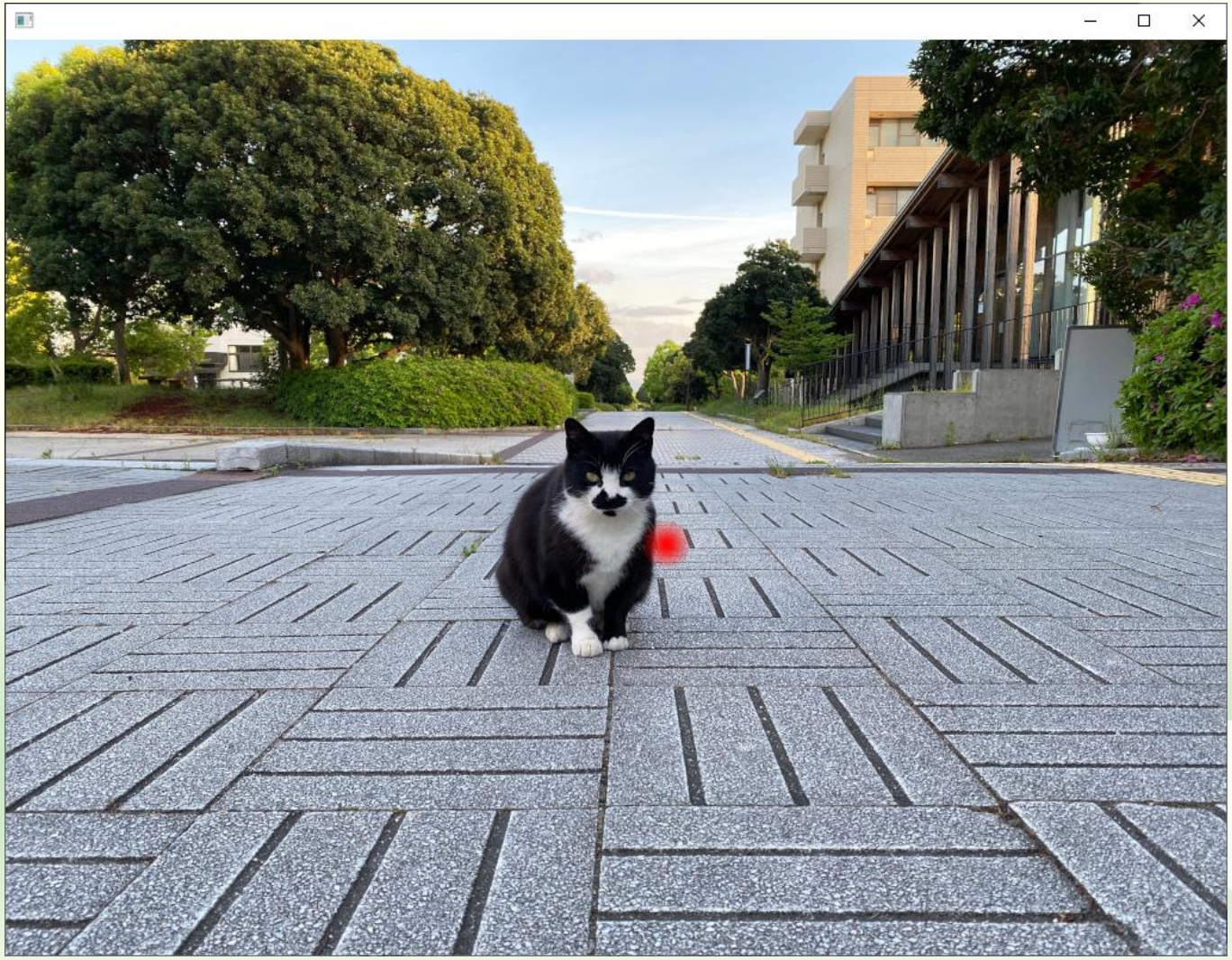
ブラシの中心から円周に向かって透明にする



- 中心から離れるにしたがって値が減少する関数をアルファ値に用いる
  - 正規分布など

$$\alpha = 255 \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

ブラシを中心から円周に向かって透明にする



課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **4-7.png** というファイル名で保存し、Moodle の第４回課題にアップロードしてください

課題４－８

ブラシを読み込んだ画像に書き込む

読み込んだ画像にアルファチャネルを追加する

```
#include "ofApp.h"

const int radius = 20;

//--------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
  polylines.emplace_back();
  image.load("image.jpg");
  image.setImageType(OF_IMAGE_COLOR_ALPHA);
  const int d = radius * 2 + 1;
  brush.allocate(d, d, OF_IMAGE_COLOR);
  ofPixels &pixels = brush.getPixels();
  for (size_t i = 0; i < pixels.size(); i += 3){
    const int n = i / 3;
    const int x = n % d - radius;
    const int y = n / d - radius;
  (途中略)
  }
}
```

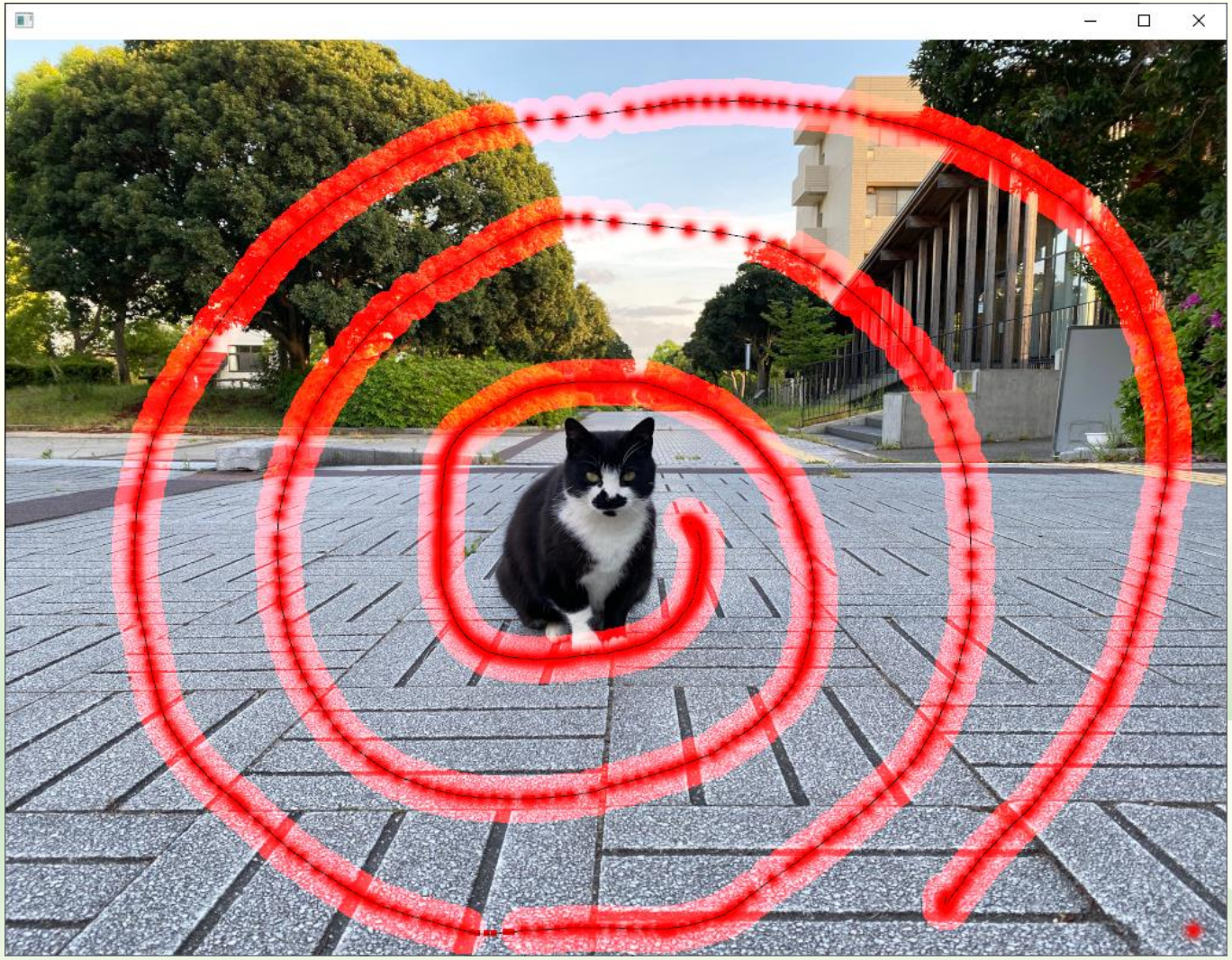
```
//--------------------------------------------
#include <commdlg.h>

void ofApp::keyPressed(int key){
  if (key == 'o' || key == 'O'){
  (途中略)

    if (GetOpenFileName(&ofn)){
      if (!image.load(filePath)) {
        MessageBox(… 途中略 …);
      }
      else{
        image.setImageType(OF_IMAGE_COLOR_ALPHA);
      }
    }
  }
}
```

brush を image にブレンド

```
//--------------------------------------------------
void ofApp::mouseDragged(int x, int y, int button){
  if (button == 0){
    auto &points = polylines.back();
    points.emplace_back(vec2{ x, y });
    brush.getPixels().blendInto(
      image.getPixels(), x - radius, y - radius);
    image.update();
  }
}

//--------------------------------------------------
void ofApp::mousePressed(int x, int y, int button){
  if (button == 0){
    auto &points = polylines.back();
    points.emplace_back(vec2{ x, y });
    brush.getPixels().blendInto(
      image.getPixels(), x - radius, y - radius);
    image.update();
  }
}
```

- blendInTo() メソッド
  - bool ofPixels_::blendInto(ofPixels_ &dst, size_t x, size_t y)
  - 画像を dst していした画像の x, y の位置とブレンドする

ブラシをつなげるところまで行かなかった…



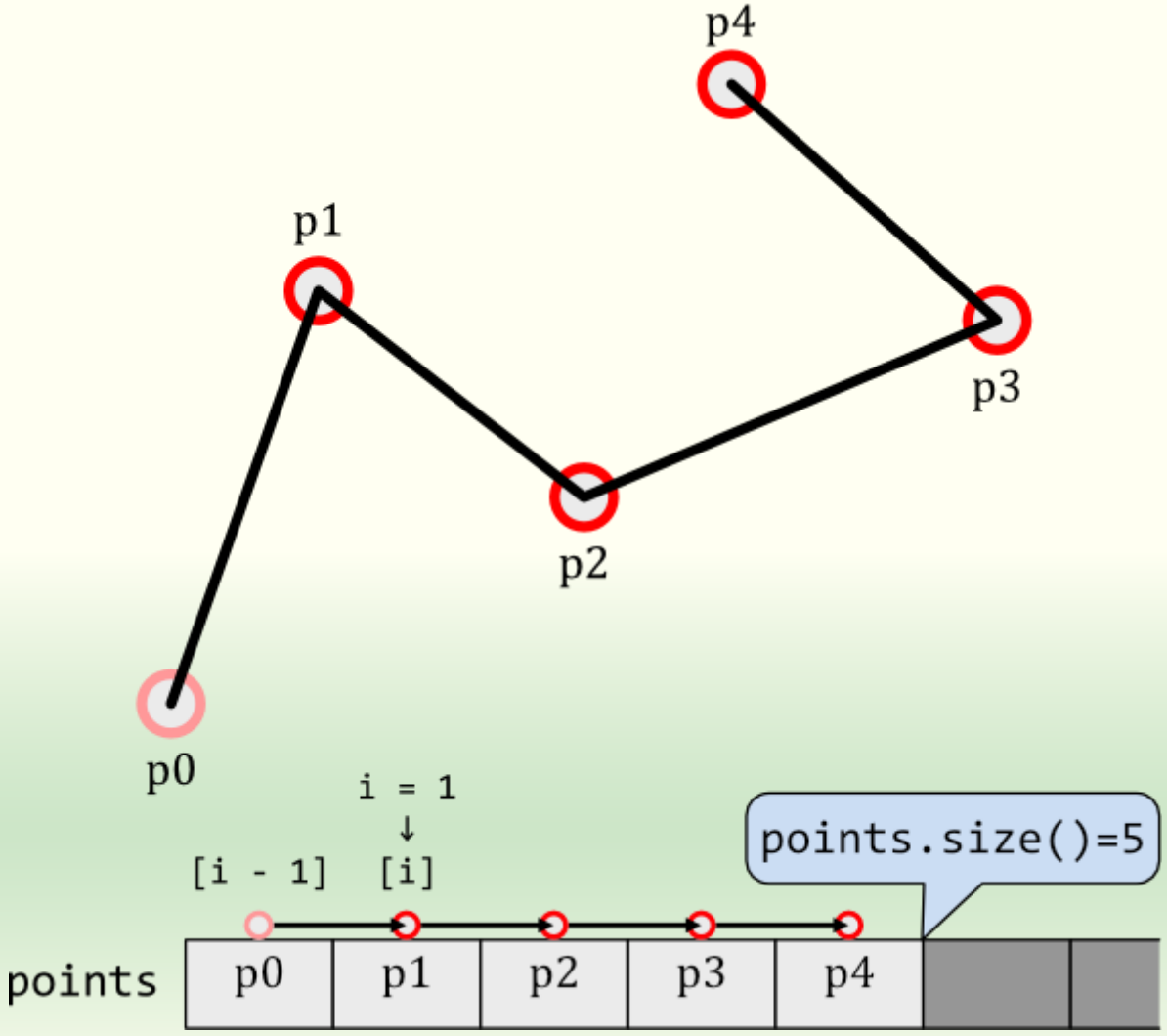
課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **4-8.png** というファイル名で保存し、Moodle の第４回課題にアップロードしてください
- ソースプログラム **ofApp.h** と **ofApp.cpp** を Moodle の第４回課題にアップロードしてください

補足

添え字を使ったデータの取り出し

クリックしたところを線分で結ぶ



- クリックした位置は points という vector に p0 → p1 → p2 → … という順に入っている
- p1 から 1 つ前の点（p1 に対しては p0）との間に線分を引く
- p2, p3, … と進めて最後の点で終わる
- 点の数は points.size() で調べられる

２つずつ点を結ぶ

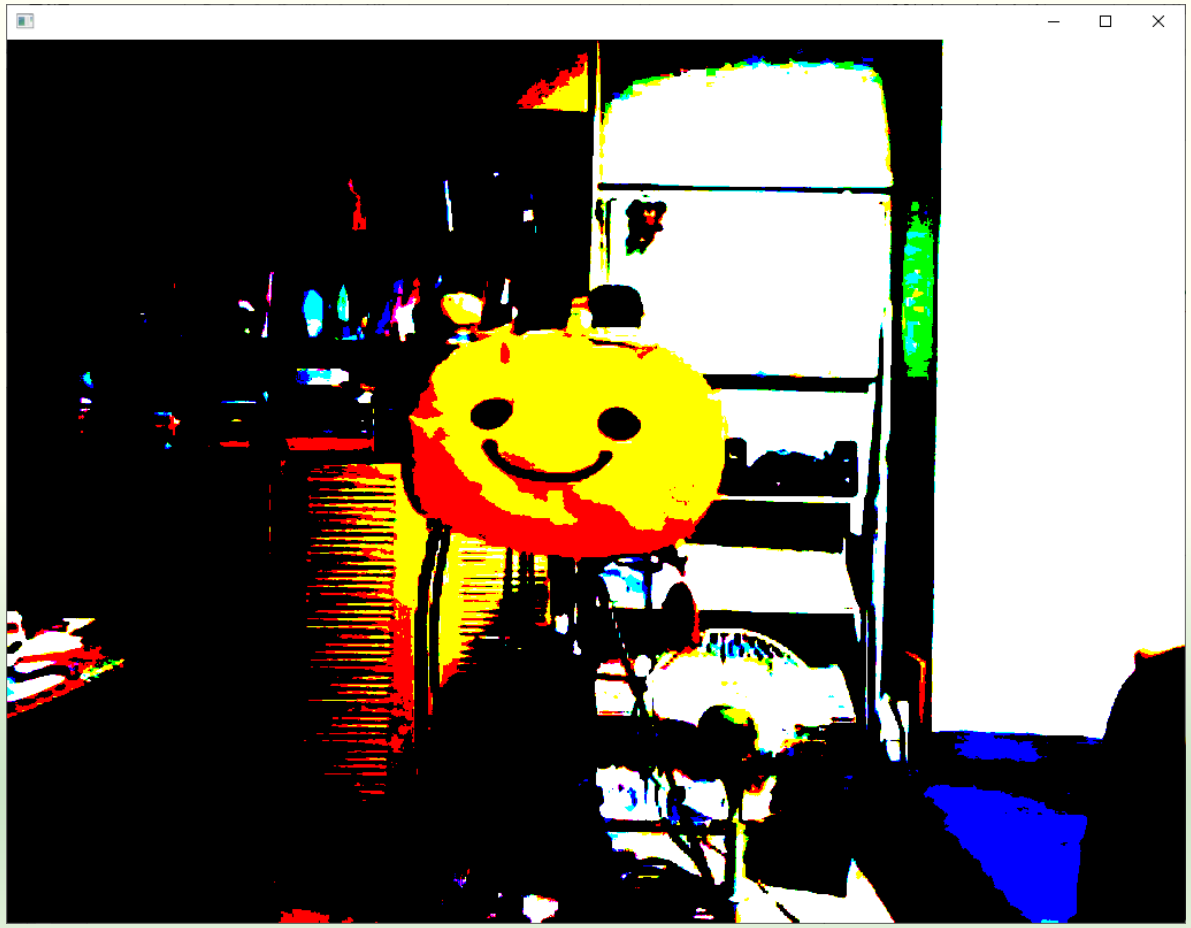
```
#include "ofApp.h"

//--------------------------------------------------
void ofApp::setup(){
  ofBackground(255, 255, 255);
}

//--------------------------------------------------
void ofApp::update(){

}

//--------------------------------------------------
void ofApp::draw(){
  for (size_t i = 1; i < points.size(); ++i){
    ofSetColor(0, 0, 0);
    ofDrawLine(points[i - 1], points[i]);
  }
}
```

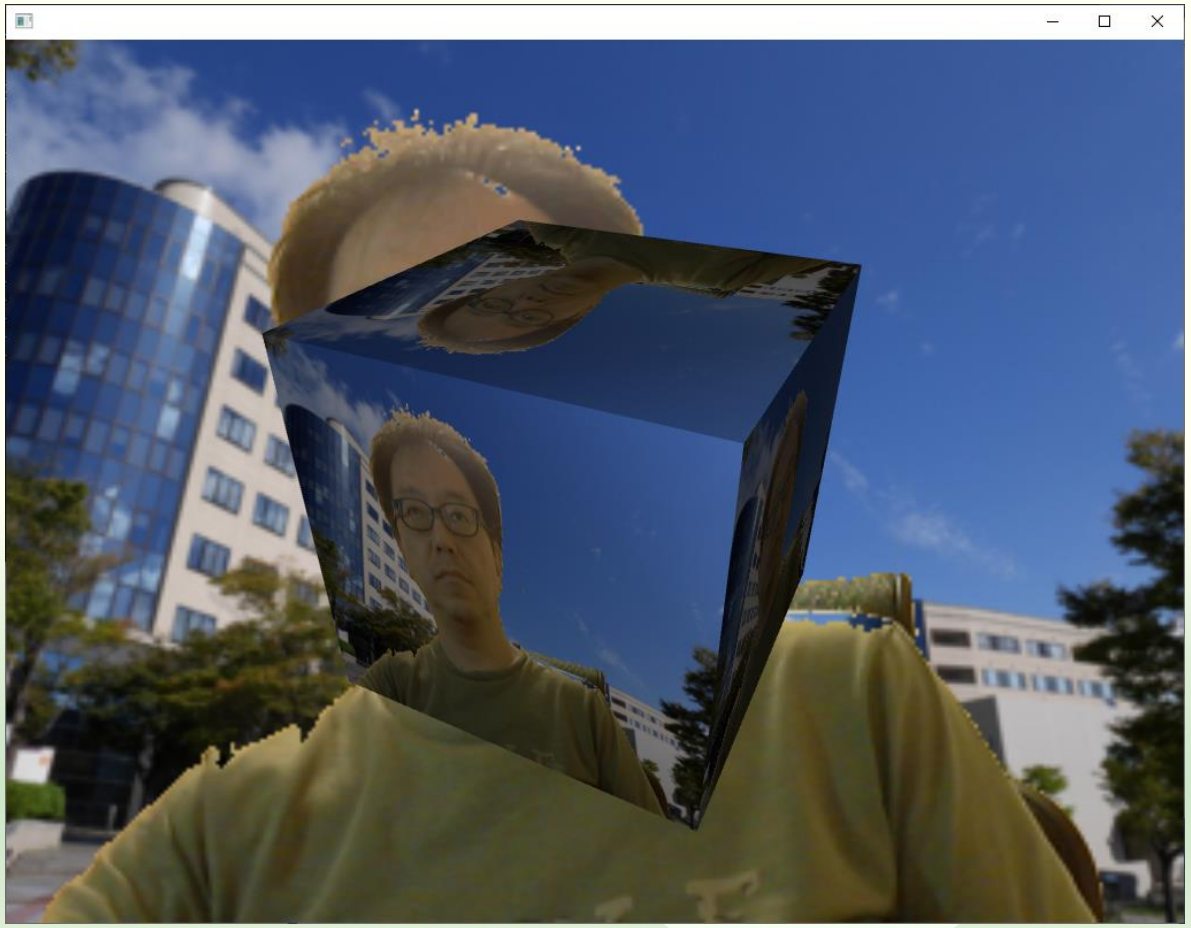
- i = 1（p1）から始める
- i が点の数 points.size() に達していなければ（i < points.size()）以降の処理を行う
- １つ前の点 i - 1 = 0（p0）から現在の点 i = 1（p1）との間に線分を引く
- ++i により i を 2, 3, … と増やしながら繰り返す（p2, p3, … と進める）

メディアプログラミング演習

第5回

本日はビデオを使ったアプリの作成



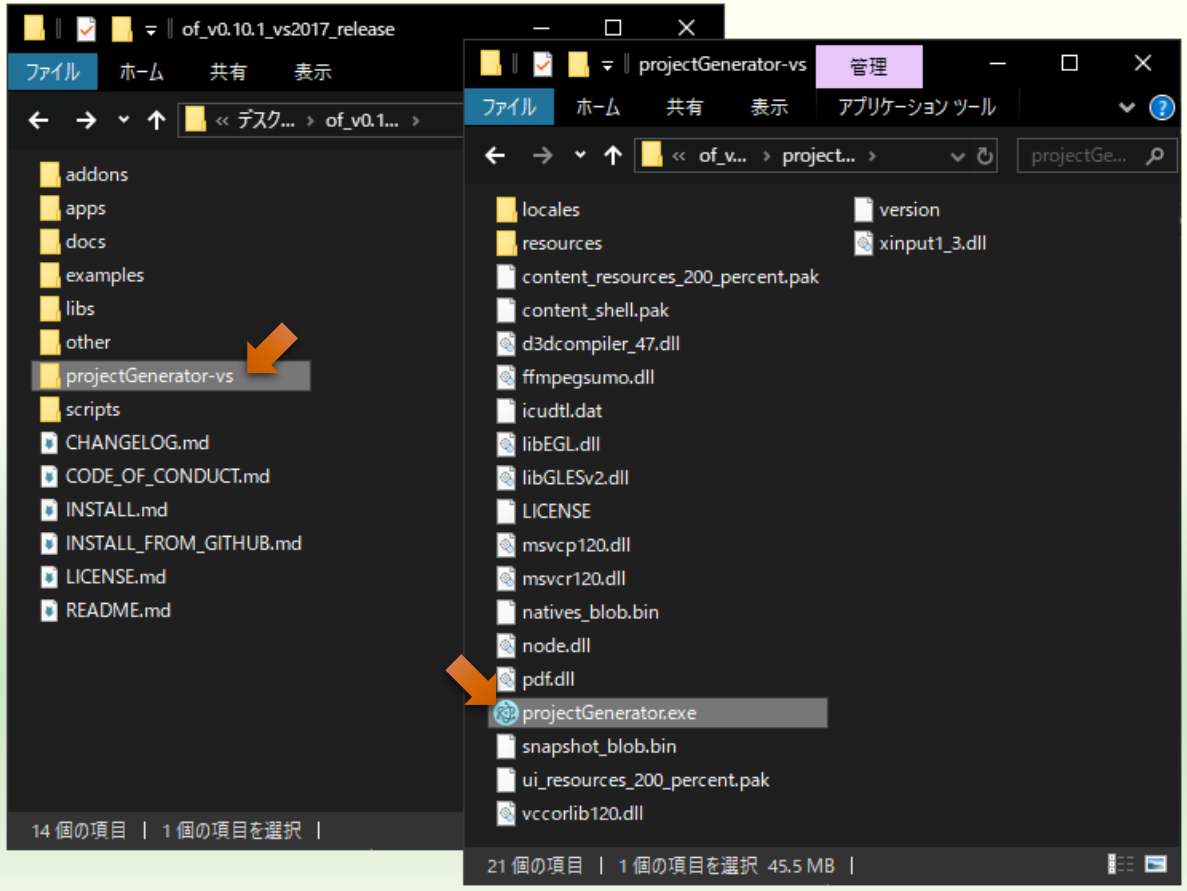


準備

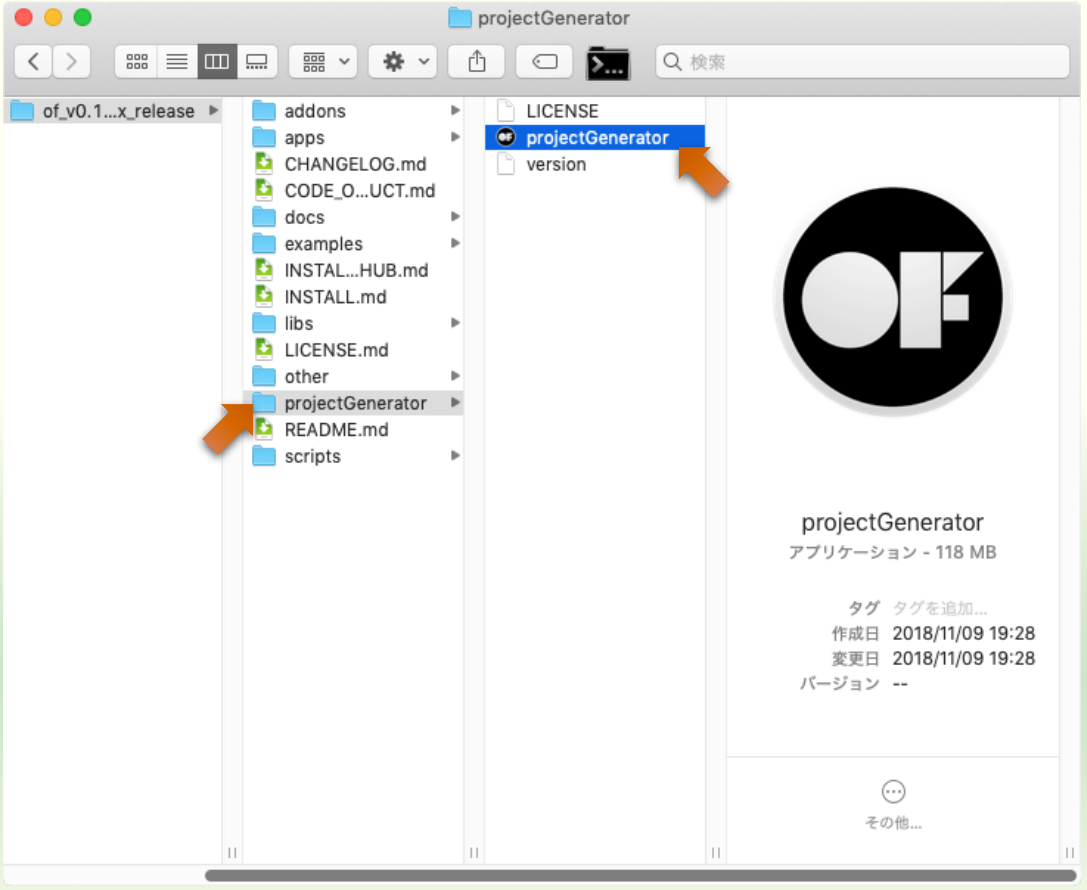
プロジェクトの作成

projectGenerator を起動する

windows 版のパッケージ



macOS 版のパッケージ

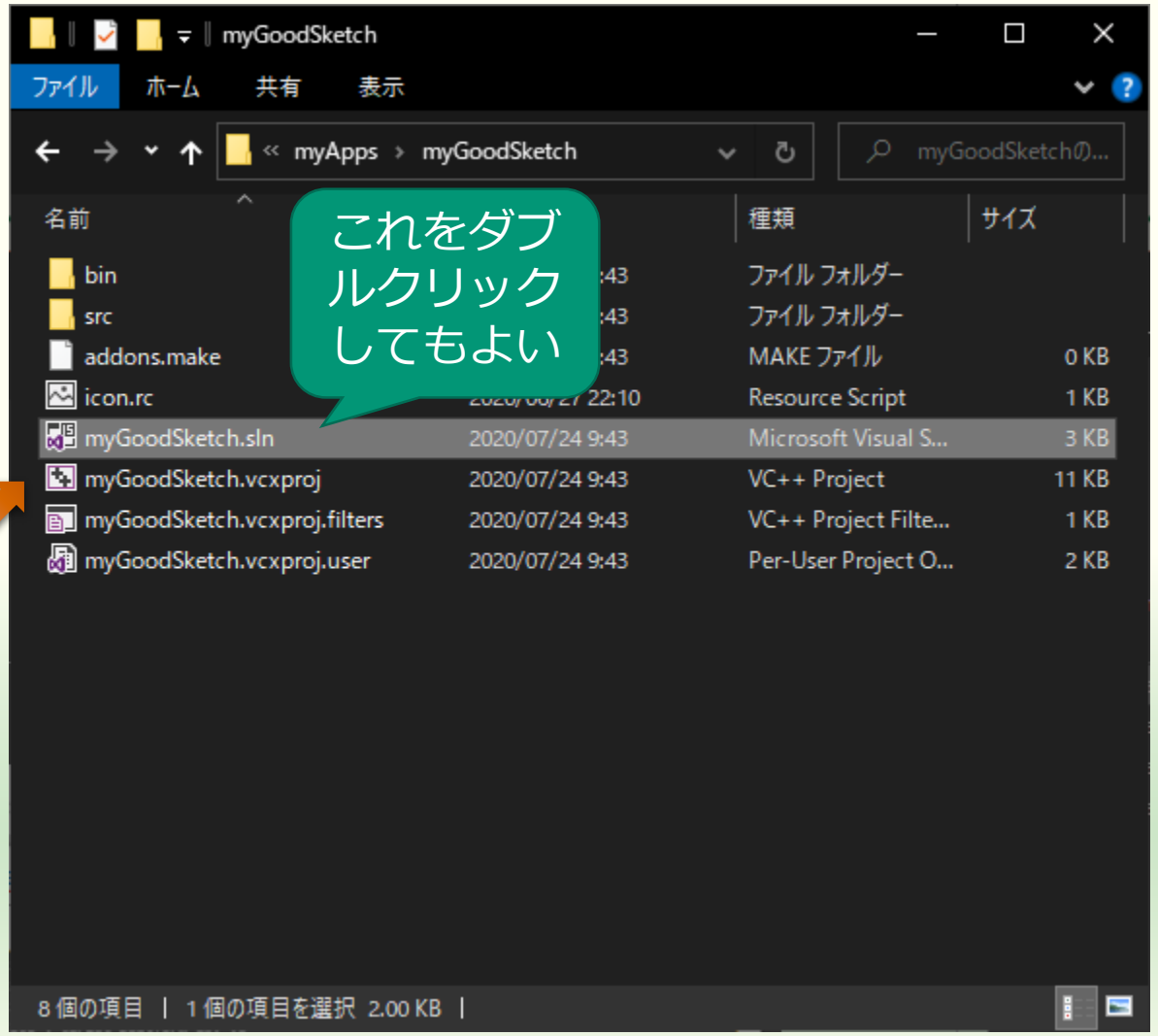


空のプロジェクトの作成

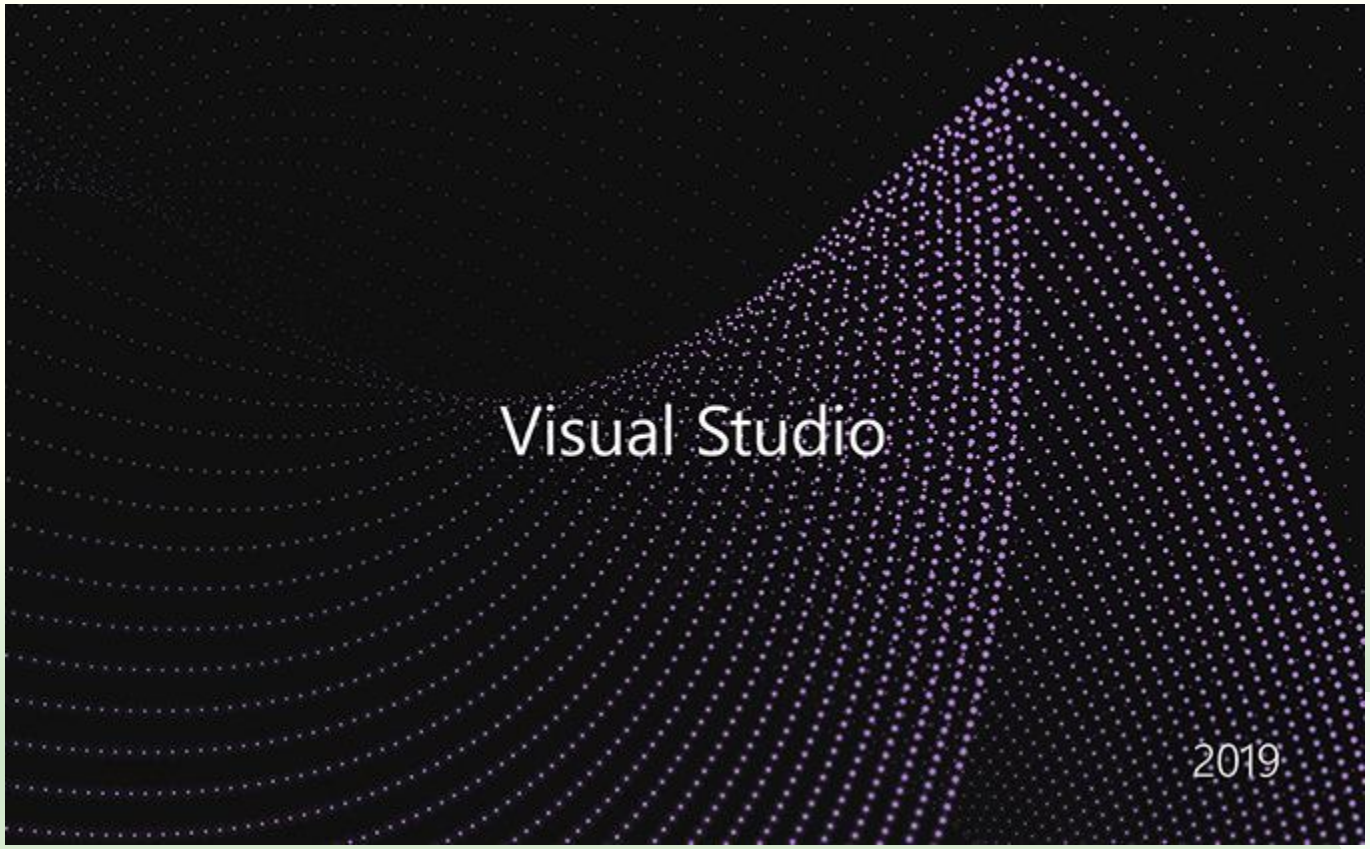
- Project name:
  - 作成するプロジェクト（プログラム）の名前
- Project path:
  - 作成するプロジェクトのファイルを置く場所
  - openFrameworks のパッケージを展開した場所の中の apps¥myApps

プロジェクトの作成成功

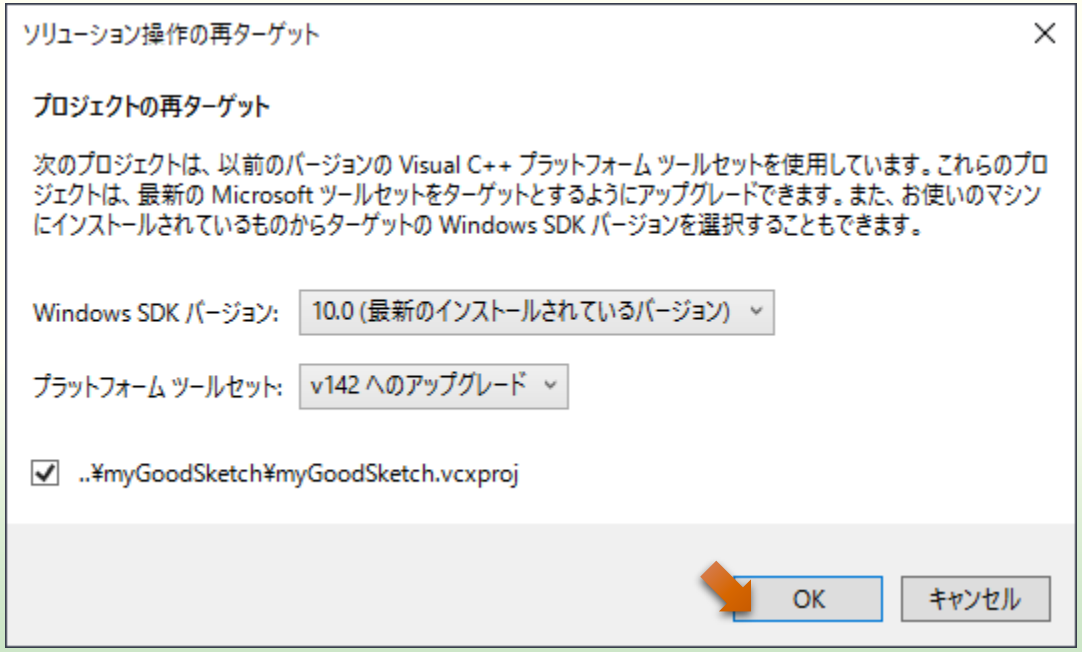
クリックすると開く



Visual Studio 2019 が起動する

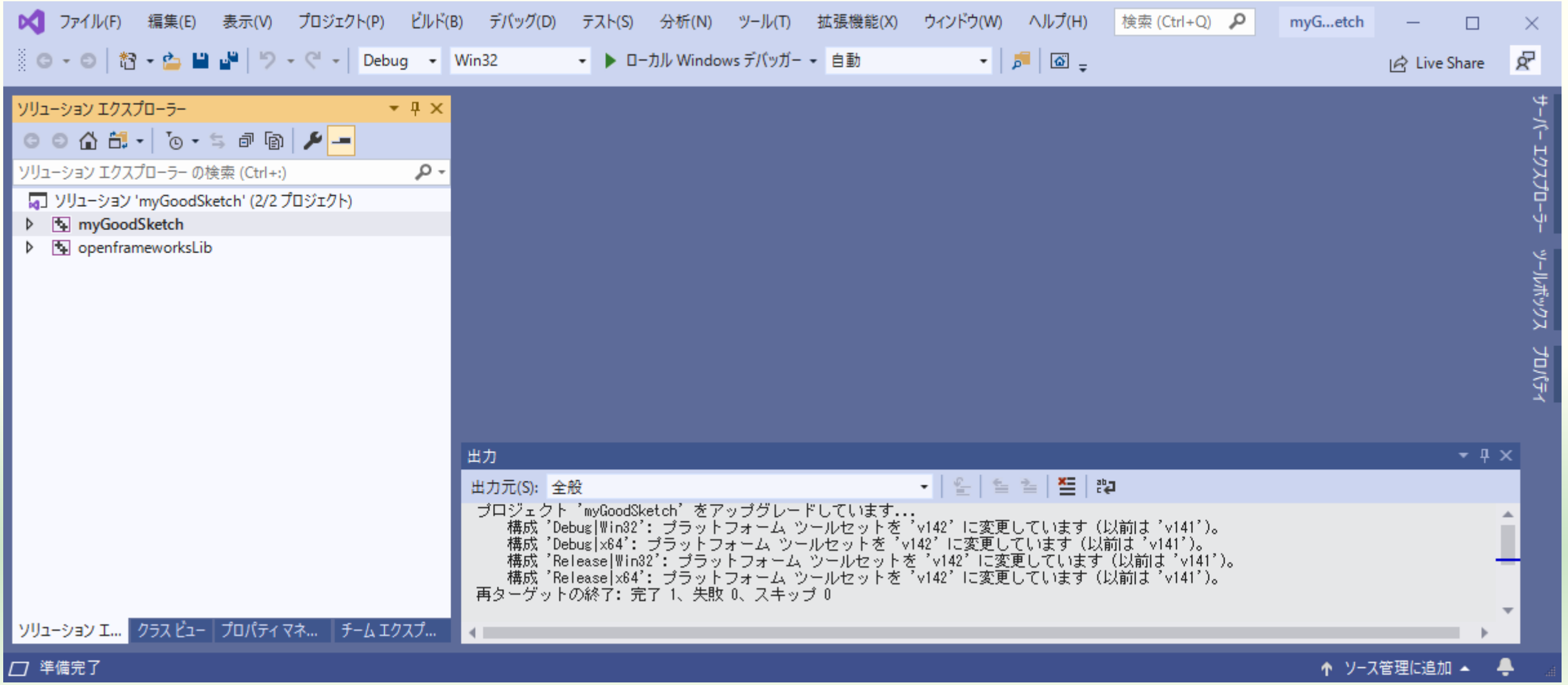


ソリューションの再ターゲット



Visual Studio は頻繁に更新しているので皆さんがお使いの Visual Studio SDK のバージョンと合わない場合がある

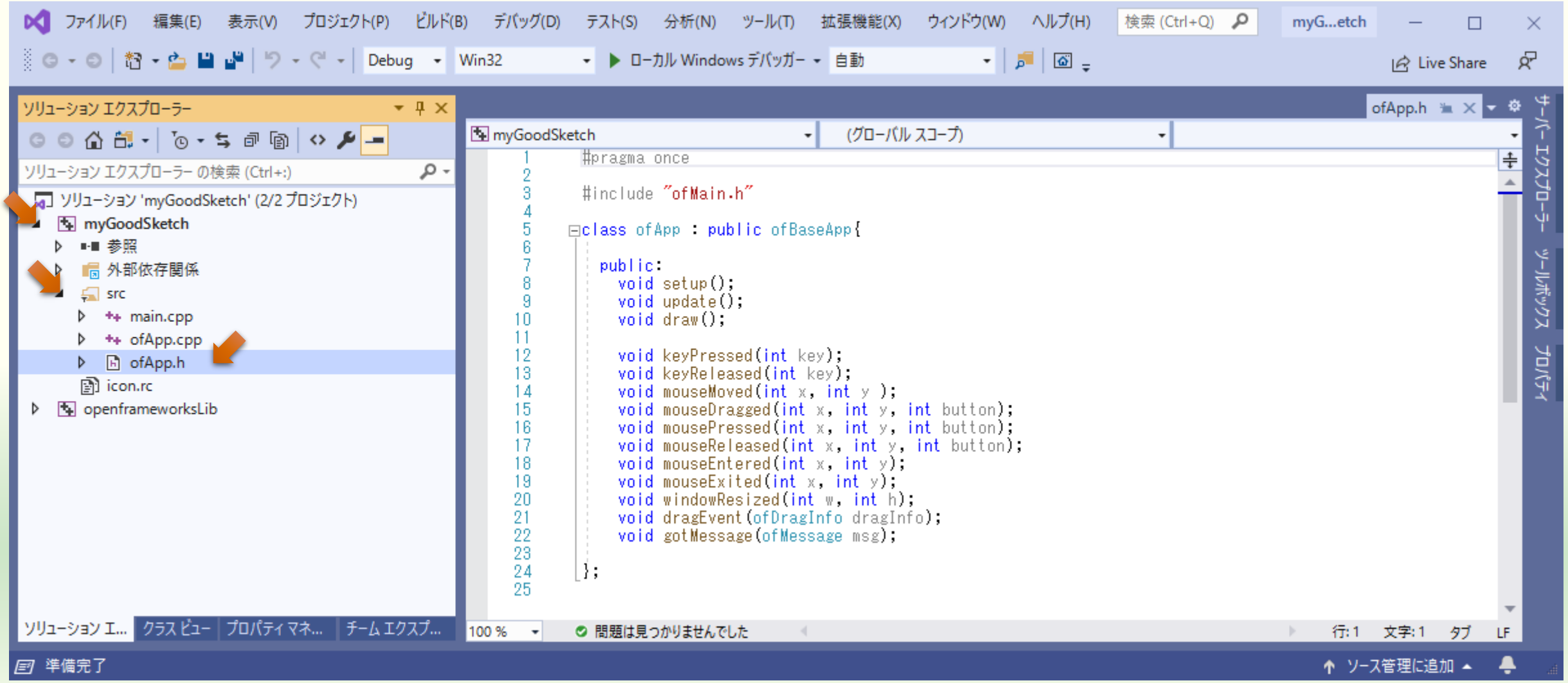
Visual Studio 起動



映像の入力

ビデオデータの取り扱い

ofApp.h を開く

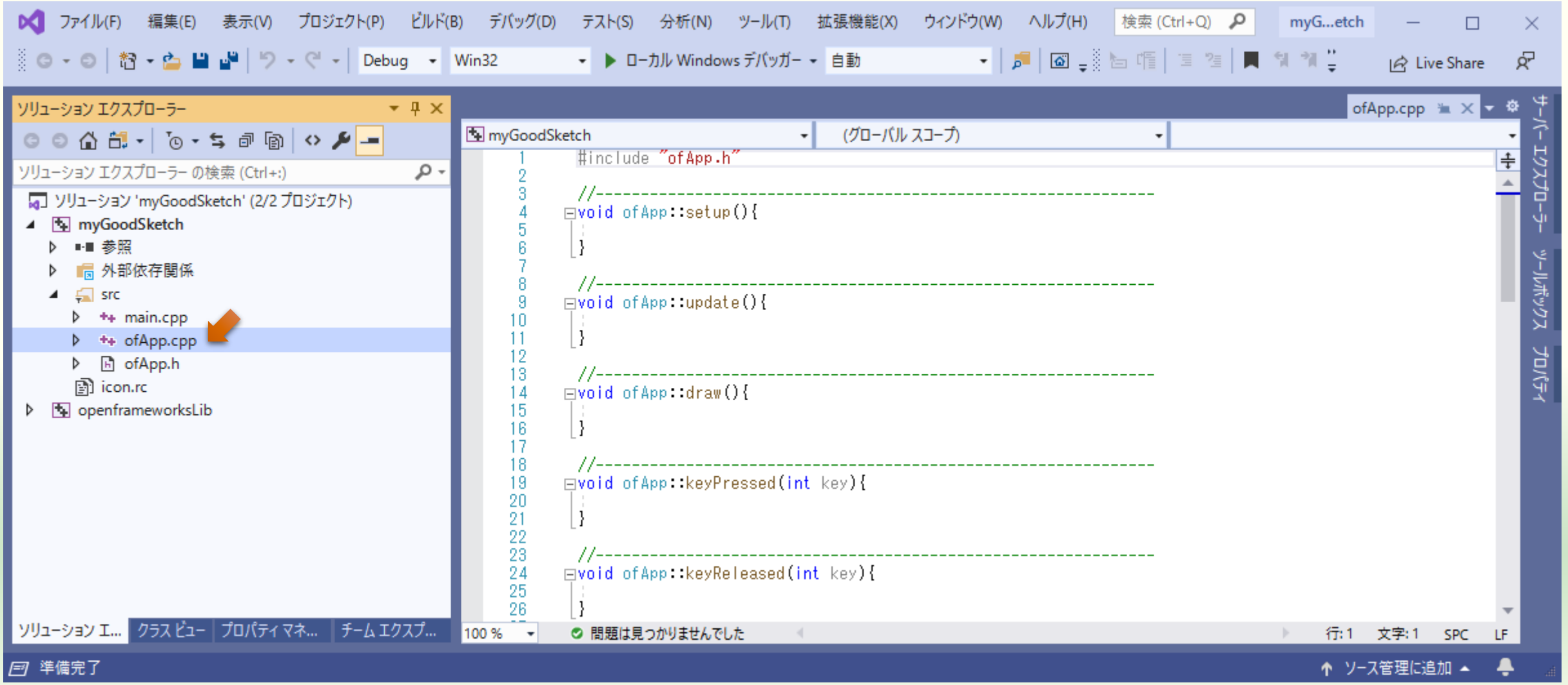

ファイル(F)
編集(E)
表示(V)
プロジェクト(P)
ビルド(B)
デバッグ(D)
テスト(S)
分析(N)
ツール(T)
拡張機能(X)
ウィンドウ(W)
ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myGoodSketch' (2/2 プロジェクト)
myGoodSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.h
myGoodSketch
(グローバル スコープ)
#pragma once
#include "ofMain.h"
class ofApp : public ofBaseApp{
public:
void setup();
void update();
void draw();
void keyPressed(int key);
void keyReleased(int key);
void mouseMoved(int x, int y);
void mouseDragged(int x, int y, int button);
void mousePressed(int x, int y, int button);
void mouseReleased(int x, int y, int button);
void mouseEntered(int x, int y);
void mouseExited(int x, int y);
void windowResized(int w, int h);
void dragEvent(ofDragInfo dragInfo);
void gotMessage(ofMessage msg);
};
100 %
問題は見つかりませんでした
行: 1
文字: 1
タブ
LF
ソリューション エ...
クラス ビュー
プロパティ マネ...
チーム エクスプ...
サーバー エクスプローラー
ツールボックス
プロパティ
準備完了
ソース管理に追加

ofAppクラスに映像入力のメンバ変数を追加する

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{
  ofVideoGrabber video;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  （以下略）
```

- ofVideoGrabber
  - カメラからの映像入力（ビデオキャプチャ）を行うクラス

ofApp.cpp を開く

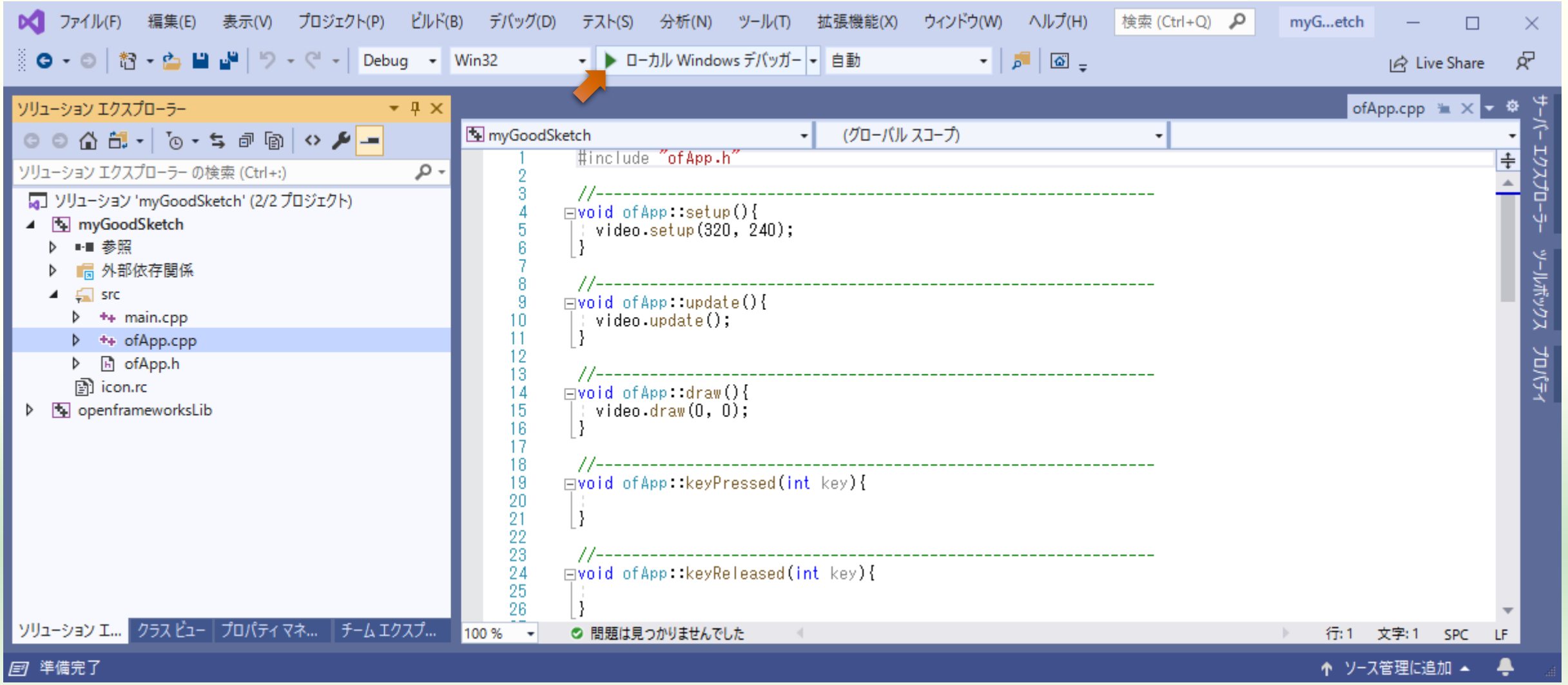


ofApp.cpp で入力した映像を表示する

```
#include "ofApp.h"

//--------------------------------------------------
void ofApp::setup(){
  video.setup(320, 240);
}

//--------------------------------------------------
void ofApp::update(){
  video.update();
}

//--------------------------------------------------
void ofApp::draw(){
  video.draw(0, 0);
}

（以下略）
```

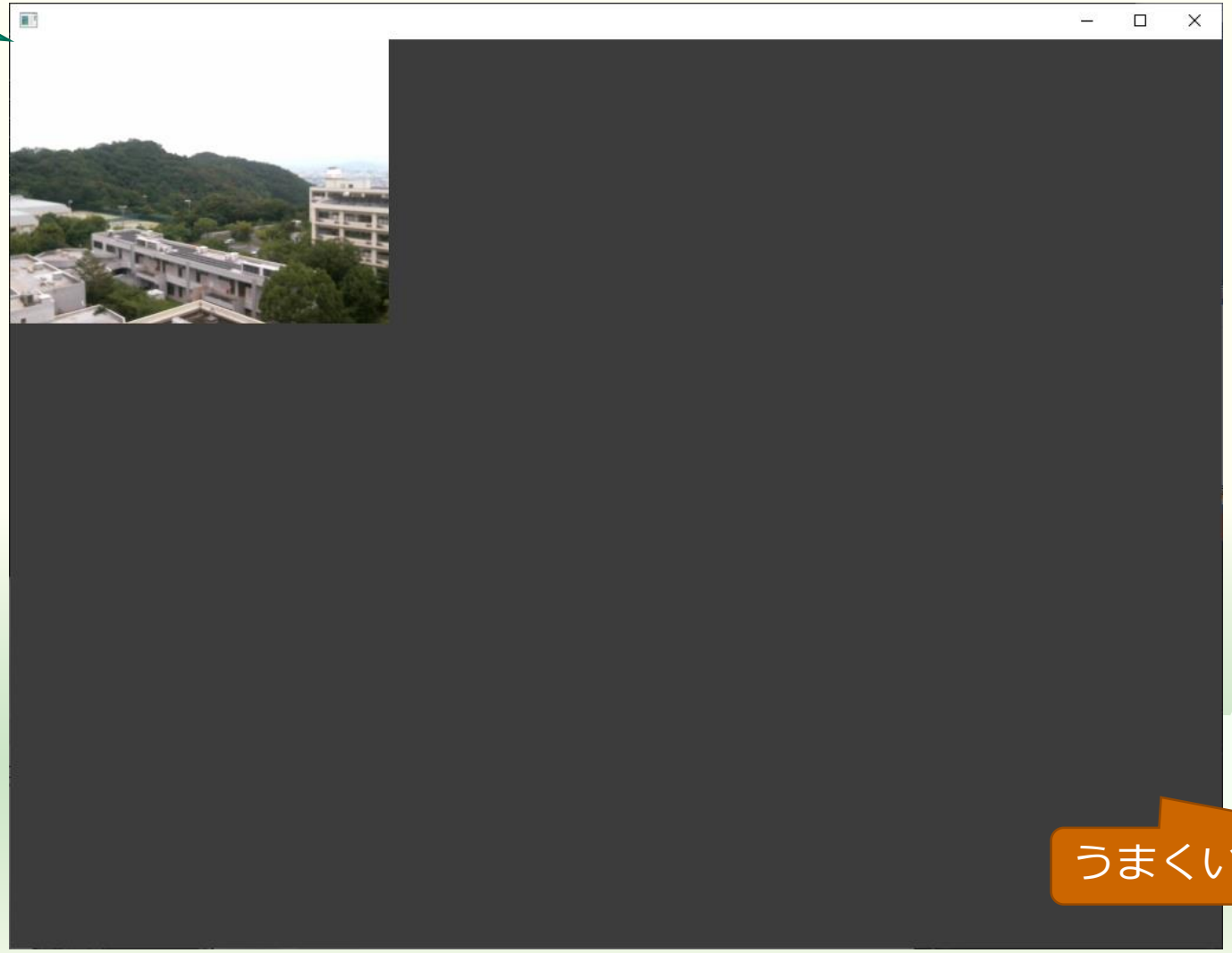
- video.setup(320, 240);
  - 映像入力の初期設定
  - 320, 240 は入力する映像の解像度
    - 640, 480 や 1280, 720 に対応するかどうかはカメラ次第なので試して
- video.update();
  - 映像入力から 1 フレーム取り込む
- video.draw(0, 0);
  - 入力した映像の 1 フレームをウィンドウの原点（0, 0、左上隅）から描画する

ビルドと実行


ファイル(F)
編集(E)
表示(V)
プロジェクト(P)
ビルド(B)
デバッグ(D)
テスト(S)
分析(N)
ツール(T)
拡張機能(X)
ウィンドウ(W)
ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myGoodSketch' (2/2 プロジェクト)
myGoodSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.cpp
myGoodSketch
(グローバル スコープ)
#include "ofApp.h"
//--
void ofApp::setup(){
video.setup(320, 240);
}
//--
void ofApp::update(){
video.update();
}
//--
void ofApp::draw(){
video.draw(0, 0);
}
//--
void ofApp::keyPressed(int key){
}
//--
void ofApp::keyReleased(int key){
}
100 %
問題は見つかりませんでした
行: 1
文字: 1
SPC
LF
準備完了
ソース管理に追加
サーバー エクスプローラー
ツールボックス
プロパティ

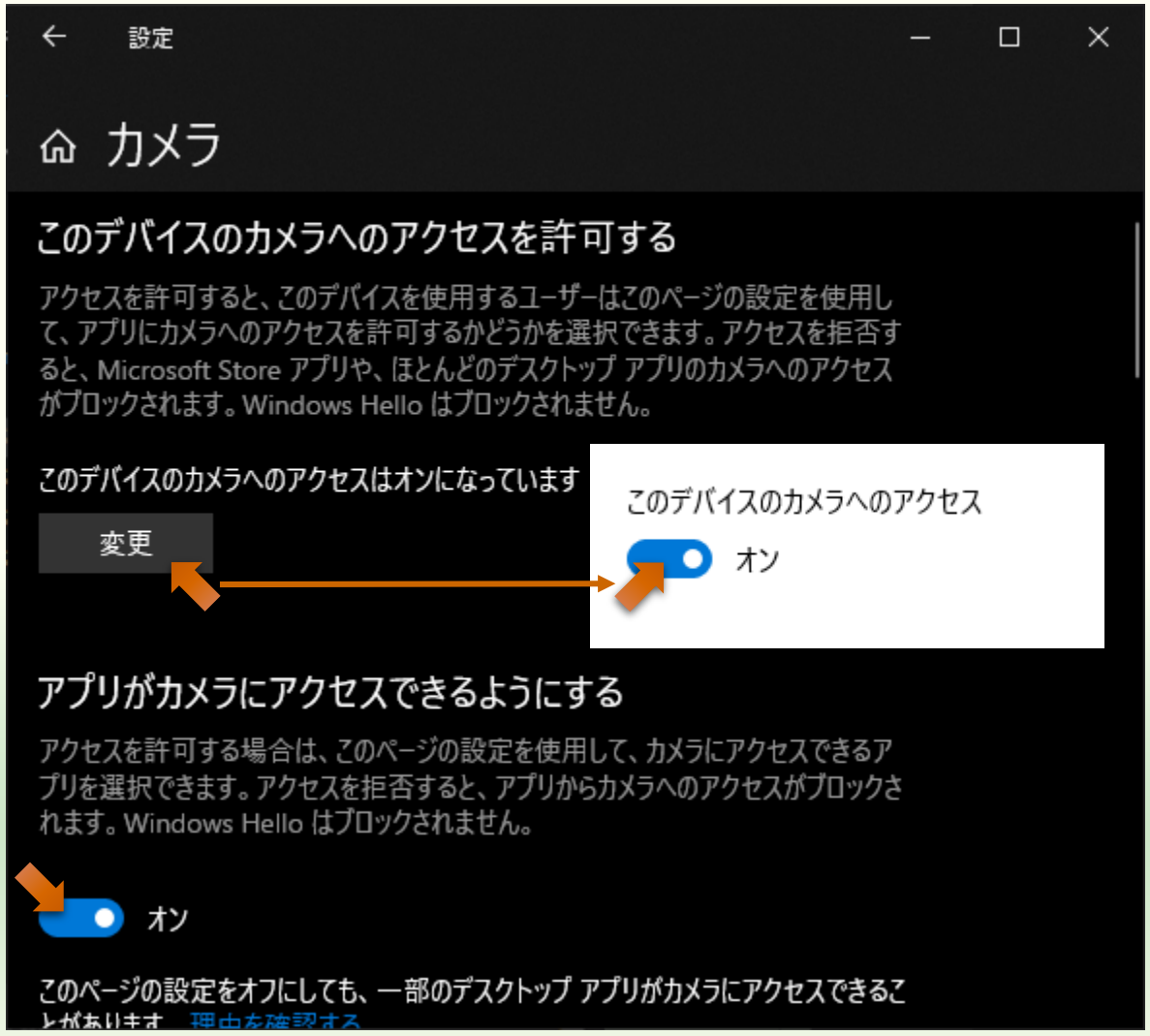
左上の原点を左上隅にして映像が表示される

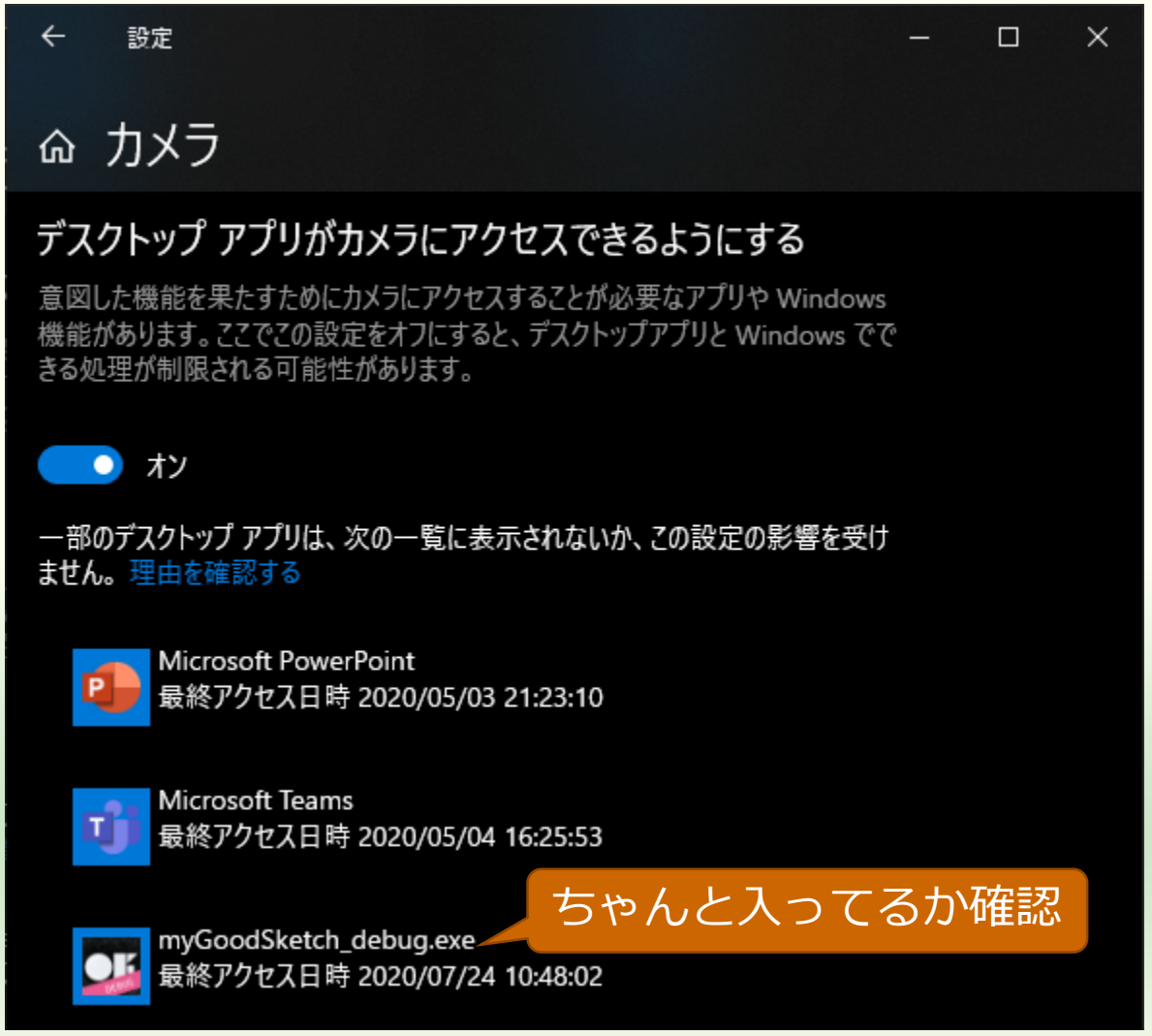
原点 (0, 0)



うまくいかなかったら次ページ

「⊞キー」→「設定」→「プライバシー」


設定
カメラ
このデバイスのカメラへのアクセスを許可する
アクセスを許可すると、このデバイスを使用するユーザーはこのページの設定を使用して、アプリにカメラへのアクセスを許可するかどうかを選択できます。アクセスを拒否すると、Microsoft Store アプリや、ほとんどのデスクトップ アプリのカメラへのアクセスがブロックされます。Windows Hello はブロックされません。
このデバイスのカメラへのアクセスはオンになっています
変更
このデバイスのカメラへのアクセス
オン
アプリがカメラにアクセスできるようにする
アクセスを許可する場合は、このページの設定を使用して、カメラにアクセスできるアプリを選択できます。アクセスを拒否すると、アプリからカメラへのアクセスがブロックされます。Windows Hello はブロックされません。
オン
このページの設定をオフにしても、一部のデスクトップ アプリがカメラにアクセスできるこ


設定
カメラ
デスクトップ アプリがカメラにアクセスできるようにする
意図した機能を果たすためにカメラにアクセスすることが必要なアプリや Windows 機能があります。ここでこの設定をオフにすると、デスクトップアプリと Windows でできる処理が制限される可能性があります。
オン
一部のデスクトップ アプリは、次の一覧に表示されないか、この設定の影響を受けません。理由を確認する
Microsoft PowerPoint
最終アクセス日時 2020/05/03 21:23:10
Microsoft Teams
最終アクセス日時 2020/05/04 16:25:53
myGoodSketch_debug.exe
最終アクセス日時 2020/07/24 10:48:02
ちゃんと入ってるか確認

デバイス番号を変えてみる

```
#include "ofApp.h"

//--------------------------------------------------------------
void ofApp::setup(){
  video.setDeviceID(1);
  video.setup(320, 240);
}

（以下略）
```

デフォルトは 0 なので
1 とかに変えてみる

1 が使えないと
こうなる

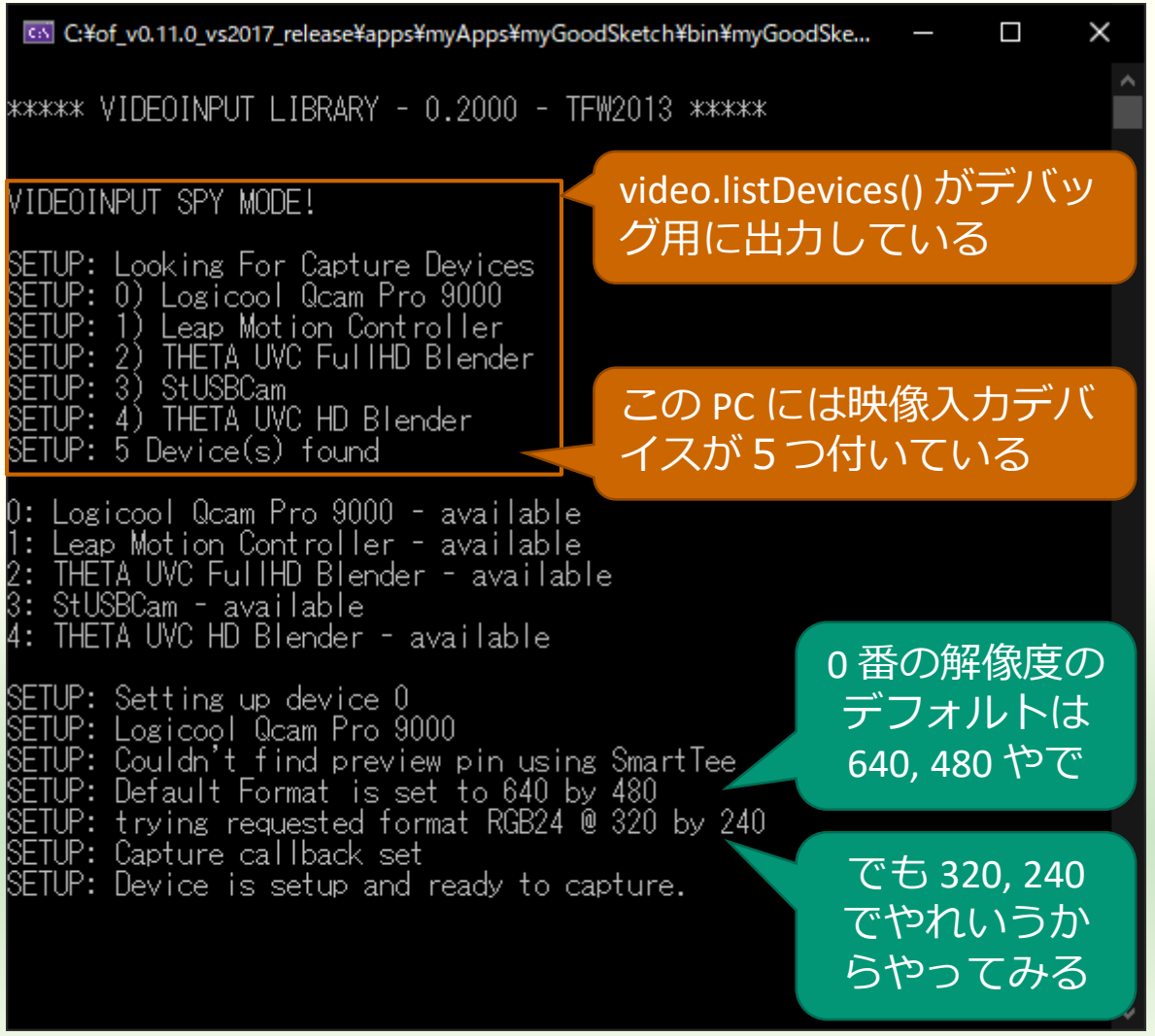
1 は使えない
というエラー

0 までしか使えないと言っている

使えるデバイスを調べる

```
#include "ofApp.h"

//--------------------------------------------------------
void ofApp::setup(){
 for (auto device : video.listDevices()){
    cout << device.id << ": " << device.deviceName;
    if (device.bAvailable){
      // 使えるデバイスの後ろに "available" を付ける
      cout << " - available¥n";
    }
    else{
      // 使えないデバイスの後ろに "unavailable" を付ける
      cout << " - unavailable¥n";
    }
  }
  cout << endl;
  video.setDeviceID(0);
  video.setup(320, 240);
}

（以下略）
```

0 番を使うことにする

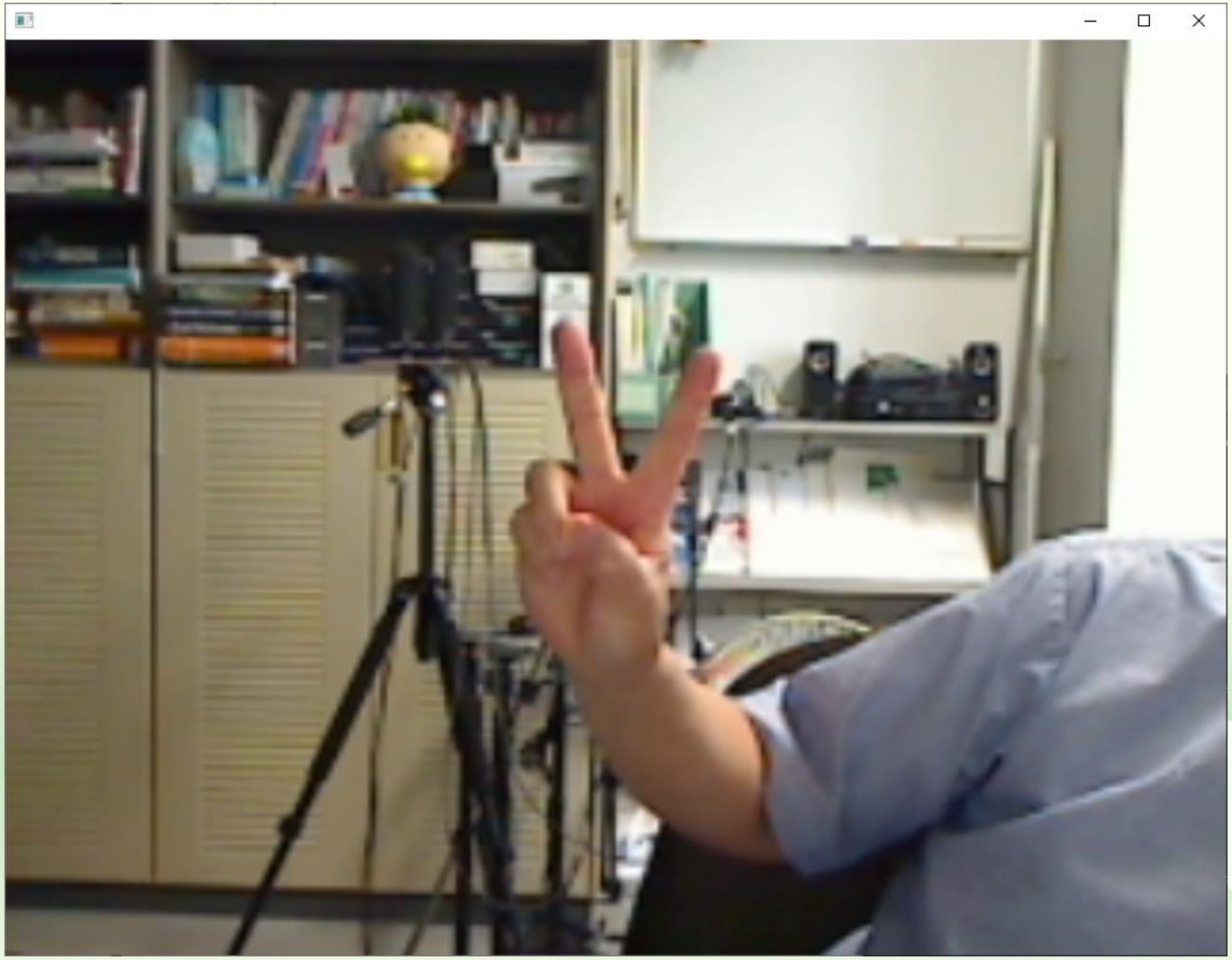


描画サイズをウィンドウサイズに合わせる

```
#include "ofApp.h"

//--------------------------------------------
void ofApp::setup(){
   （途中略）
}

//--------------------------------------------
void ofApp::update(){
  video.update();
}

//--------------------------------------------
void ofApp::draw(){
  video.draw(0, 0, ofGetWidth(), ofGetHeight());
}

（以下略）
```

- void ofVideoGrabber::draw(float x, float y)
  - ofVideoGrabber クラスの内部画像データの左上隅がウィンドウの (x, y) の位置になるように描画する
  - 内部画像データの画素数がそのまま反映される
- void ofVideoGrabber::draw(float x, float y, float w, float h)
  - 内部画像データの画素数を幅 w、高さ h に変換して描画する

解像度は低いが全画面に表示される

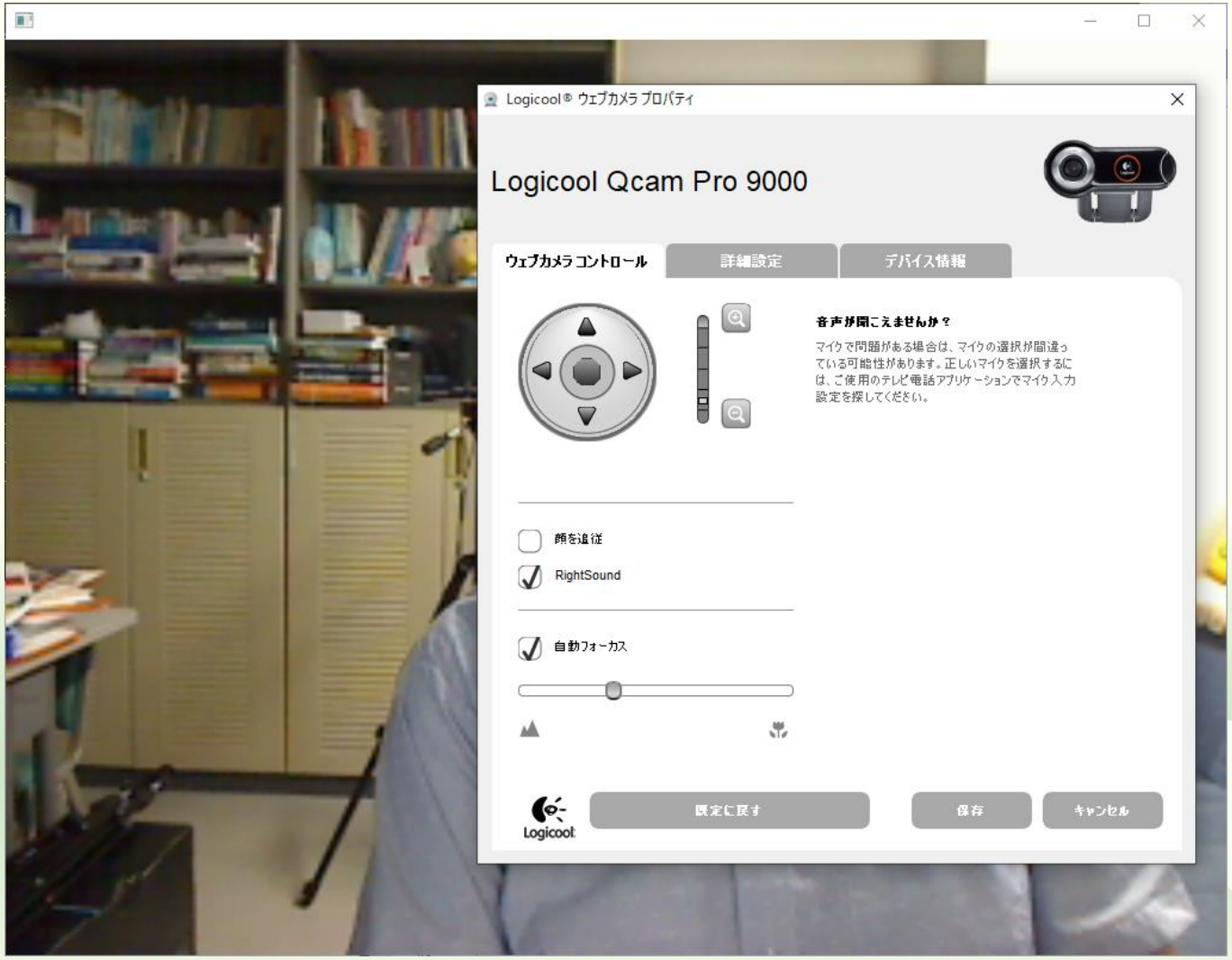


デバイスの設定パネルを呼び出せるようにする

```
#include "ofApp.h"

 (途中略)

//--------------------------------------------------
void ofApp::keyPressed(int key){
  if (key == 's' || key == 'S'){
    video.videoSettings();
  }
}

 (以下略)
```

- ‘s’ キーか ‘S’ キーをタイプしたら映像入力デバイスの設定パネルを開く
  - 映像入力デバイスが対応している場合

制御パネルを呼び出す


Logicool® ウェブカメラ プロパティ
Logicool Qcam Pro 9000
ウェブカメラ コントロール
詳細設定
デバイス情報
音声が聞こえませんか？
マイクで問題がある場合は、マイクの選択が間違っている可能性があります。正しいマイクを選択するには、ご使用のテレビ電話アプリケーションでマイク入力設定を探してください。
顔を追従
RightSound
自動フォーカス
既定に戻す
保存
キャンセル
Logicool

映像の加工

画素データの変更

ofApp クラスに画素データとテクスチャのメンバ変数を追加する

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{
  ofVideoGrabber video;
  ofPixels color;
  ofTexture texture;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  （以下略）
```

- ofPixels
  - 画像の画素データのクラス
- ofTexture
  - 描画する画像を保持するクラス
  - テクスチャマッピング用のデータ

ofApp.cpp で画素データ用のメモリを確保する

```
#include "ofApp.h"

//--------------------------------------------------
void ofApp::setup(){
   （途中略）
  video.setDeviceID(0);
  video.setup(320, 240);
  color = video.getPixels();
}

//--------------------------------------------------
void ofApp::update(){
  video.update();
}

（以下略）
```

- 画素データの表示に用いる変数 color のサイズやチャネル数を入力映像と同じにするために入力画像 video の画素データ自体をコピーする

フレームが更新されたか調べる

```
#include "ofApp.h"

//--------------------------------------------------
void ofApp::setup(){
    (途中略)
}

//--------------------------------------------------
void ofApp::update(){
  video.update();
  if (video.isFrameNew()){
    (次ページに続く)
```

- video.update();
  - 映像入力からフレームを取り込む
- if (video.isFrameNew()){
  - もしフレームが更新されていたら { } 内の処理をする
  - ofApp::update() や ofApp::draw() は画面表示のタイミングで実行されるが映像入力のタイミングは必ずしもそれと一致しない
  - そのため video.update(); しても前のフレームのまま更新されていないことがある

更新されたフレームの画素データをコピーする

```
    （前ページからの続き）
    ofPixels &input = video.getPixels();
    for (size_t i = 0; i < input.size(); ++i){
      color[i] = input[i];
    }
    texture.loadData(color);
  }
}

（以下略）
```

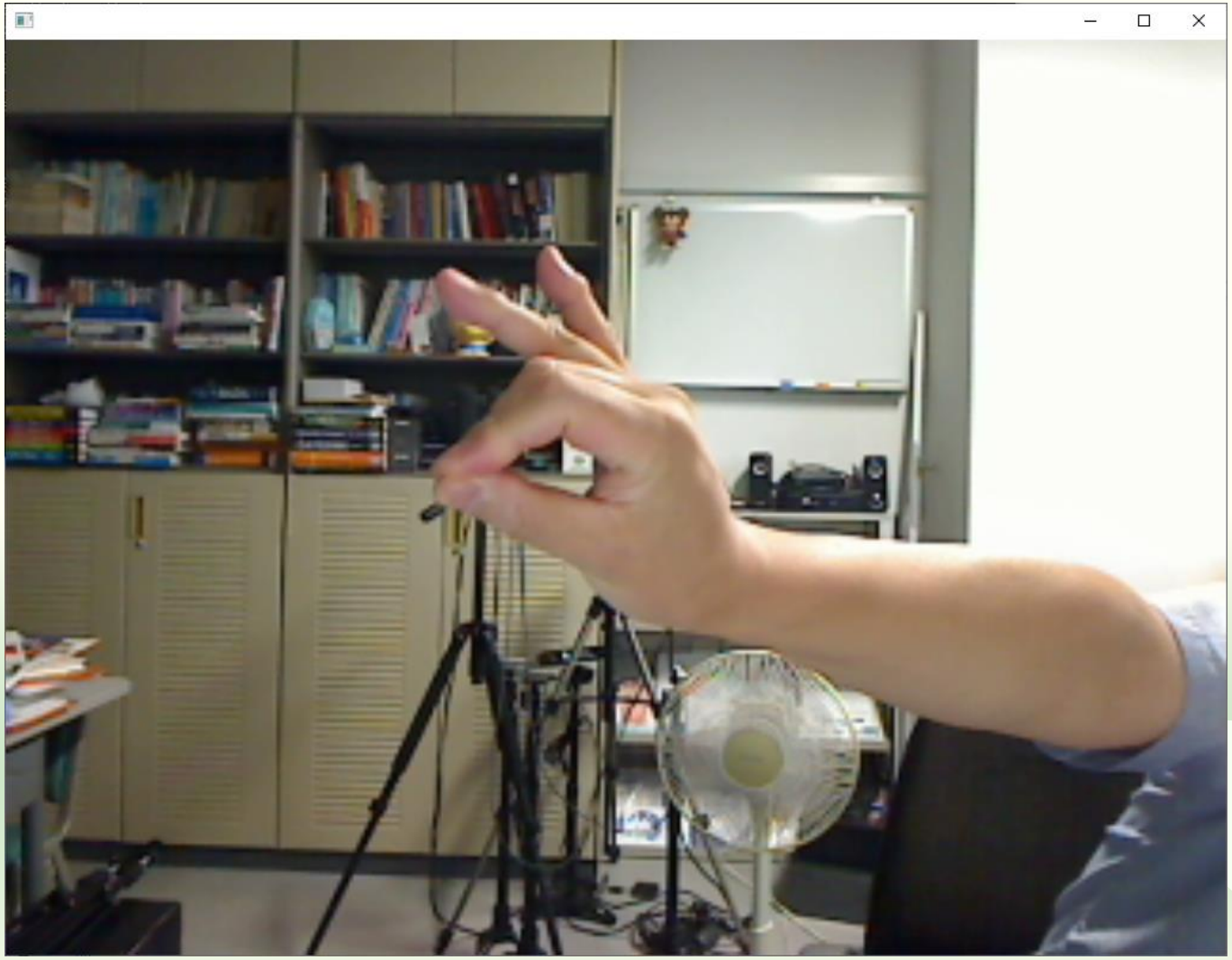
- ofPixels &input = video.getPixels();
  - 更新されたフレームの画素データを input で**参照**できるようにする
- for (size_t i = 0; i < input.size(); ++i){
  - 更新されたフレームの画素データ input の数だけ { } 内を繰り返す
  - color[i] = input[i];
    - 更新されたフレームの画素データ input を表示用画素データ color にコピーする
- texture.loadData(color);
  - コピーした画素データを描画に使うテクスチャデータに転送する

画素データを描画する

```
#include "ofApp.h"

//--------------------------------------------------
void ofApp::setup(){
   (途中略)
}

//--------------------------------------------------
void ofApp::update(){
   (途中略)
}

//--------------------------------------------------
void ofApp::draw(){
  texture.draw(0, 0, ofGetWidth(), ofGetHeight());
}

 (以下略)
```

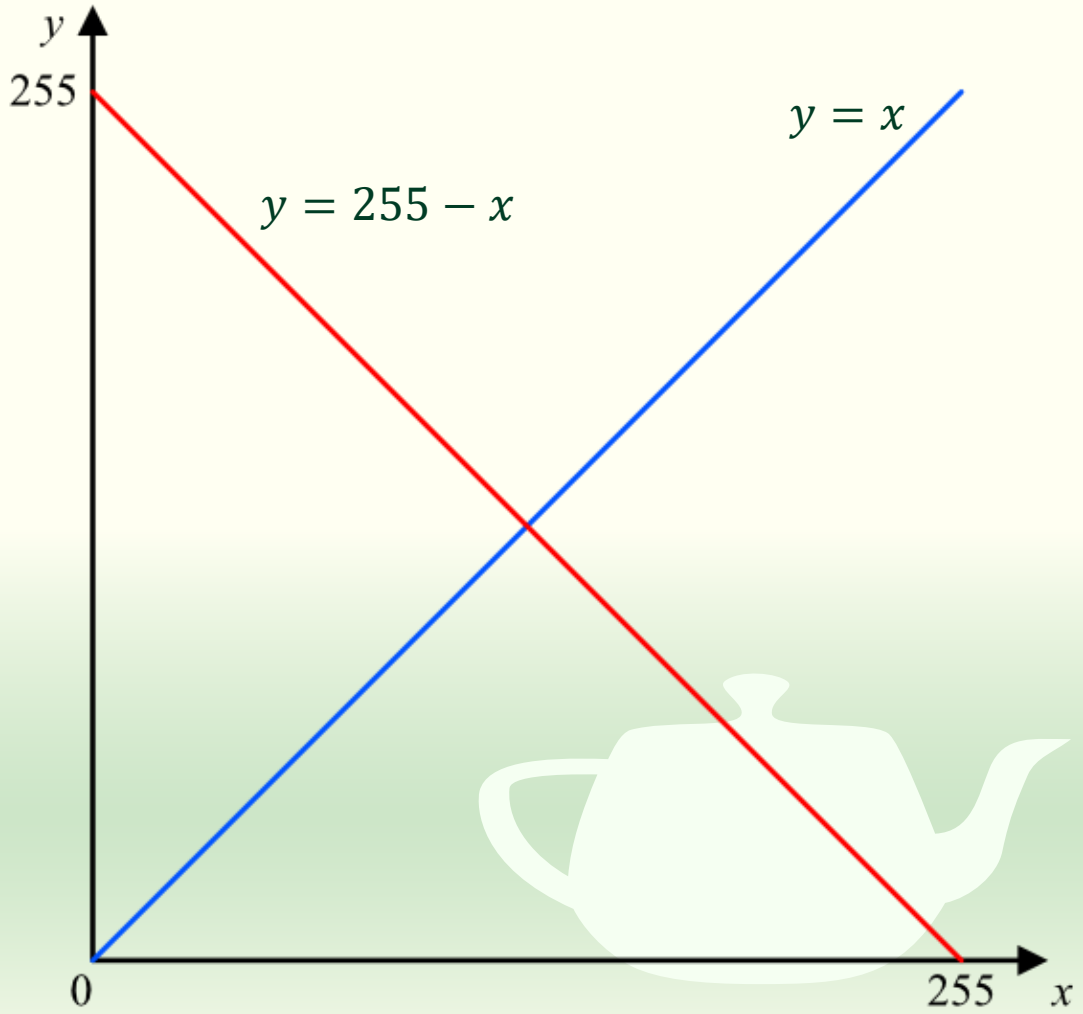
- ofPixels クラスの画素データは直接描画できない
  - ofPixels::draw() などというメソッドはない

特に変わらない

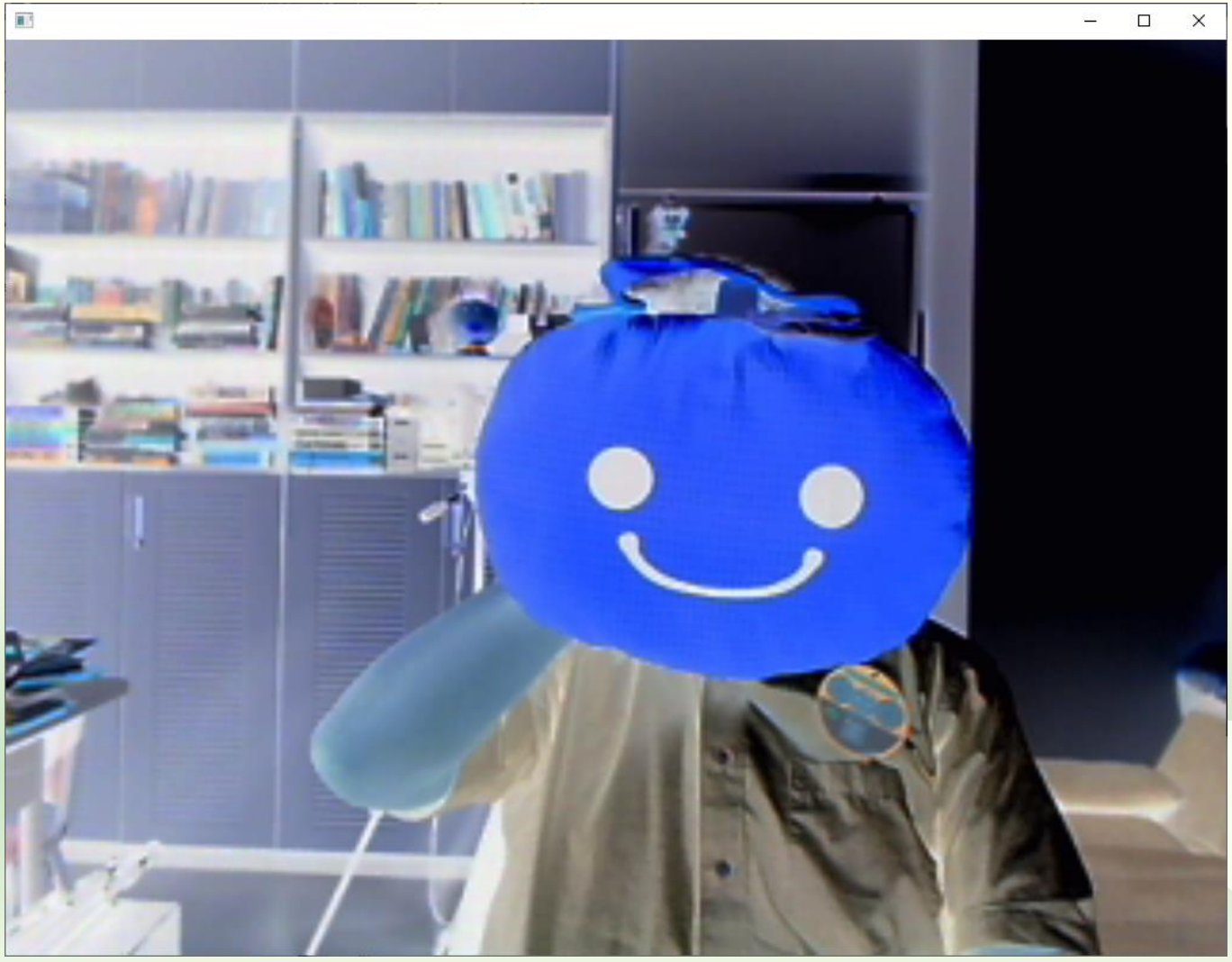


明るさを反転する

```
#include "ofApp.h"

//--------------------------------------------------
void ofApp::setup(){
   (途中略)
}

//--------------------------------------------------
void ofApp::update(){
  video.update();
  if (video.isFrameNew()){
    ofPixels &input = video.getPixels();
    for (size_t i = 0; i < input.size(); ++i){
      color[i] = 255 - input[i];
    }
    texture.loadData(color);
  }
}

 (以下略)
```



階調の反転



加工方法の切り替え

キー操作で処理を切り替えられるようにする

ofApp クラスにタイプしたキーを保持するメンバ変数を追加する

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{
  ofVideoGrabber video;
  ofPixels color;
  ofTexture texture;
  int select;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  （以下略）
```

- ofApp クラスに int 型の select というメンバ変数を追加する
  - これにキー操作の結果を入れる

ofApp.cpp でキー操作の初期設定を行う

```
#include "ofApp.h"

//--------------------------------------------------
void ofApp::setup(){
   （途中略）
  video.setDeviceID(0);
  video.setup(320, 240);
  color = video.getPixels();
  select = '0';
}

（以下略）
```

- select の初期値は文字定数の ‘0’ にしておく
  - 実はこの後（次の次のページ）で select が ‘0’ の時に color = input; という代入をして color に input のコピーを用意するので、ここでメモリを確保する必要はなくなる

タイプしたキーを select に代入する

```
#include "ofApp.h"


//--------------------------------------------
void ofApp::draw(){
  texture.draw(0, 0, ofGetWidth(), ofGetHeight());
}

//--------------------------------------------
void ofApp::keyPressed(int key){
  if (key == 's' || key == 'S'){
    video.videoSettings();
  }
  else{
    select = key;
  }
}
```

- key を select に代入する
  - ‘s’, ‘S’ 以外の**文字コード**が入る

select の内容によって異なり処理をする

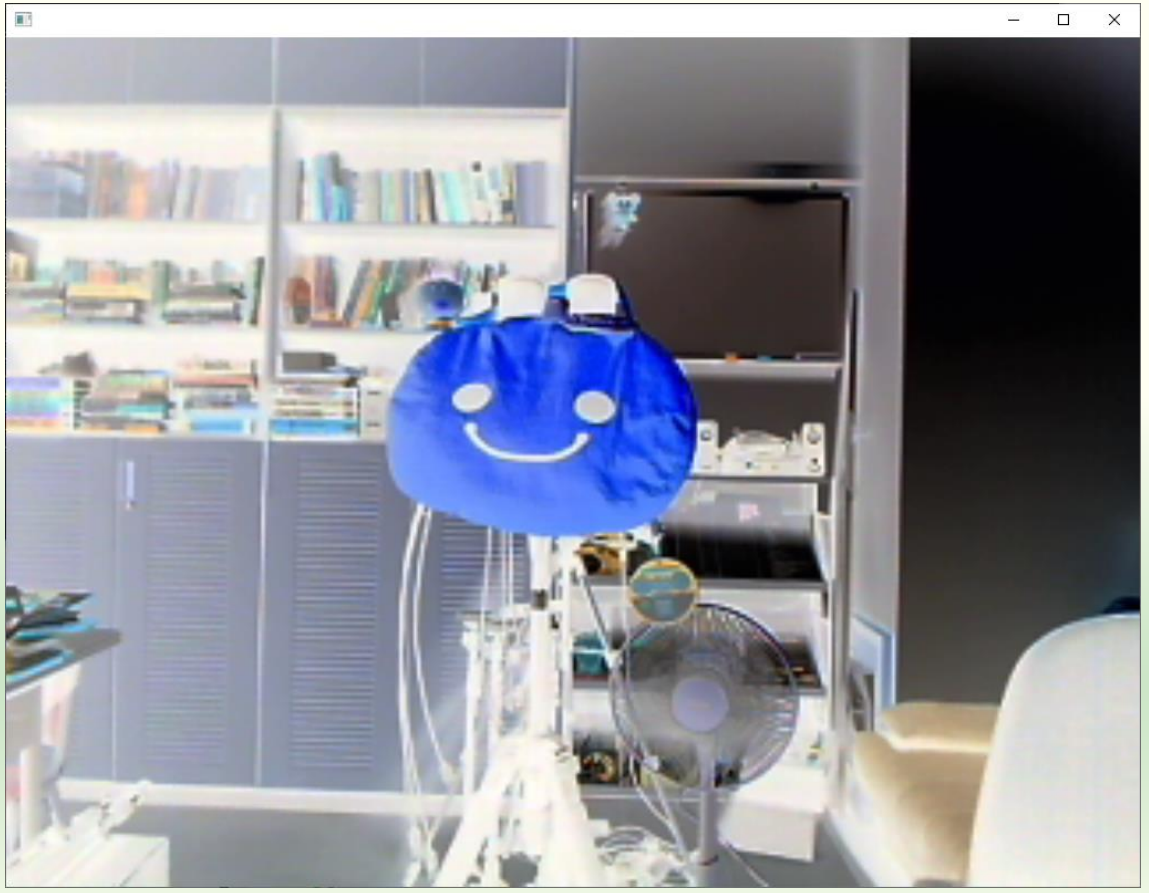
```
//--------------------------------------------
void ofApp::update(){
  video.update();
  if (video.isFrameNew()){
    ofPixels &input = video.getPixels();
    switch (select){
    case '0':
      color = input;
      break;
    case '1':
      for (size_t i = 0; i < input.size(); ++i){
        color[i] = 255 - input[i];
      }
      break;
    default:
      break;
    }
    texture.loadData(color);
  }
}
```

- swich (select){
  - select の内容に従って { } 内の case ラベルに分岐する
- case '0':
  - select の内容が ‘0’ のとき、これ以降が実行される
- color = input;
  - input の全画素データを color にコピーする
- break;
  - swich の { } 内の処理から抜け出る
- default:
  - select がどの case ラベルとも一致しないとき、これ以降が実行される

キーのタイプで結果が変わる

‘0’ のタイプ以降

‘1’ のタイプ以降

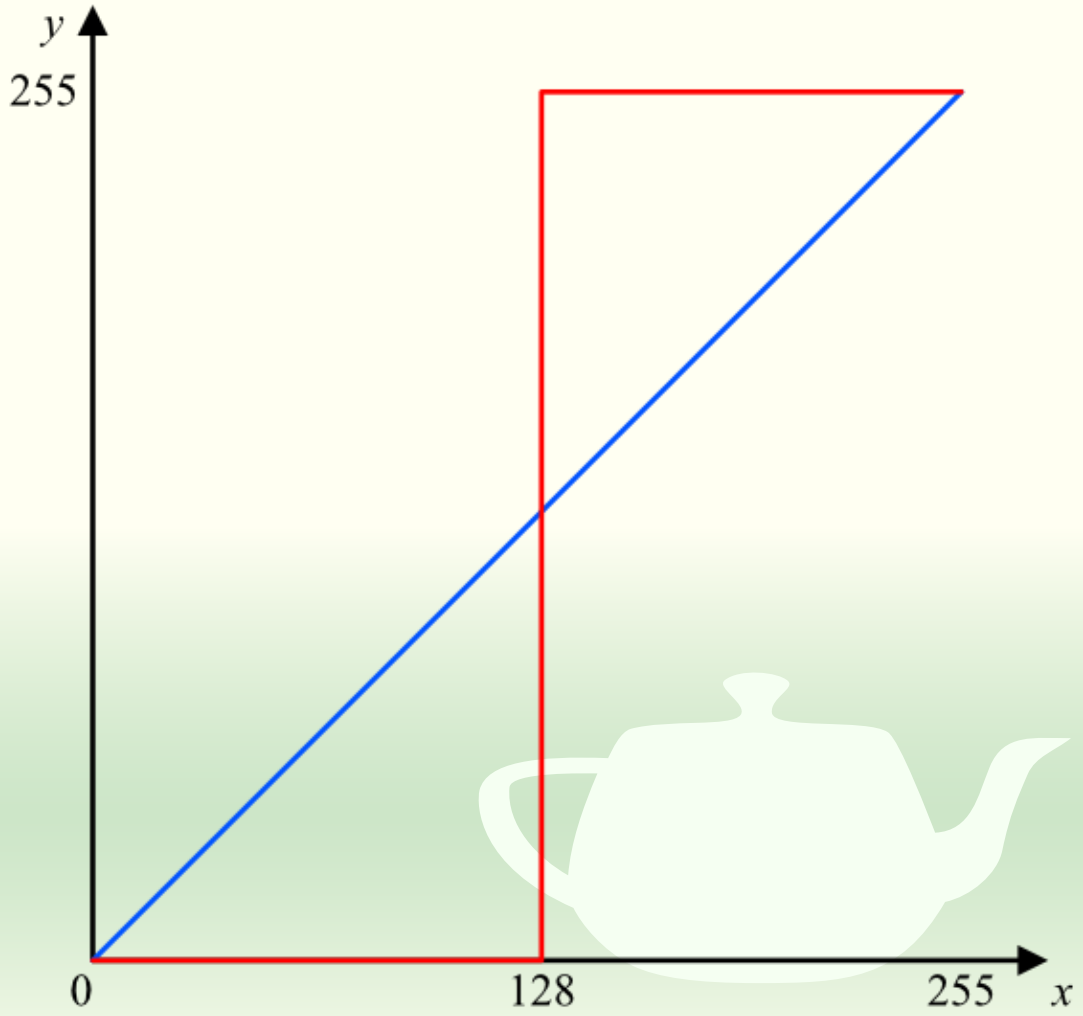


課題５－１

２値化

画像の各チャネルを二値化して表示する

- ‘2’ キーをタイプしたら R, G, B のそれぞれのチャネルを２値化して表示するようにしなさい
- ２値化
  - 入力データ x を閾値（いきち）t と比較して、$x < t$ なら値を最小値（ここでは 0）に、それ以外なら最大値（ここでは 255）にする
  - ここでは閾値 t を 128 とする



結果の例

課題のアップロード

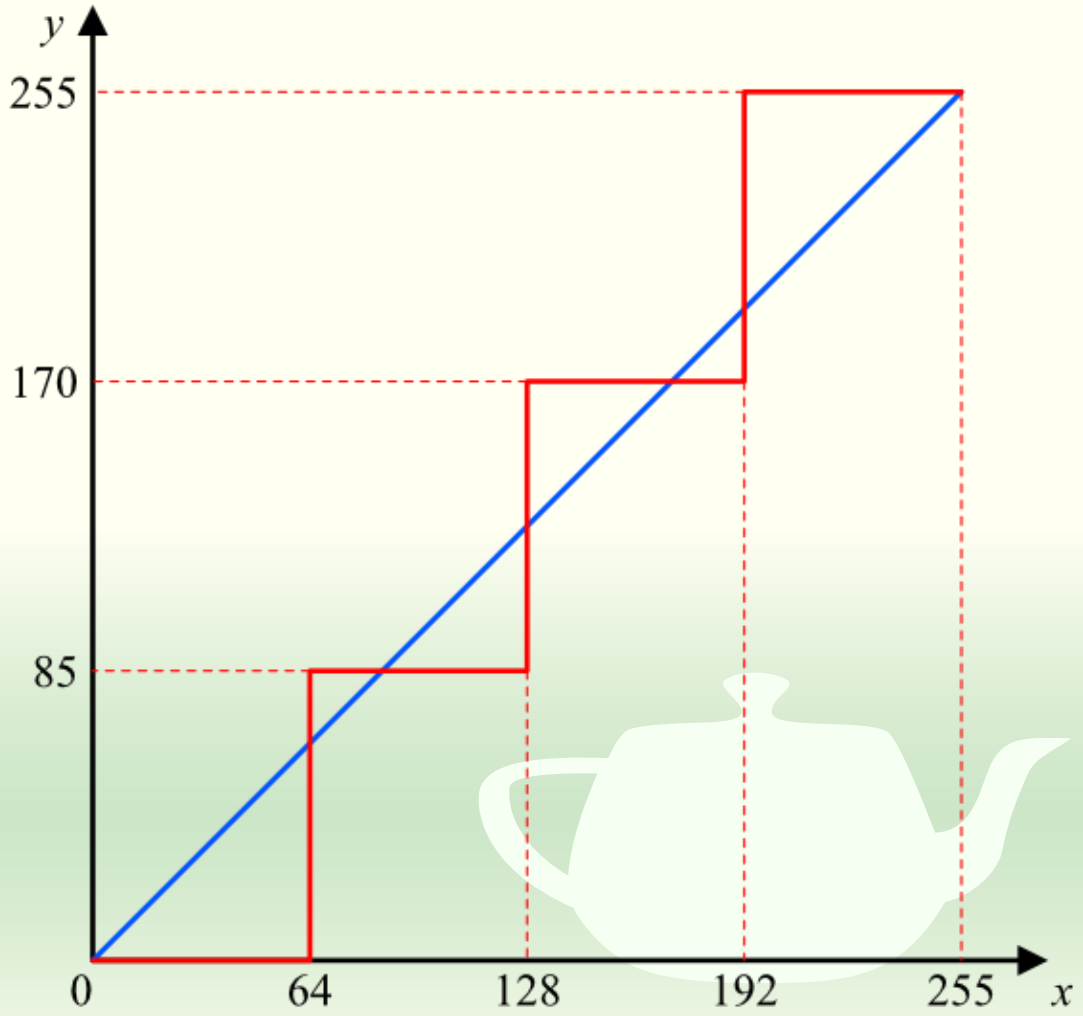
- 作成したプログラムの実行結果のスクリーンショットを撮って **5-1.png** というファイル名で保存し、Moodle の第 5 回課題にアップロードしてください

課題５－２

ポスター化

画像の各チャネルを４階調化して表示する

- ‘3’ キーをタイプしたら映像を４階調化（４階調のポスター化）して表示するようにしなさい
- 画素値 x は 0~255 の値を持つ
- x を 64 で割り小数点以下を切り捨てれば 0~3 の値になる
  - 整数の除算をすれば小数点以下は切り捨てられる
- これに 85 を掛ければ 0, 85, 170, 255 のいずれかの値が得られる



結果の例

課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **5-2.png** というファイル名で保存し、Moodle の第 5 回課題にアップロードしてください

チャネルの入れ替え

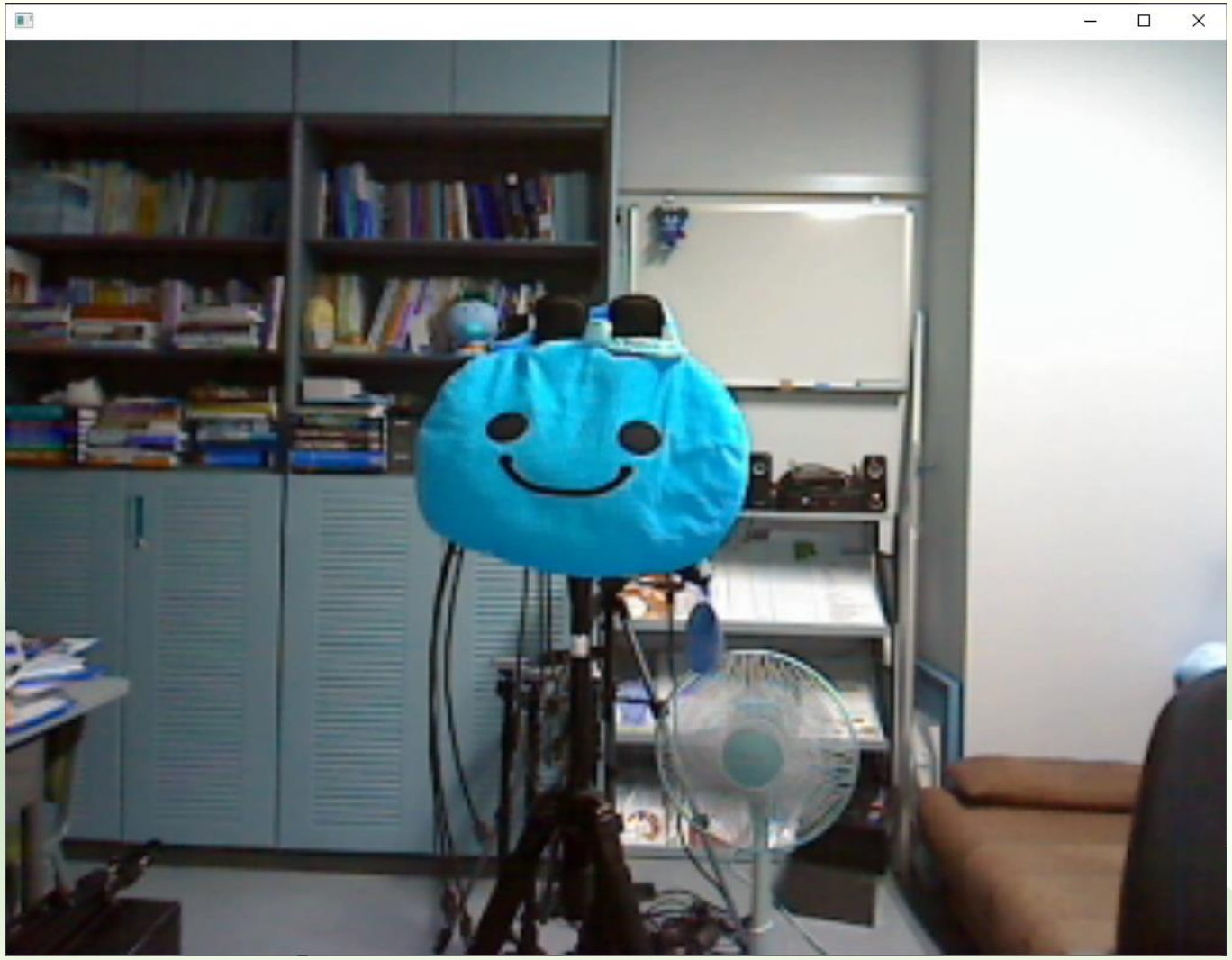
R と B を入れ替えて RGB を BGR にする

‘4’ キーで赤と青を入れ替えて表示する

```
//--------------------------------------------------
void ofApp::update(){
  video.update();
  if (video.isFrameNew()){
    ofPixels &input = video.getPixels();
    switch (select){
     （途中略）
    case '4':
      for (size_t i = 0; i < input.size(); i += 3){
        int r = input[i];
        int g = input[i + 1];
        int b = input[i + 2];
        color[i] = b;
        color[i + 1] = g;
        color[i + 2] = r;
      }
      break;
    default:
      break;
    }
     （以下略）
```

- ofPixels &input = video.getPixels();
  - input は 1 画素の各チャネルが要素ごとに入っている
  - RGB（アルファチャネルなし）
    - input[0] → R, input[1] → G, input[2] → B
    - input[3] → R, input[4] → G, input[5] → B
  - 従って要素の番号 i = 0 から始めて i < input.size() の間以下を繰り返す
    - r に input[i], g に input[i + 1], b に input[i + 2] を代入する
    - color[i] に b, color[i + 1] に g, color[i + 2] に r を代入する
    - i を 3 増やす (i += 3)

結果の例



課題５－３

グレースケール化

画像をグレースケール化して表示する

- ‘5’ キーをタイプしたら映像をグレイスケール化（白黒画像化）して表示するようにしなさい
- グレイスケール化
  - $Y = (R + G + B)/3$
    - RGB 平均、または
  - $Y = 0.299R + 0.587G + 0.114B$
    - ITU-R Rec BT.601
    - 人間の目は G に対して感度が高く B に対して低いという特性があるため RGB 平均だと G が低めに出る
  - 他にも多くの変換式がある
- ofPixels &input = video.getPixels();
  - input は 1 画素の各チャネルが要素ごとに入っている
  - RGB（アルファチャネルなし）
    - input[0] → R, input[1] → G, input[2] → B
    - input[3] → R, input[4] → G, input[5] → B
  - 従って要素の番号 i = 0 から始めて i < input.size() の間以下を繰り返す
    - input[i], input[i + 1], input[i + 2] をグレースケール化して y を求める
    - color[i], color[i + 1], color[i + 2] にいずれも y を代入する
    - i を 3 増やす (i += 3)

結果の例

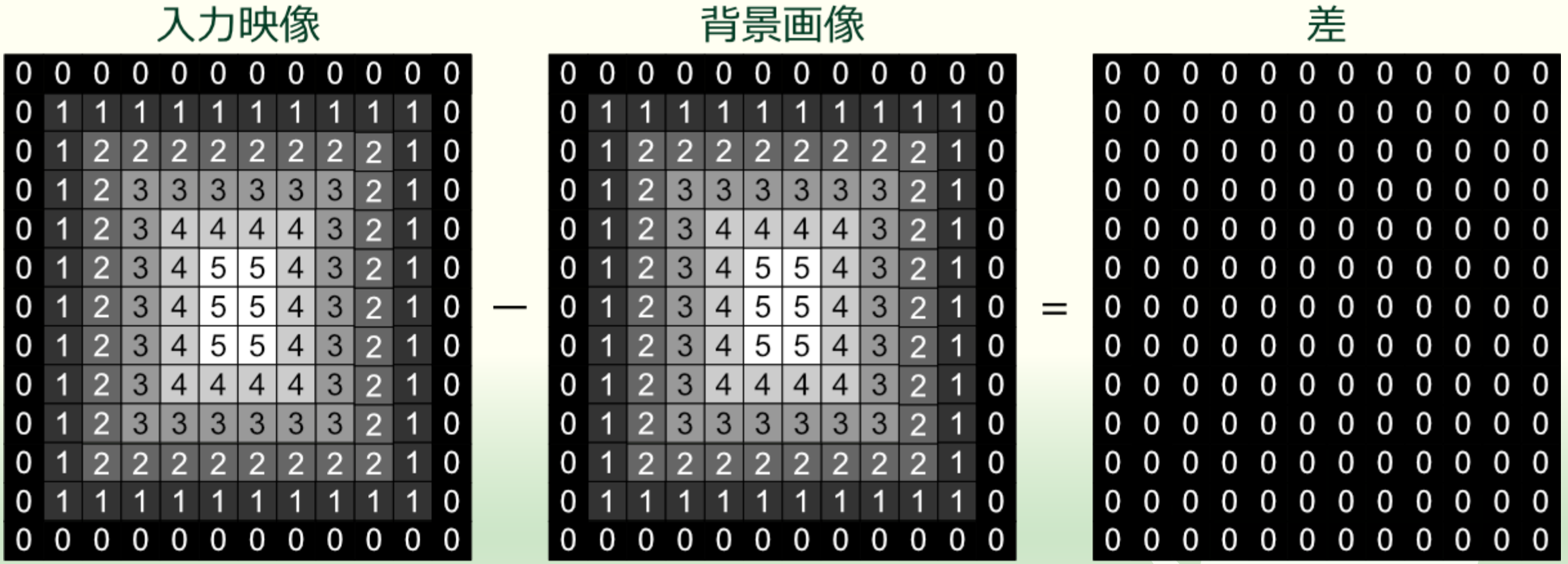
課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **5-3.png** というファイル名で保存し、Moodle の第 5 回課題にアップロードしてください

背景差分法

背景を切り抜く

画像の差

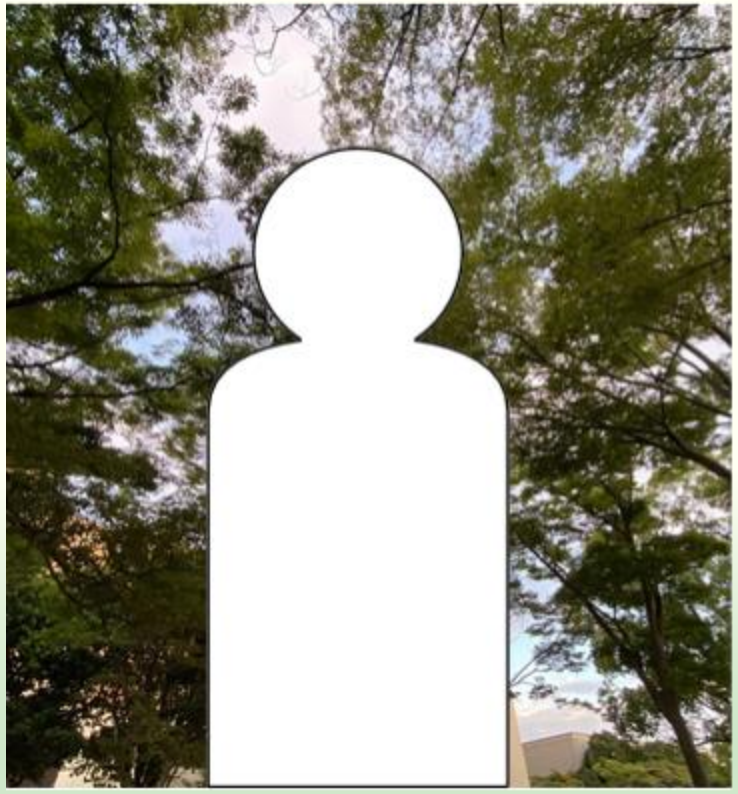


画像は２次元の（画素の）データの配列⇒対応する画素同士の引き算が可能

同じ画像同士を引き算すると、当然全部０すなわち**黒**

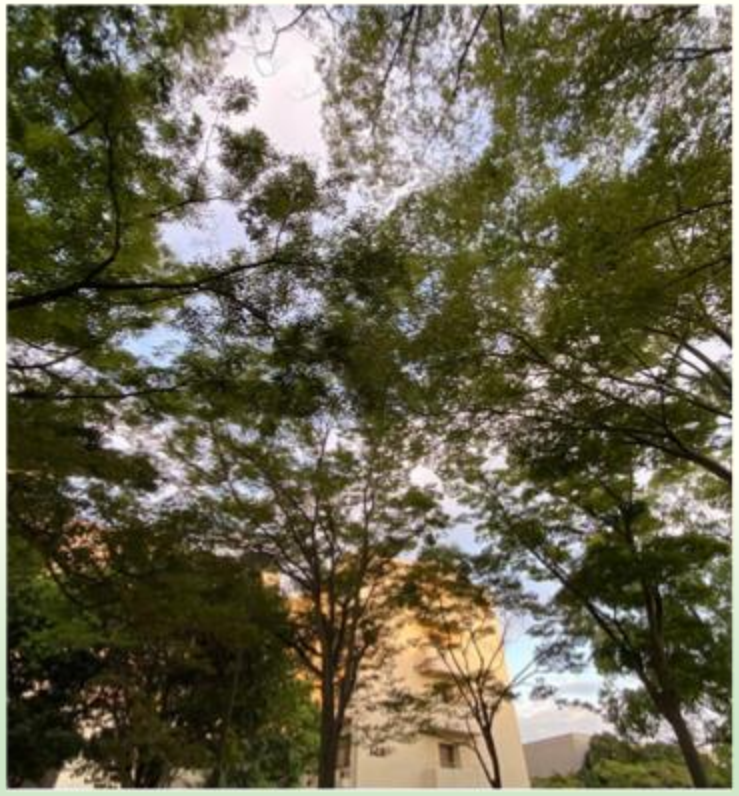
差異部分の抽出

入力映像



−

背景画像



=

差

したがって二つの画像に異なる部分があれば、その部分だけが 0 でなくなる
逆に 0 に近い部分は内容が似ているので、その部分が背景だと判断できる

ofApp.h で ofApp クラスに背景を保存するメンバ変数を追加する

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{
  ofVideoGrabber video;
  ofPixels color, saved;
  ofTexture texture;
  int select;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  （以下略）
```

- ofApp クラスのメンバ変数に ofPixels クラスの saved というインスタンスを追加する
  - saved に背景の画像を保存する

ofApp.cpp で保存用のメモリを確保する

```
#include "ofApp.h"

//--------------------------------------------------
void ofApp::setup(){
   （途中略）
  video.setDeviceID(0);
  video.setup(320, 240);
  saved = color = video.getPixels();
  select = '0';
}

//--------------------------------------------------
void ofApp::update(){
  video.update();
}

（以下略）
```

- 画素データの保存に用いる変数 saved のサイズとチャネル数も入力映像と同じにする

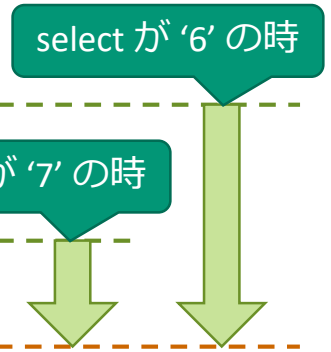
ofApp.cpp で ‘6’ キーをタイプした時にフレームを保存する

```
//--------------------------------------------
void ofApp::update(){
  video.update();
  if (video.isFrameNew()){
    ofPixels &input = video.getPixels();
    switch (select){
    case '0':
      color = input;
      break;
    case '1':
      for (size_t i = 0; i < color.size(); ++i){
        color[i] = 255 - input[i];
      }
      break;
     (途中略)
    case '6':
      saved = input;
      select = '7';
      break;
    default:
     (以下略)
```

- select が ‘6’ なら現在のフレームの画素データ input を saved にコピーして取っておく
- select に ‘7’ を代入しておく
  - この後で select が ‘7’ の時に saved の内容を表示するようにする

‘7’ キーをタイプしたら保存したフレームを表示する

```
//--------------------------------------
void ofApp::update(){
  video.update();
  if (video.isFrameNew()){
    ofPixels &input = video.getPixels();
    switch (select){
     (途中略)
    case '6': ----------------------------
      saved = input;
      select = '7';
    case '7': ----------------------------
      color = saved;
      break; -------------------------------
    default:
      break;
    }
     (以下略)
```



- select が ‘7’ なら保存したフレーム saved を表示するフレーム color にコピーする
- この処理を case ‘6’ の break の前に置く
  - したがって case ‘6’ と次の case ‘7’ 間には break がない
  - そのため select が ‘6’ の時は case ‘6’ と case ‘7’ の両方の処理が実行され、select が ‘7’ の時は case ‘7’ 以降の処理だけが実行される

‘6’ キーをタイプして画像を保存していれば

‘0’ で現在の映像が表示される

‘7’ で保存した画像が表示される

課題５－４

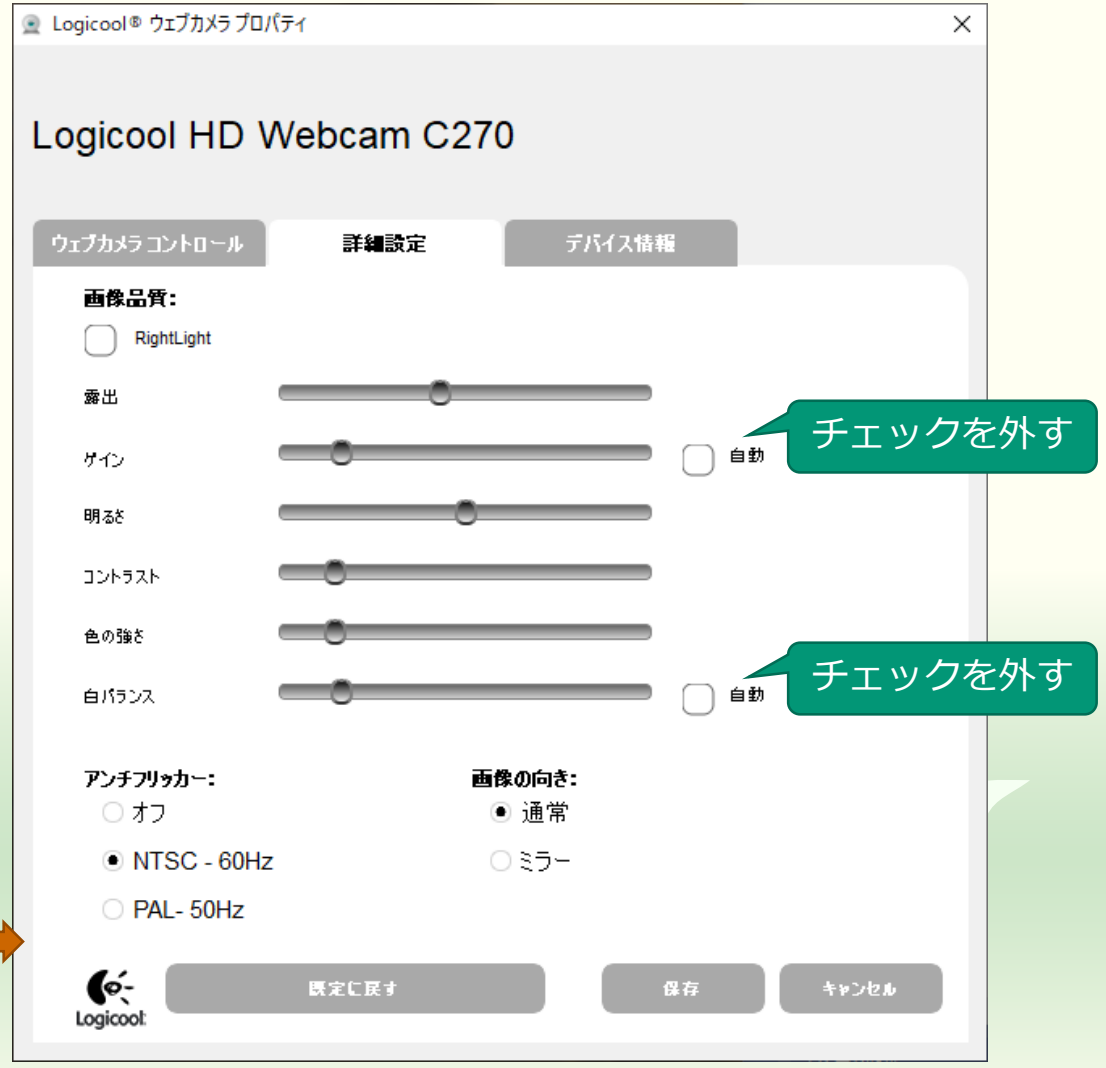
入力映像と背景画像の差の絶対値

入力映像と背景画像の差の絶対値を表示する

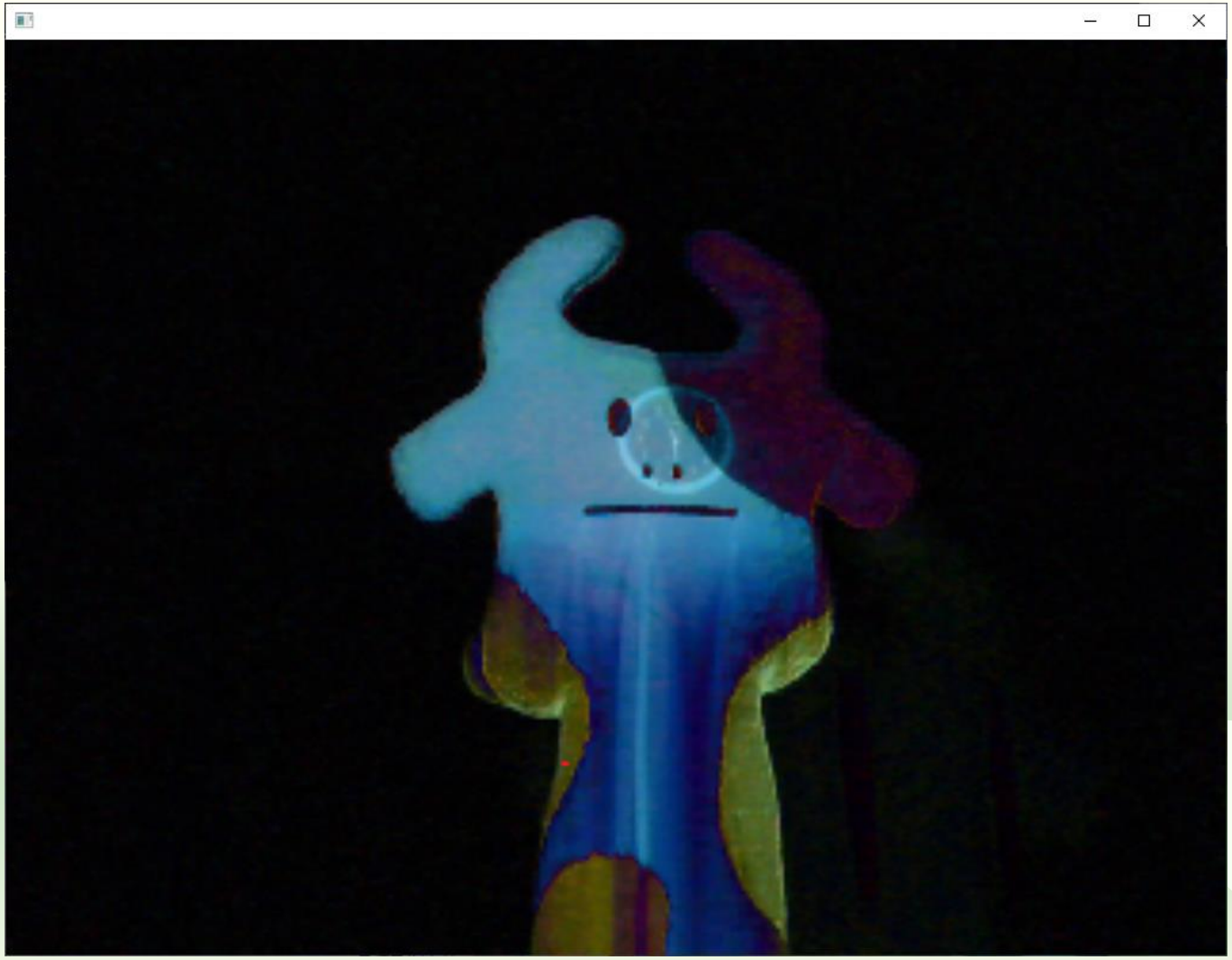
- ‘8’ キーをタイプしたら入力映像と背景画像の差の絶対値を表示するようにしなさい
  - input[i] と saved[i] の差の絶対値を color[i] に代入する
- 絶対値を求める標準ライブラリ関数 abs()
  - z = abs(x – y);
    - z は x と y の差の絶対値
  - https://cpprefjp.github.io/reference/cmath/abs.html

カメラの露出・利得を固定する

- この方法はカメラの露出や利得（ゲイン）を固定しておかないと背景がきれいに黒くならない
- ‘s’ または ‘S’ キーをタイプして制御パネルで調整する
  - カメラが制御パネルの呼び出しに対応していない場合はあきらめてください



結果の例（入力映像と背景画像の差の絶対値）



課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **5-4.png** というファイル名で保存し、Moodle の第 5 回課題にアップロードしてください

課題５－５

入力映像と背景画像の差の絶対値が閾値以下の画素を黒にする

差の絶対値が閾値以下の画素を黒で表示する

- ‘9’ キーをタイプしたら入力映像と背景画像の差の絶対値が閾値以下の画素を黒で表示するようにしなさい
- 画素値の差の絶対値との比較
  - 入力映像と背景画像の R, G, B のチャネルごとの差の絶対値のそれぞれと閾値を比較する
  - 全てのチャネルの差の絶対値が閾値を超えていれば入力映像の画素を表示し、そうでなければ黒 (R = G = B = 0) を表示する
- ofPixels &input = video.getPixels();
- 要素の番号 i = 0 から始めて i < input.size() の間以下を繰り返す
  - input[i] と saved[i]、input[i + 1] と saved[i + 1]、input[i + 2] と saved[i + 2] の差の絶対値をそれぞれ求める
  - これらの値のそれぞれと閾値を比較し、すべて閾値を超えていたら color[i], color[i + 1], color[i + 2] に input[i], input[i + 1], input[i + 2]、そうでなければ 0 を代入する
  - i を 3 増やす (i += 3)

結果の例（差の絶対値が閾値以下の画素を黒）

課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **5-5.png** というファイル名で保存し、Moodle の第 5 回課題にアップロードしてください

課題５－６

入力映像と背景画像の差の絶対値が閾値以下の領域に別の画像を表示する

ofApp クラスに別の画像を保持するメンバ変数を追加する

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{
  ofVideoGrabber video;
  ofPixels color, saved;
  ofTexture texture;
  ofImage image;
  int select;

public:
  void setup();
  void update();
  void draw();

  void keyPressed(int key);
  void keyReleased(int key);
  void mouseMoved(int x, int y);
  （以下略）
```

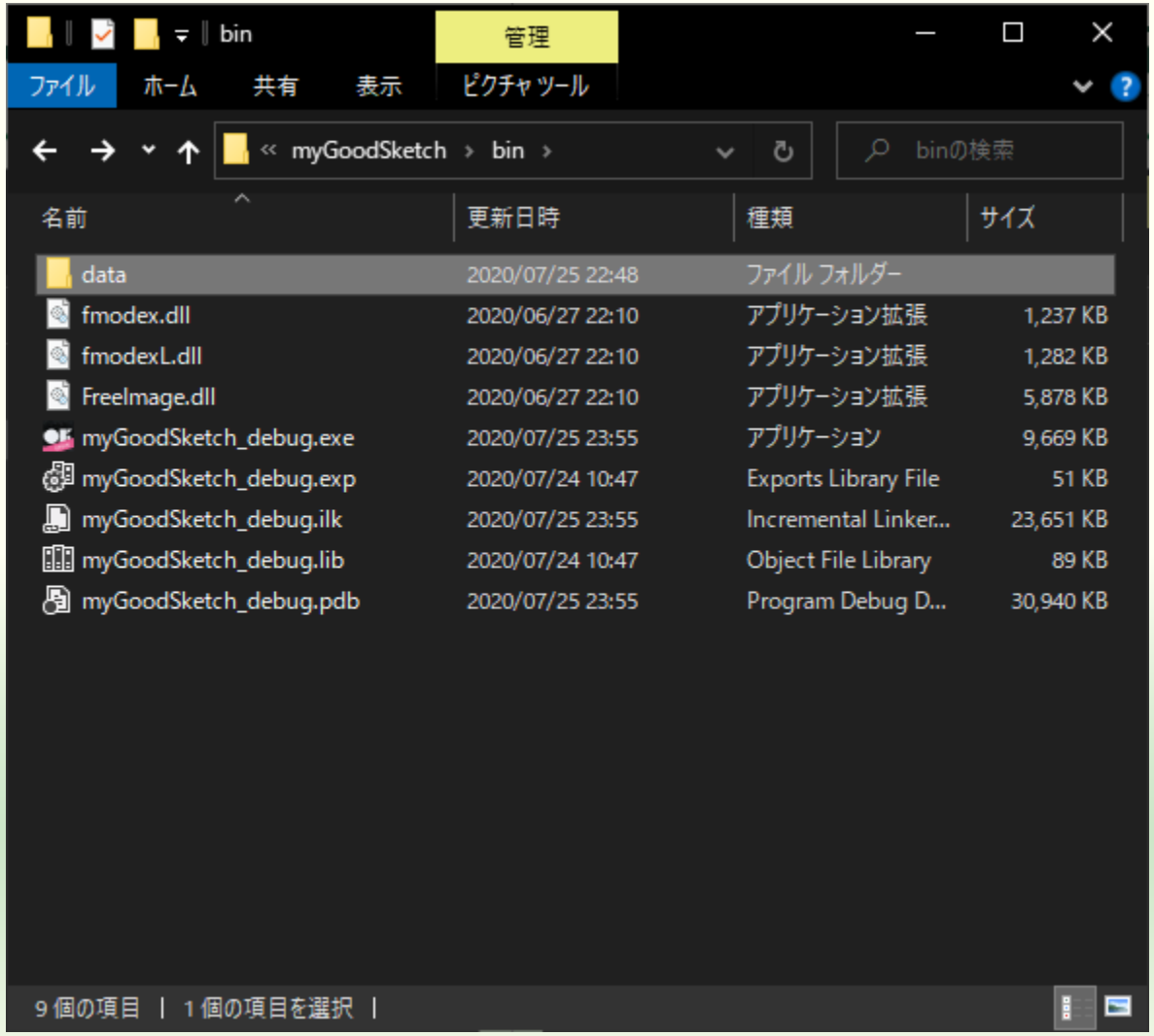
- ofApp クラスに ofImage クラスの image というメンバ変数を追加する

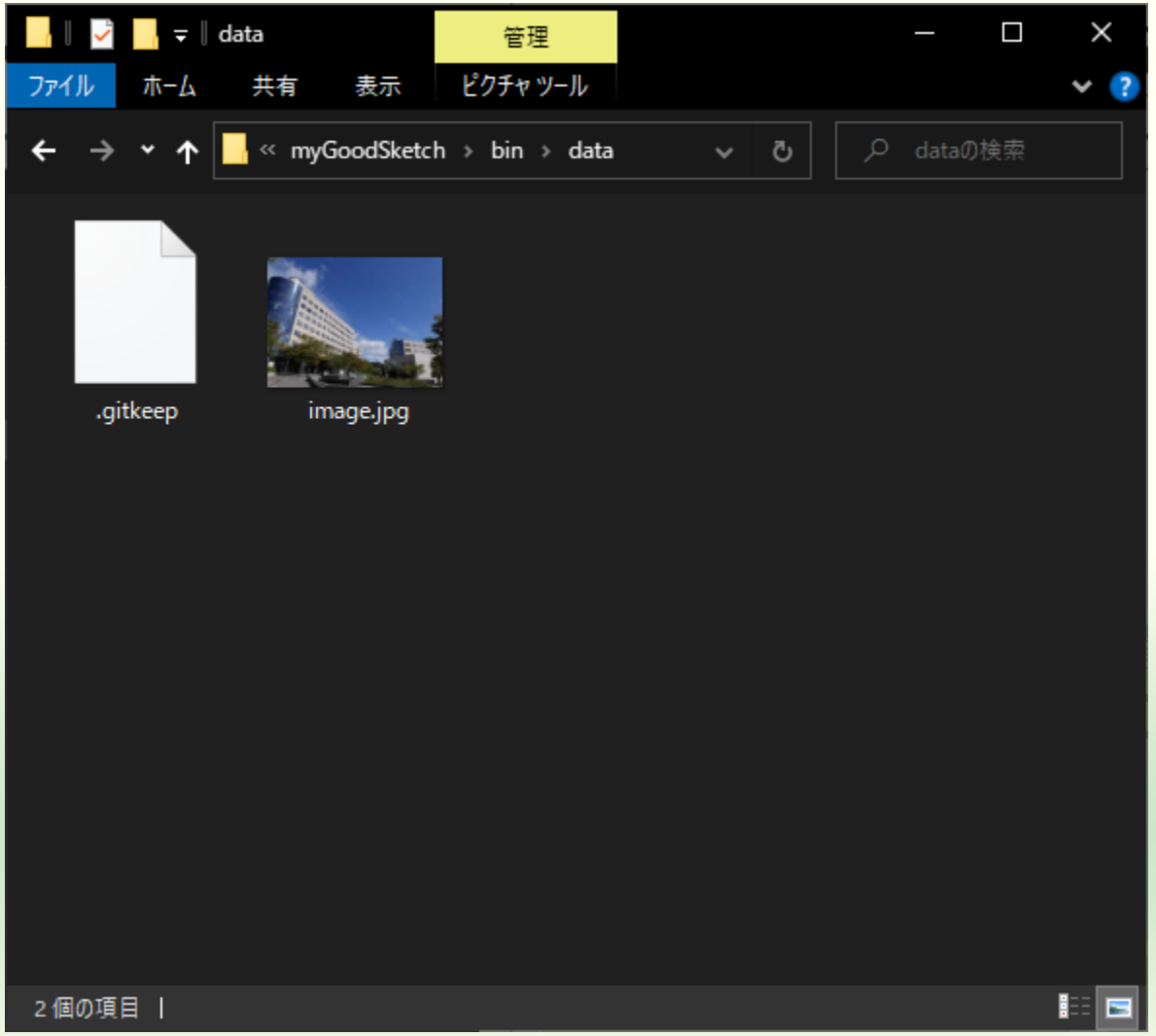
ofApp.cpp の setup() で image に画像を読み込む

```
#include "ofApp.h"

//--------------------------------------------------
void ofApp::setup(){
   （途中略）
  video.setDeviceID(0);
  video.setup(320, 240);
  saved = color = video.getPixels();
  select = '0';
  image.load("image.jpg");
  image.resize(video.getWidth(),
    video.getHeight());
}

（以下略）
```

- image.load("image.jpg");
  - プロジェクトのフォルダの bin の data の中にある image.jpg という画像ファイルを image に読み込む
- "image.jpg" は画像ファイル名
  - JPEG, PNG, GIF 画像が読み込める
- image.resize(video.getWidth(), video.getHeight());
  - 読み込んだ画像のサイズを入力映像のサイズに合わせる

画像は bin フォルダの中の data に配置する





入力映像と背景画像の差の絶対値が閾値以下の領域に別の画像を表示する

- 入力映像と背景画像の差の絶対値が閾値以下の画素を黒にする代わりに別の画像の同じ位置の画素の色を表示する
- ofImage の画像から画素データを取り出すには getPixels() メソッドを使う
  - ofPixels &back = image.getPixels();
    - back は入力映像と同じサイズにした画像 image の画素データを参照する

課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **5-6.png** というファイル名で保存し、Moodle の第 5 回課題にアップロードしてください

テクスチャマッピング

入力映像を３Ｄオブジェクトに貼り付ける

ofApp クラスに箱とライト、カメラのメンバ変数を追加する

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{
  ofVideoGrabber video;
  ofPixels color, saved;
  ofTexture texture;
  ofImage image;
  ofBoxPrimitive box;
  ofLight light;
  ofEasyCam camera;
  int select;

public:
  void setup();
  void update();
  void draw();

 (以下略)
```

- ofBoxPrimitive は箱のクラス
- ofLight はライトのクラス
- ofEasyCam はマウスで制御できるカメラのクラス

カメラとライトの設定を行う

```
#include "ofApp.h"

//--------------------------------------------
void ofApp::setup(){
   (途中略)
  video.setDeviceID(0);
  video.setup(320, 240);
  saved = color = video.getPixels();
  select = '0';
  image.load("image.jpg");
  image.resize(video.getWidth(),
    video.getHeight());
  camera.setPosition(0.0f, 0.0f, 200.0f);
  light.setPosition(60.0f, 80.0f, 100.0f);
  light.enable();
}
```

- camera.setPosition(0.0f, 0.0f, 200.0f);
  - カメラ camera の位置を設定する
- light.setPosition(60.0f, 80.0f, 100.0f);
  - ライト light の位置を設定する
- light.enable();
  - ライト light を有効にする

表示する画像を箱の表面のサイズに合わせる

```
//--------------------------------------------
void ofApp::update(){
  video.update();
  if (video.isFrameNew()){
    ofPixels &input = video.getPixels();
    switch (select) {
       (途中略)
    default:
      break;
    }
    texture.loadData(color);
    box.mapTexCoordsFromTexture(texture);
  }
}
```

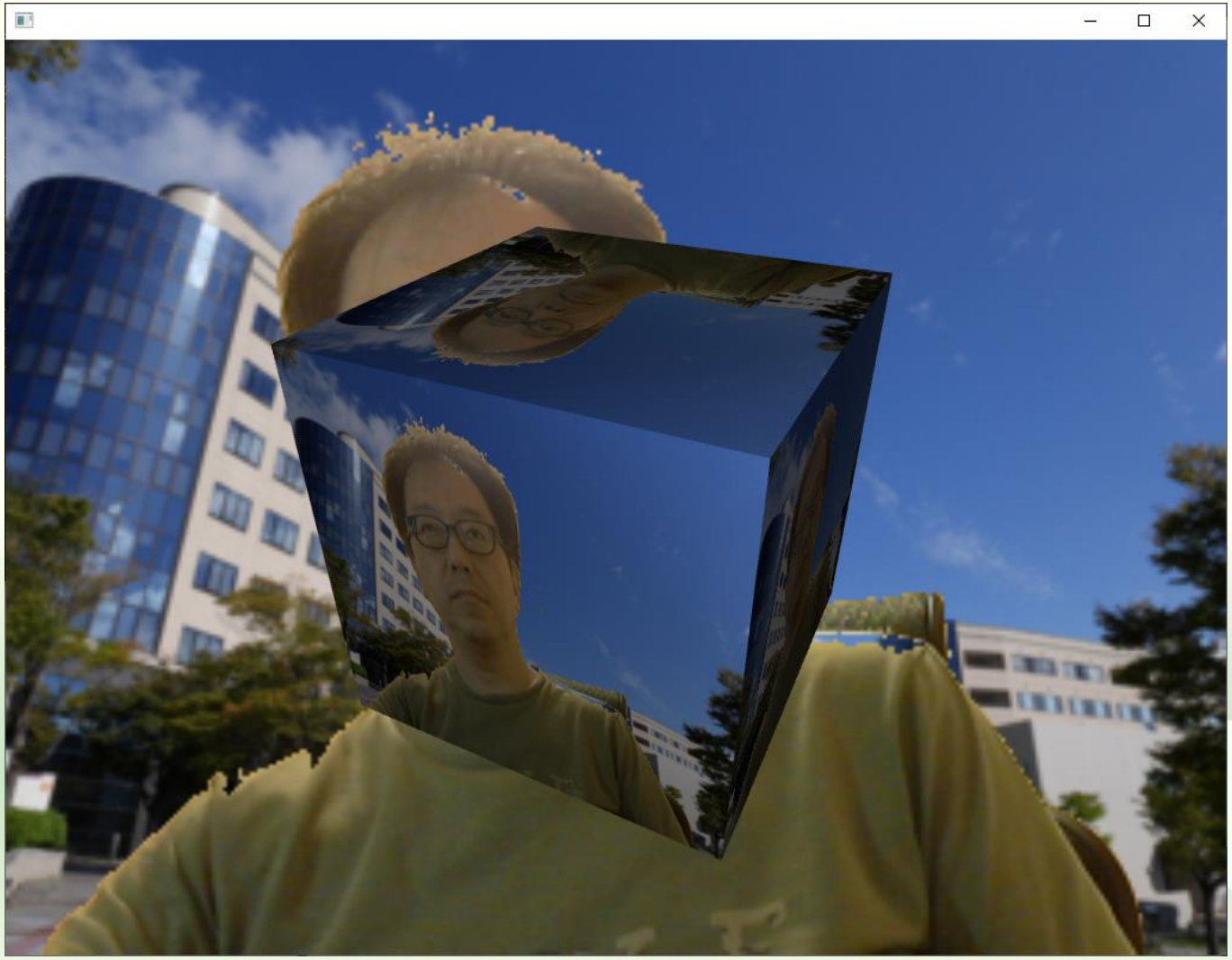
- box.mapTexCoordsFromTexture(texture);
  - box の表面に texture をぴったり貼り付ける設定を行う

表示している画像に重ねて箱を描く

```
//--------------------------------------------------
void ofApp::draw(){
  ofDisableLighting();
  texture.draw(0, 0, ofGetWidth(), ofGetHeight());
  ofEnableLighting();
  camera.begin();
  ofEnableDepthTest();
  texture.bind();
  box.draw();
  texture.unbind();
  ofDisableDepthTest();
  camera.end();
}
```

- ofEnableLighting();～ ofDisableLighting();
  - この間の 3D CG の描画で陰影付けが有効になる
- ofEnableDepthTest();～ ofDisableDepthTest();
  - この間の 3D CG の描画で隠面消去処理を有効にする
- texture.bind();～ texture.unbind();
  - この間に描画する 3D CG の図形で texture のマッピング（貼り付け）を有効にする

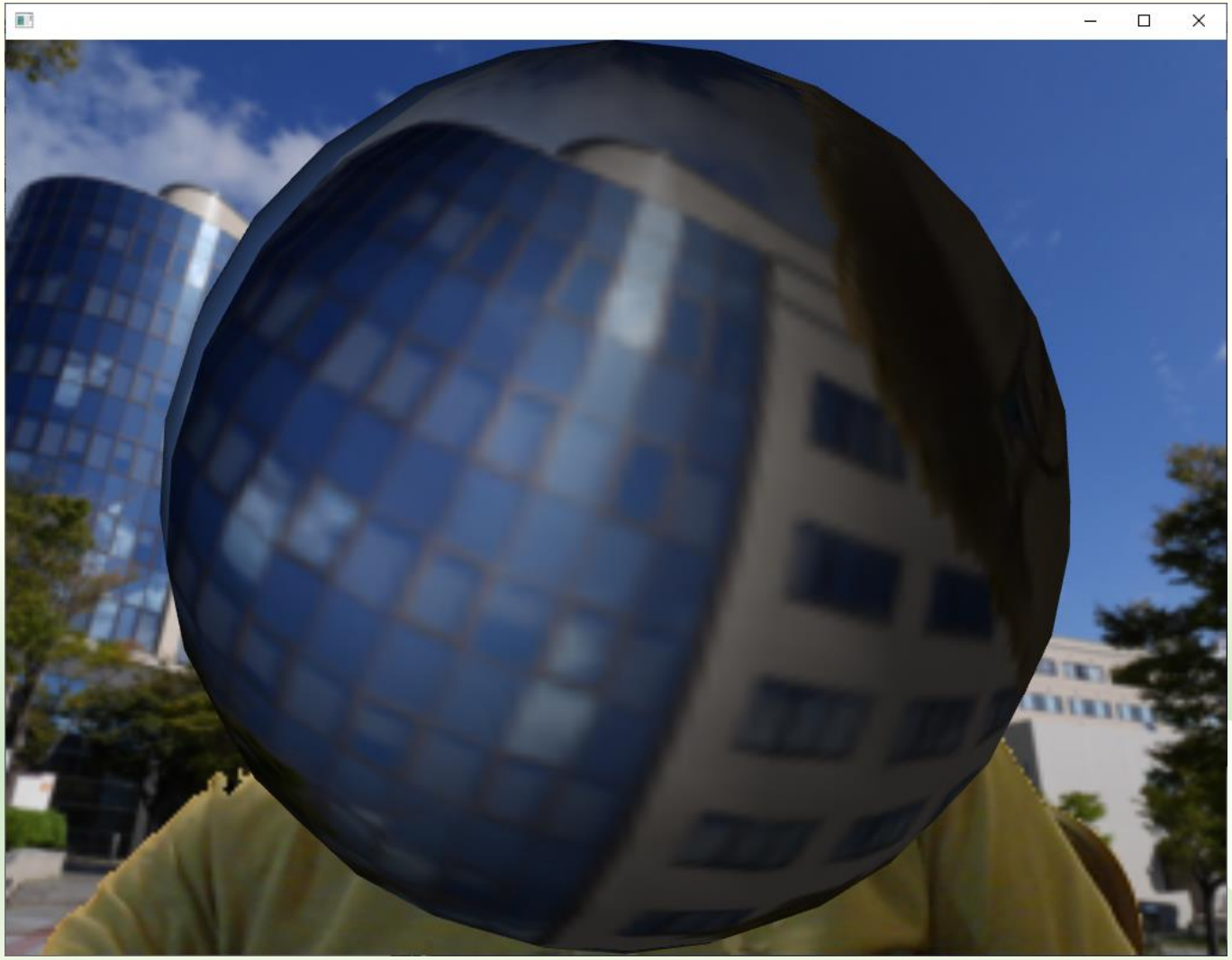
映像が箱の表面にマッピングされる



課題５－７

箱の代わりに映像を球にマッピングする

球にマッピングした結果



箱の代わりに映像を球にマッピングしなさい

- ofBoxPrimitive を ofSpherePrimitive に替えるだけでできる
  - 球のサイズが小さいので setRadius() メソッドで設定してください
  - 光源の位置は変更したほうがいいかもしれません
- 余裕があれば ofConePrimitive, ofCylinderPrimitive, ofIcoSpherePrimitive, ofPlanePrimitive でも試してみてください

課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **5-7.png** というファイル名で保存し、Moodle の第 5 回課題にアップロードしてください
- ソースプログラム **ofApp.h** と **ofApp.cpp** を Moodle の第 5 回課題にアップロードしてください

時間の余った人向け課題

第３回の「階層構造」や「課題３－４」の球の部分に映像を貼り付けてみてください

補足

グレースケール化について

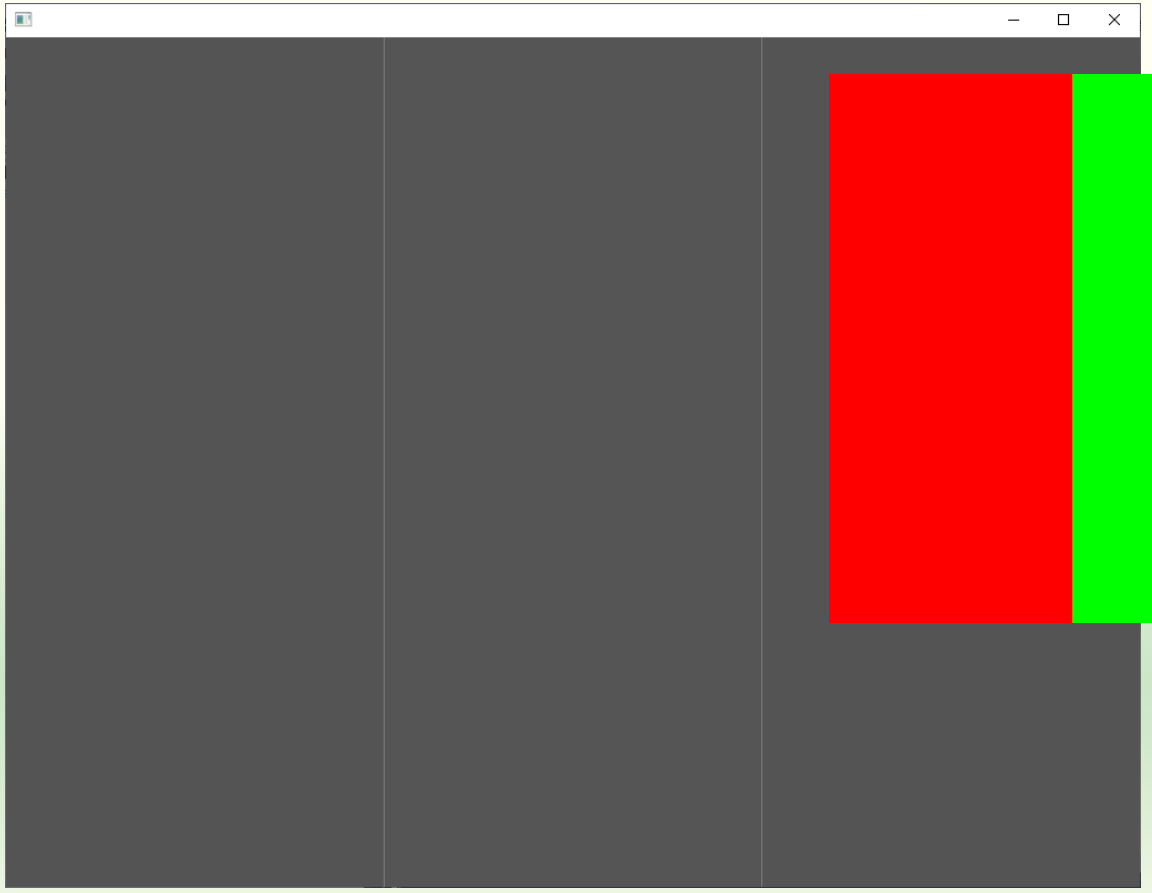
グレースケール化の方法による違い (1)

RGB 平均

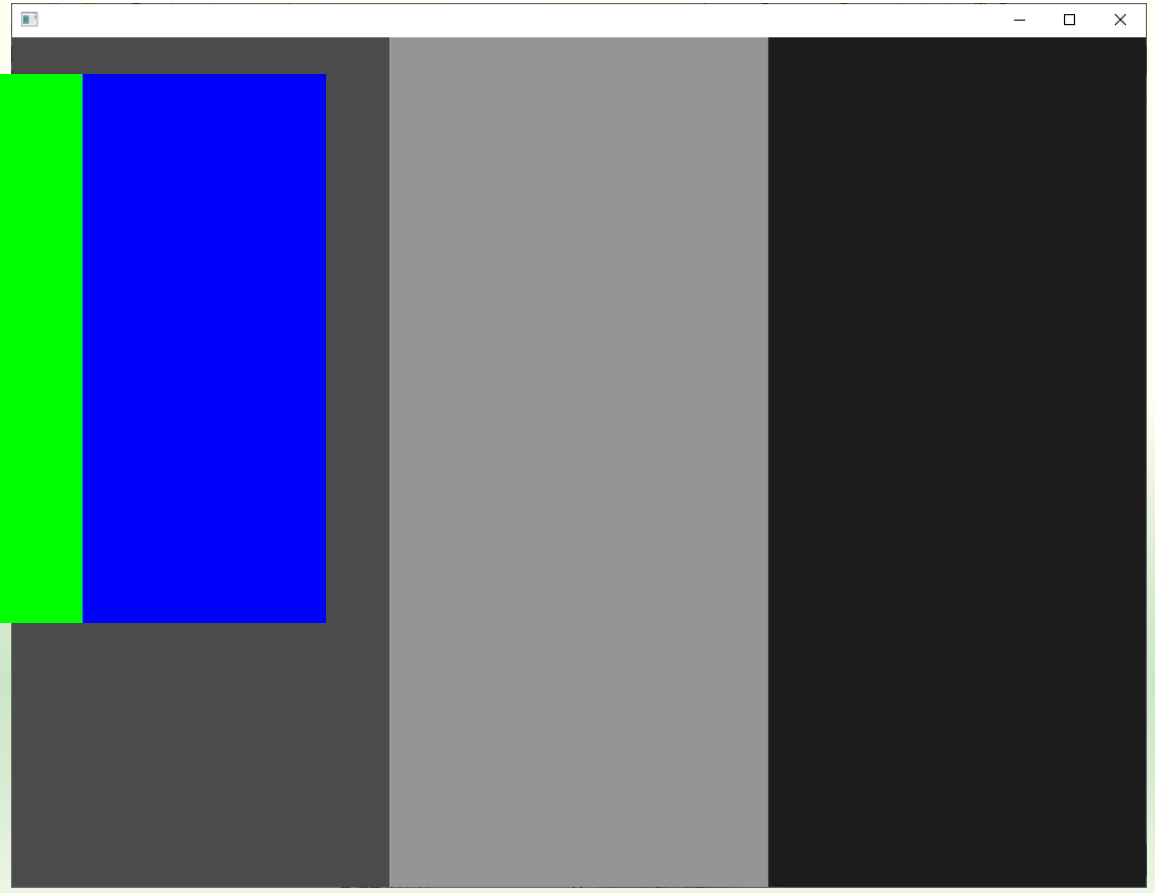
ITU-R Rec BT.601

グレースケール化の方法による違い (2)

RGB 平均



ITU-R Rec BT.601

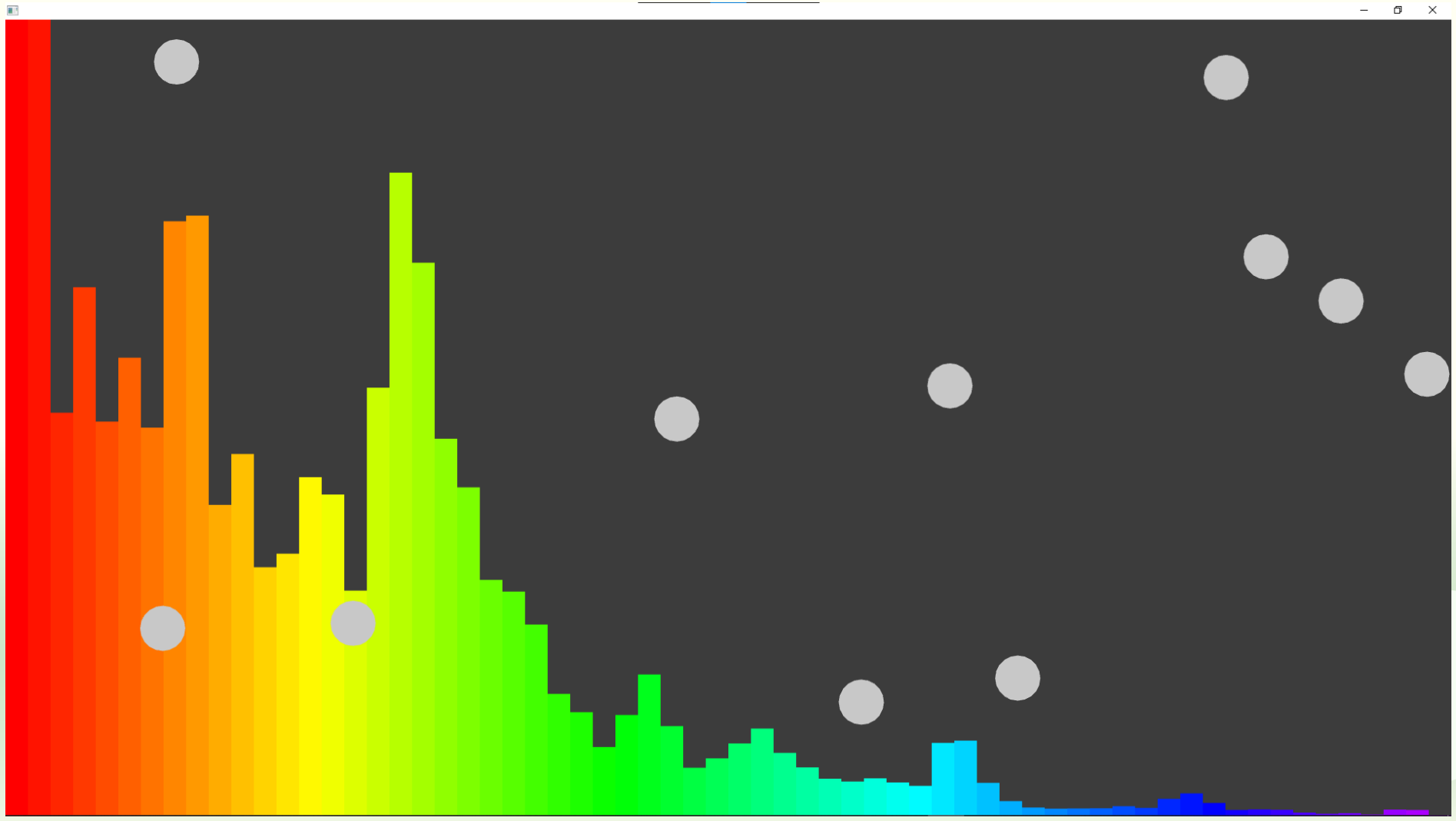


メディアプログラミング演習

第6回

今回の課題の締め切りは本日１８時です

本日はサウンドを使ったアプリの作成

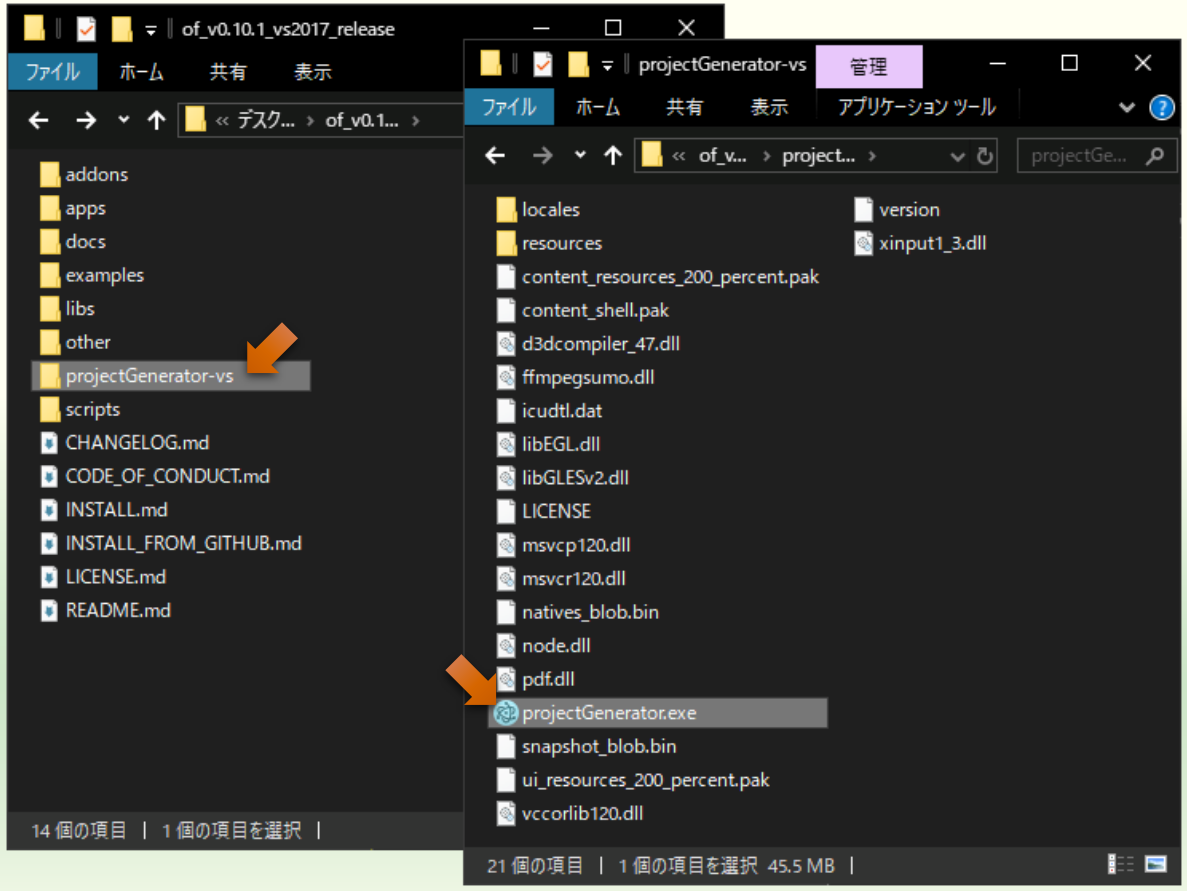


準備

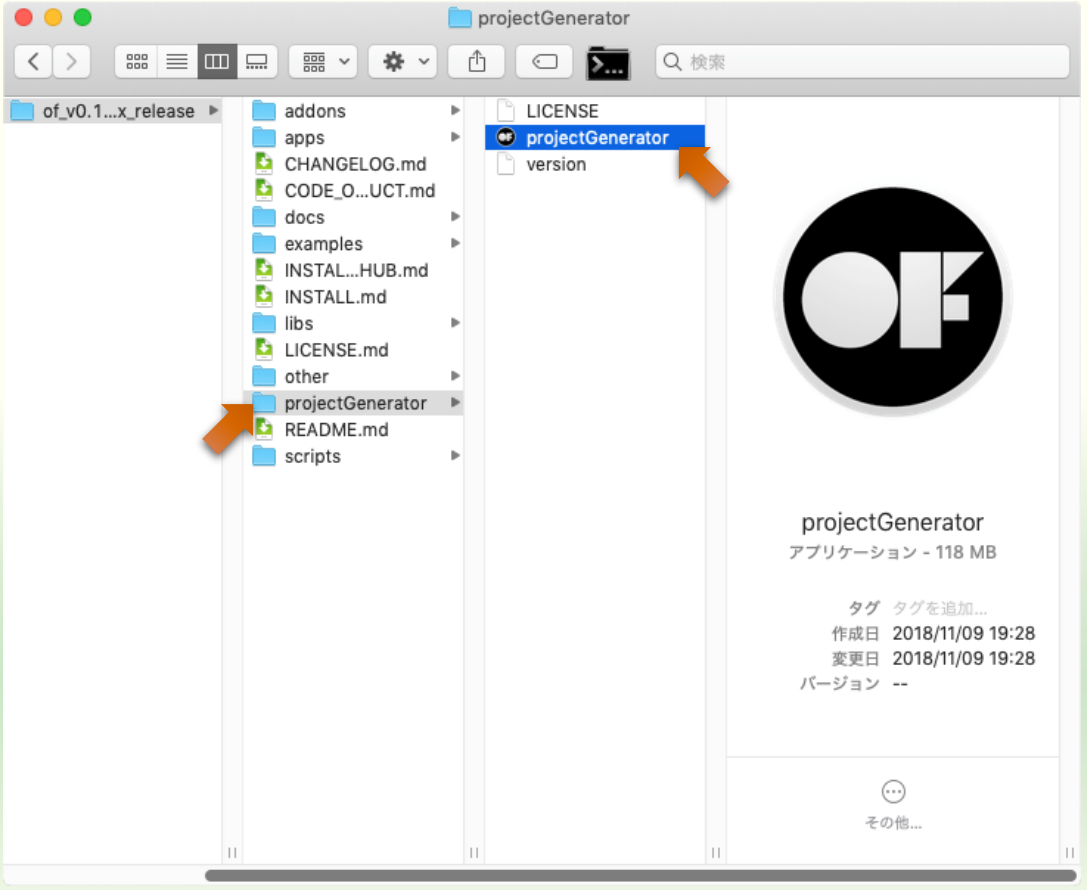
プロジェクトの作成

projectGenerator を起動する

windows 版のパッケージ



macOS 版のパッケージ

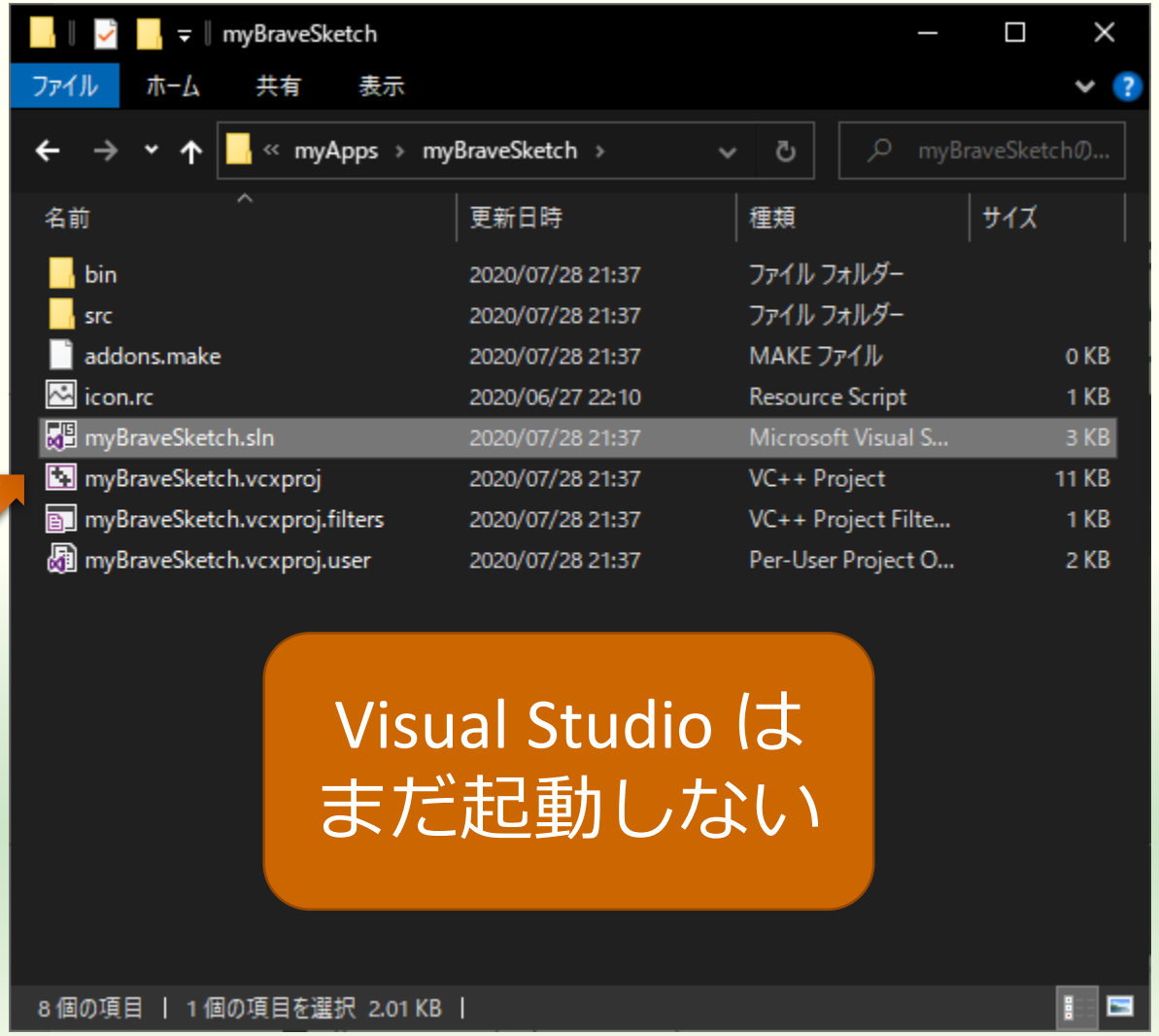


空のプロジェクトの作成

- Project name:
  - 作成するプロジェクト（プログラム）の名前
- Project path:
  - 作成するプロジェクトのファイルを置く場所
  - openFrameworks のパッケージを展開した場所の中の apps¥myApps

プロジェクトの作成成功

クリックして開く



Visual Studio はまだ起動しない

samples.zip のダウンロードと展開

［第６回］音声

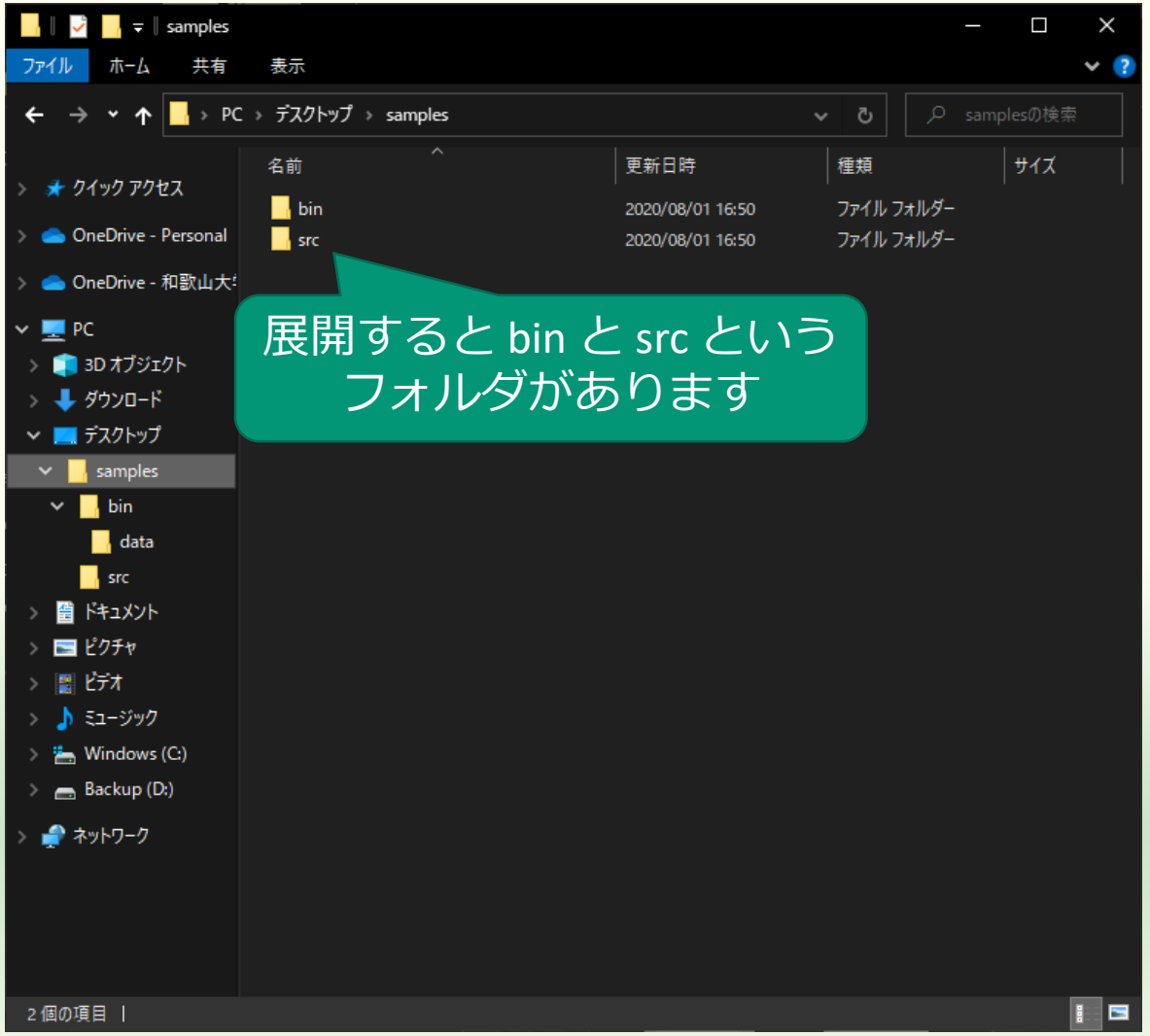
第６回資料

samples.zip

第６回課題

今回の課題の締め切りは本日１８時です

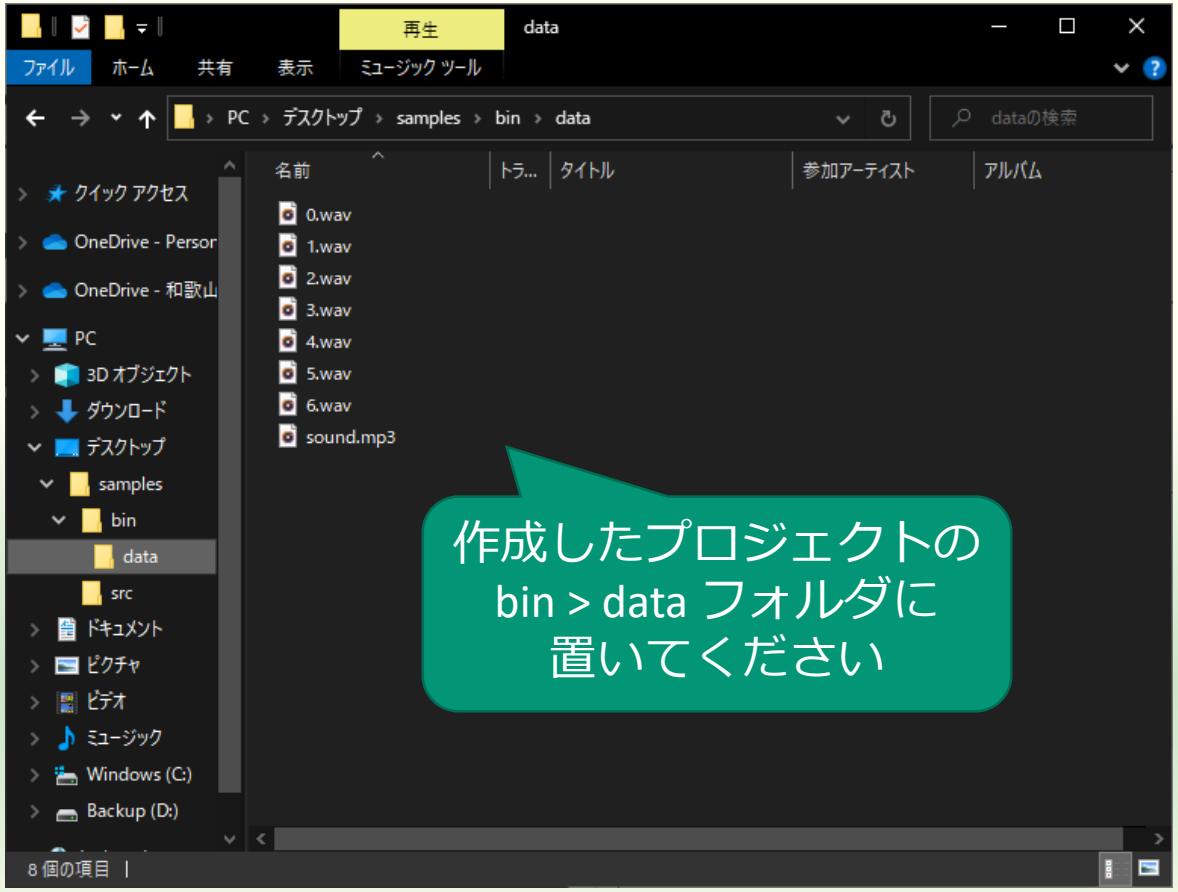
Moodle から samples.zip をダウンロードしてください



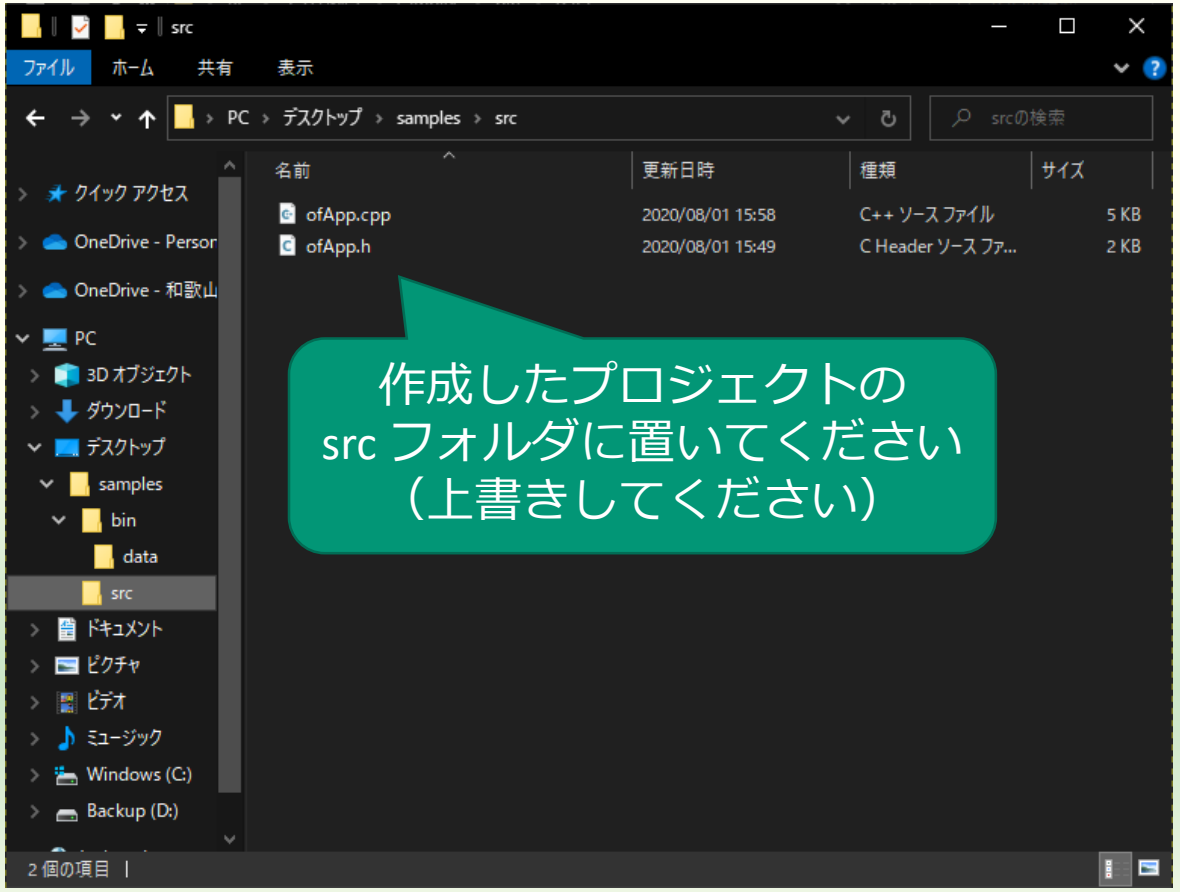
展開すると bin と src というフォルダがあります

ファイルの配置

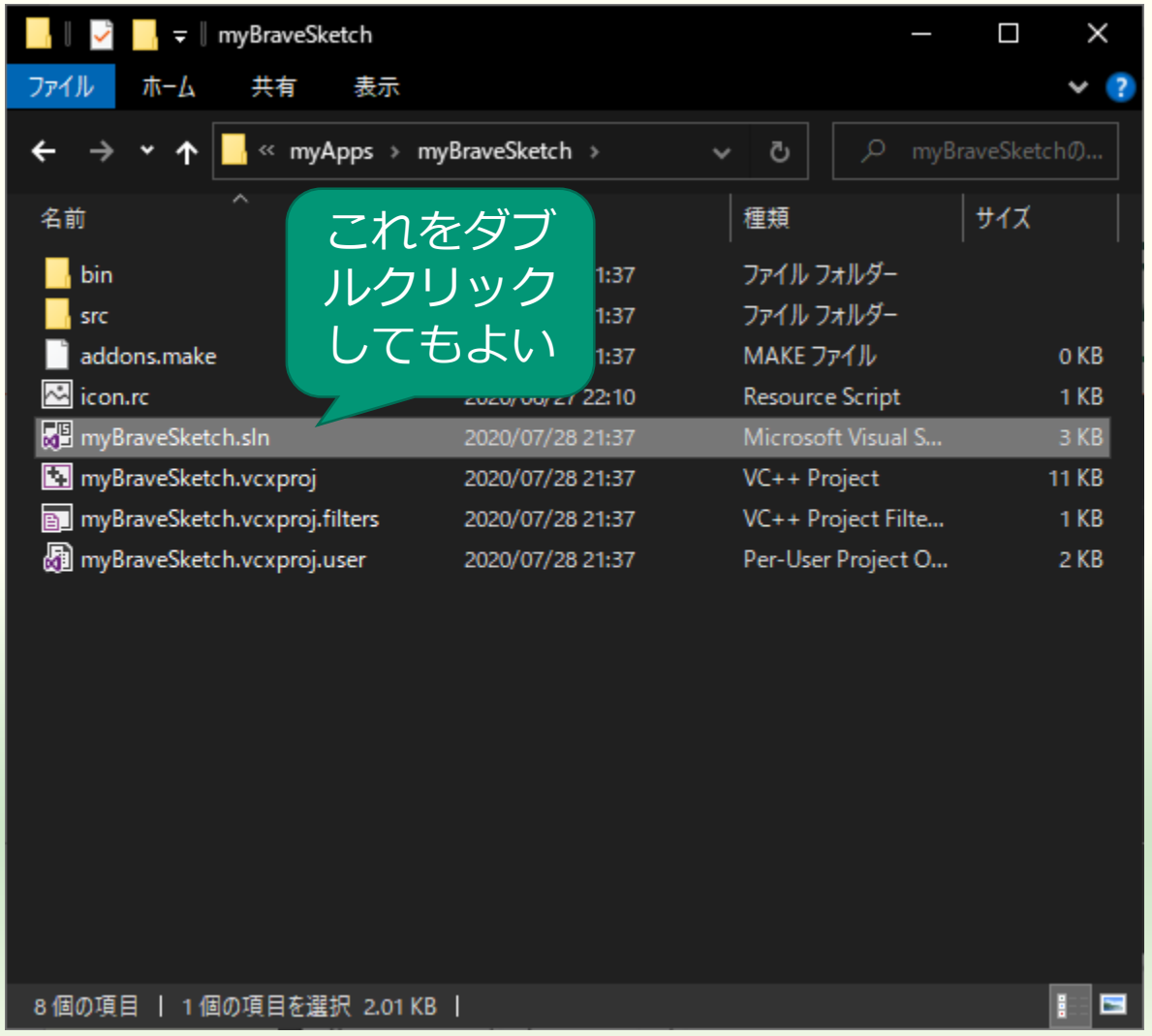
bin > data の内容



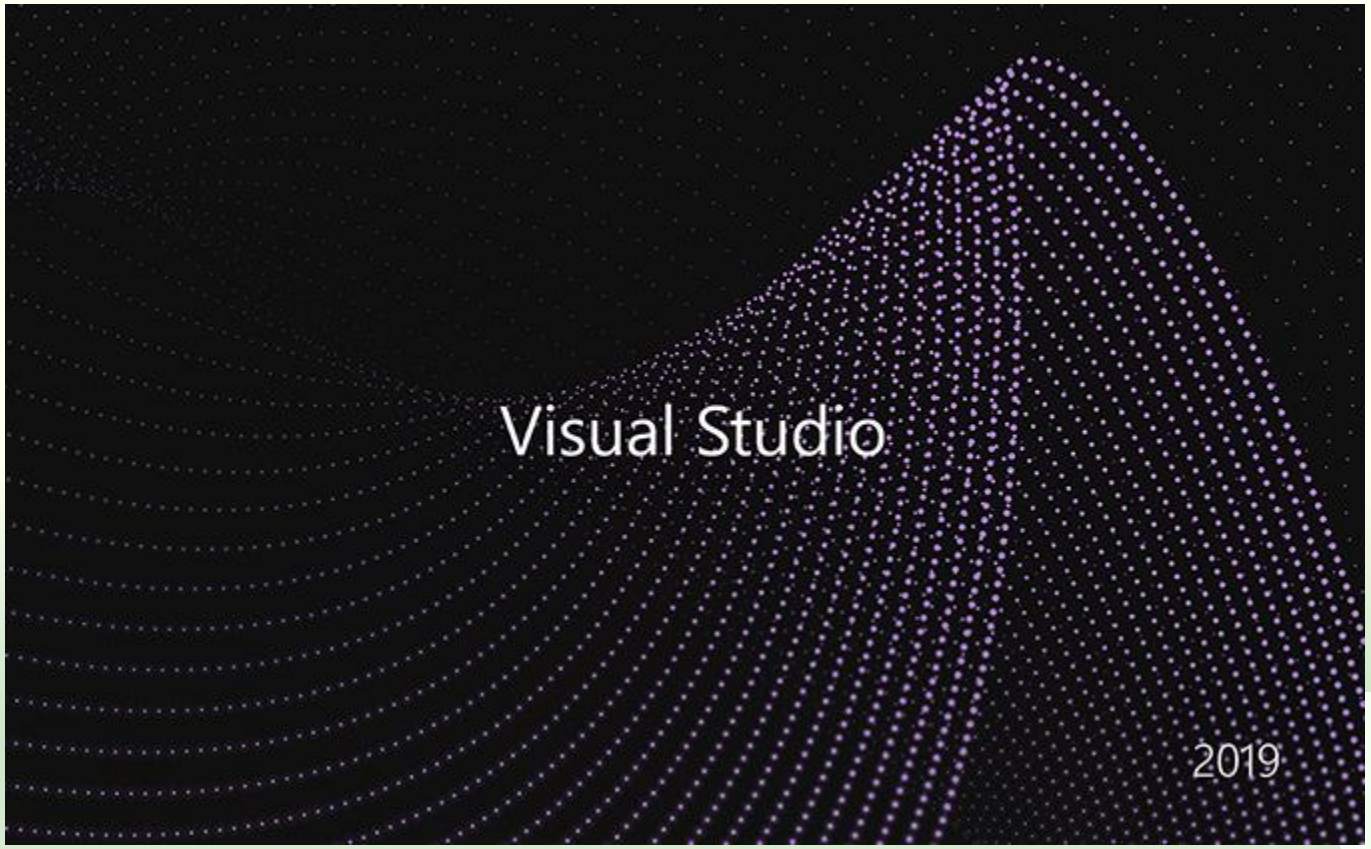
src の内容



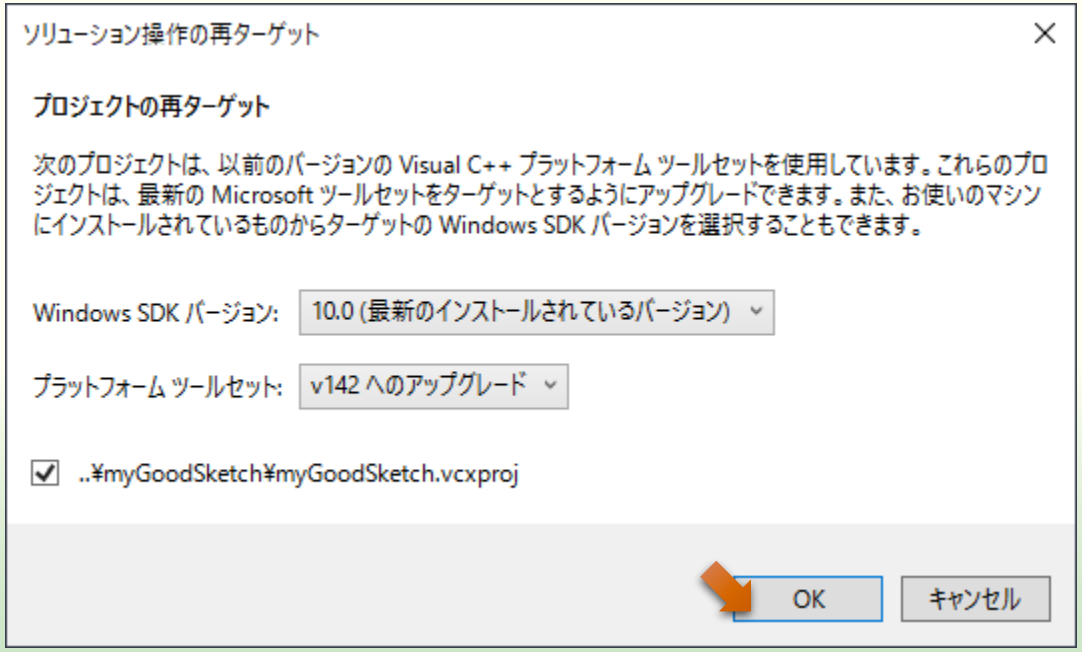
ソリューションファイルを開く



Visual Studio 2019 が起動する

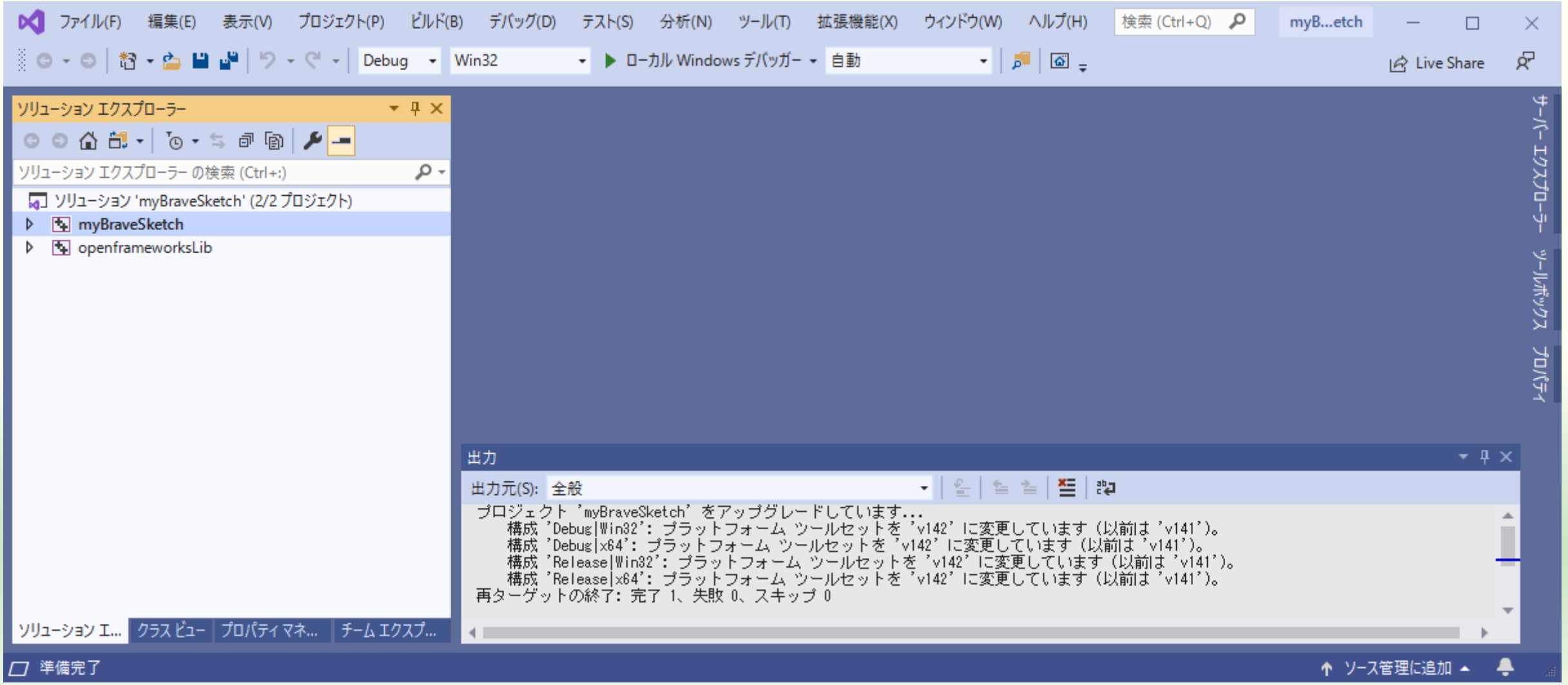


ソリューションの再ターゲット

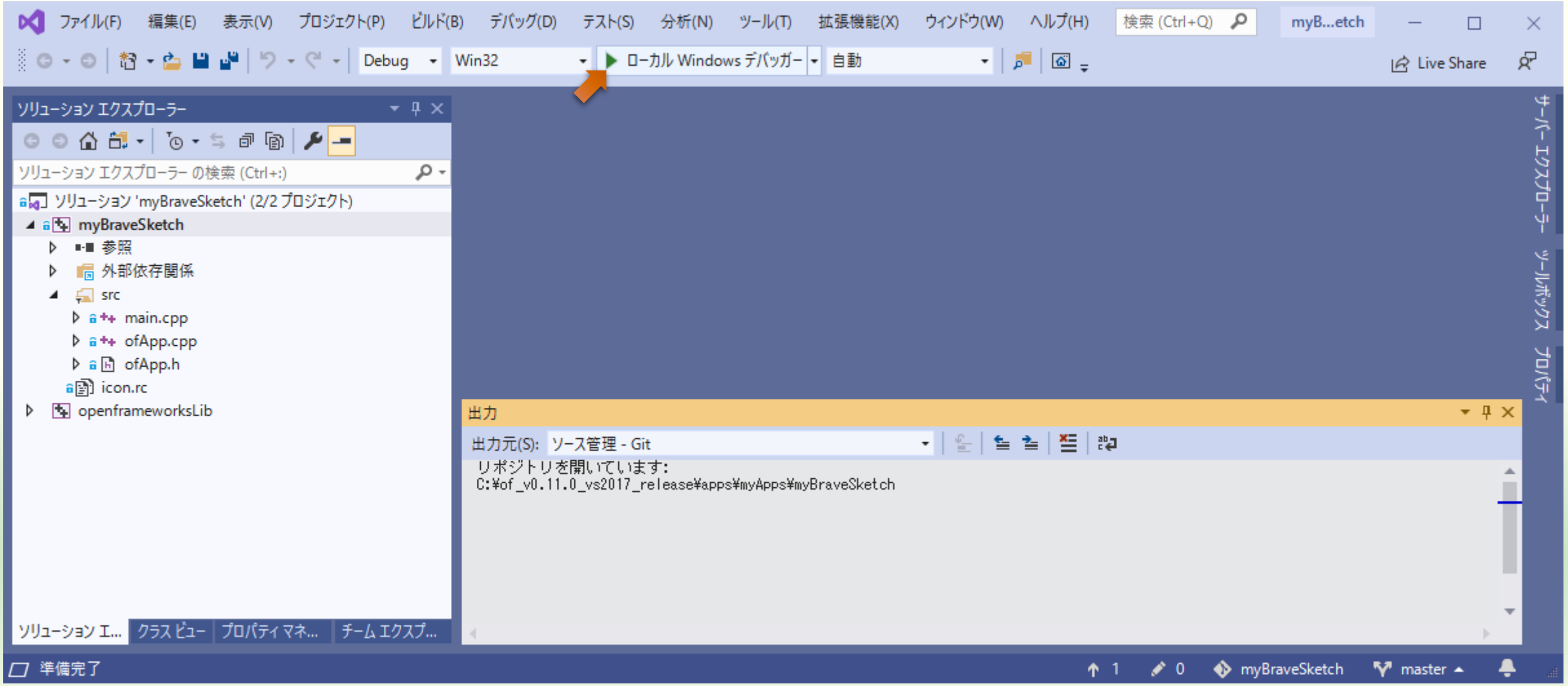


Visual Studio は頻繁に更新しているので皆さんがお使いの Visual Studio SDK のバージョンと合わない場合がある

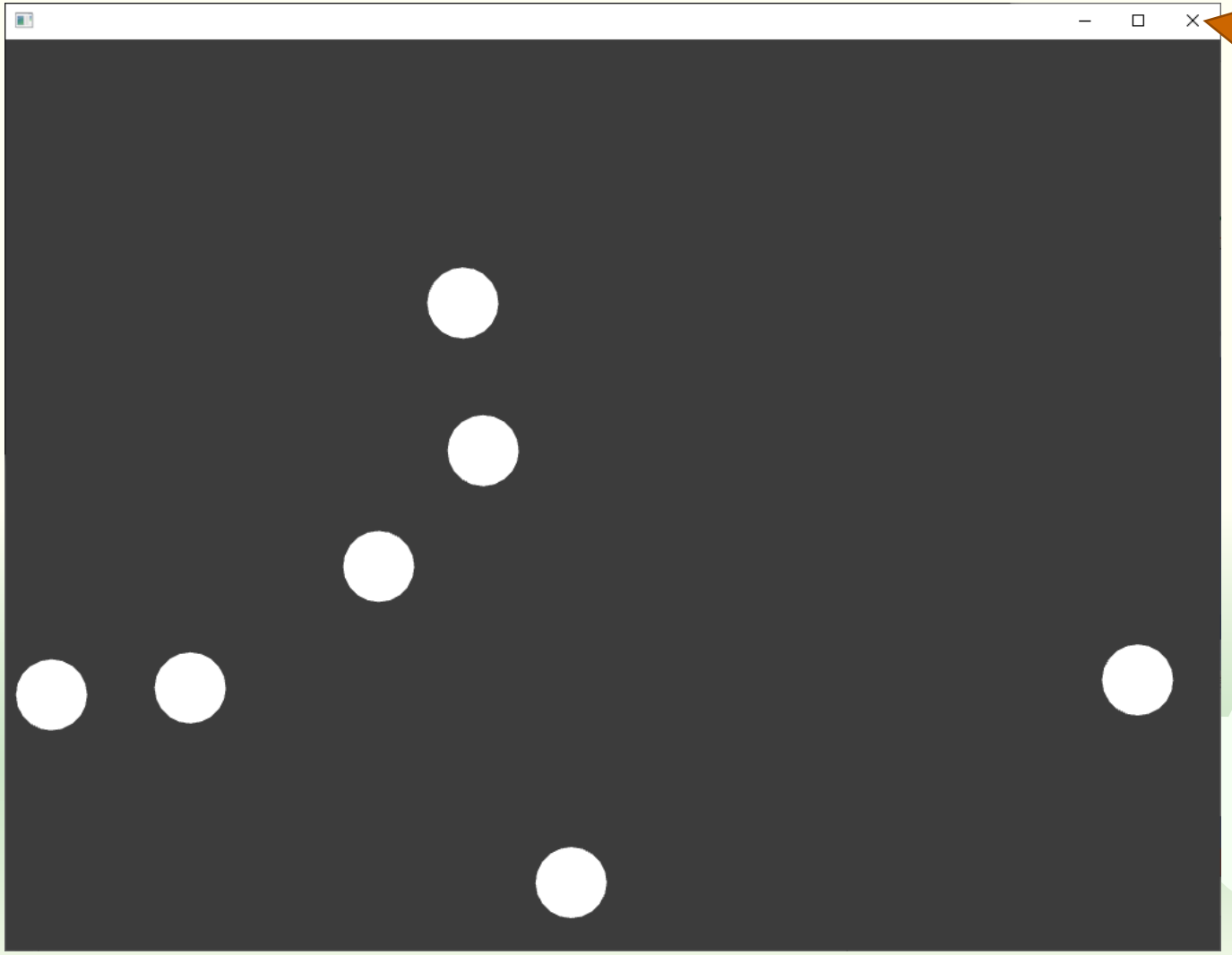
Visual Studio 起動



ビルドと実行


ファイル(F)
編集(E)
表示(V)
プロジェクト(P)
ビルド(B)
デバッグ(D)
テスト(S)
分析(N)
ツール(T)
拡張機能(X)
ウィンドウ(W)
ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myBraveSketch' (2/2 プロジェクト)
myBraveSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
出力
出力元(S): ソース管理 - Git
リポジトリを開いています:
C:¥of_v0.11.0_vs2017_release¥apps¥myApps¥myBraveSketch
サーバー エクスプローラー
ツールボックス
プロパティ
ソリューション エ...
クラス ビュー
プロパティ マネ...
チーム エクスプ...
準備完了
1
0
myBraveSketch
master

課題２－５を複数の円に対応したもの



動作を確認したら止める

ofApp.h に Circle クラスの定義を追加している

```
#pragma once

#include "ofMain.h"

using namespace glm;

class Circle{

public:
  vec2 position;
  vec2 velocity;
  float radius;
  ofColor color;
};

class ofApp : public ofBaseApp{
  vector<Circle> circles;
  vec2 startPosition;
  float startTime;

   (以下略)
```

ひとまとまりのデータとして扱う

position など円の４つのデータを１つの vector に保持できる

- class Circle { … };
  - Circle というクラスの定義
    - メンバ変数: position（位置）velocity（速度）radius（半径）color（色）
    - public: 以降にあるメンバ変数・メンバ関数はメンバ関数以外から参照できる
- vector<Circle> circles;
  - Circle クラスの vector として circles を宣言
    - 個々の要素は position, velocity, radius, color のメンバを持つ

マウスボタンが押されたときに円を生成する

```
//--------------------------------------------------------------
void ofApp::mousePressed(int x, int y, int button){
  if (button == 0){                                  // もし押されたのが左ボタンなら
    startTime = ofGetElapsedTimef();                 // 押された時刻（経過時間）を startTime に保存する
    startPosition = vec2{ x, y };                    // 押された位置を startPosition に保存する
    Circle circle;                                   // Circle クラスのオブジェクト circle を一つ生成する
    circle.position = startPosition;                 // circle の位置 position をボタンを押した位置を設定する
    circle.velocity = vec2{ 0.0f, 0.0f };            // circle の速度 velocity を 0 にする
    circle.radius = 30.0f;                           // circle の半径を 30 にする
    circle.color = ofColor{ 200, 200, 200 };         // circle の色を R:200, G:200, B:200 にする
    circles.push_back(circle);                       // vector の circles に circle を追加する
  }
}
```

ofApp::mousePressed() は Circle クラスのメンバではないが、position、velocity、radius、color は public: メンバなので、circle.position のように “.”（ドット演算子）を使ってメンバにアクセスできる

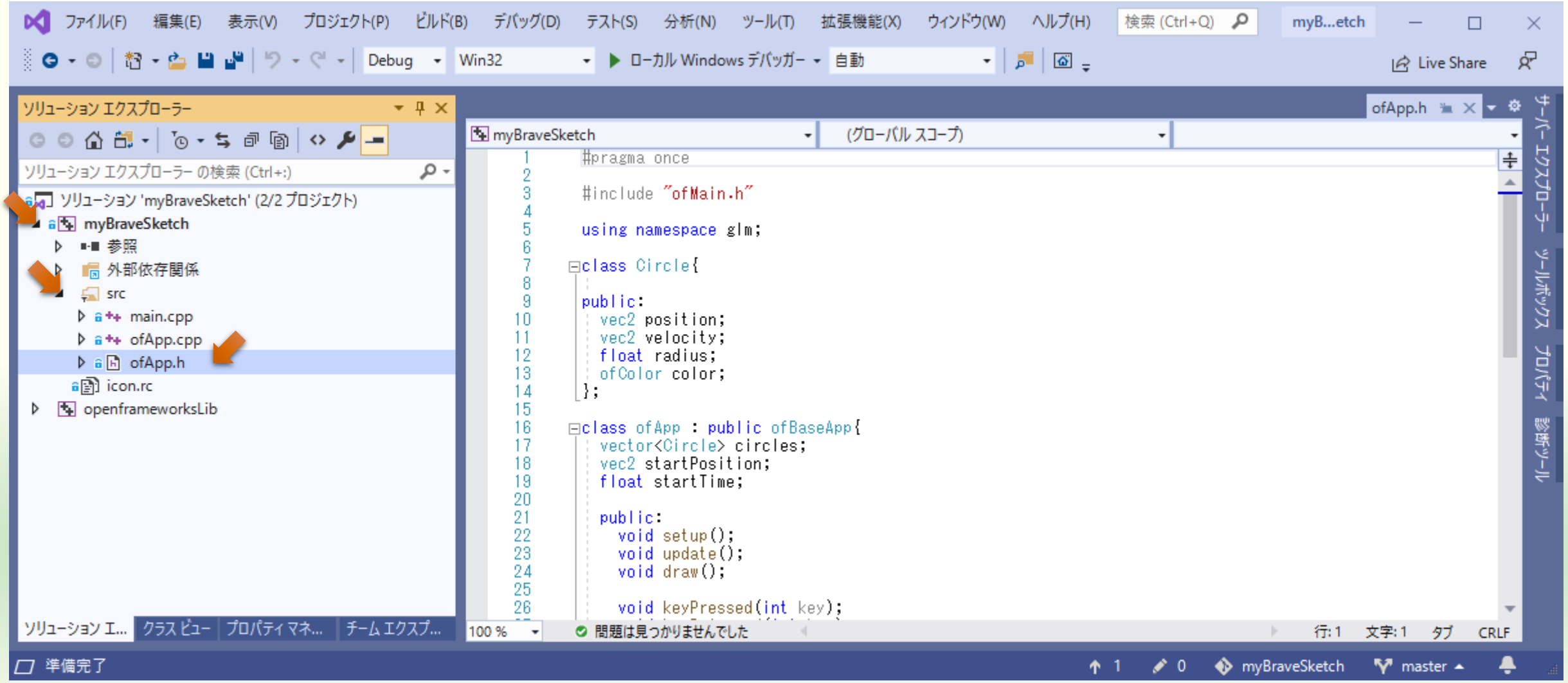
この場合は初期化を用いて以下のように書くこともできる

```
Circle circle{ startPosition, vec2{ 0.0f, 0.0f }, 30.0f, ofColor{ 200, 200, 200 } };
```

音声の再生

音声ファイルの読み込み

ofApp.h を開く

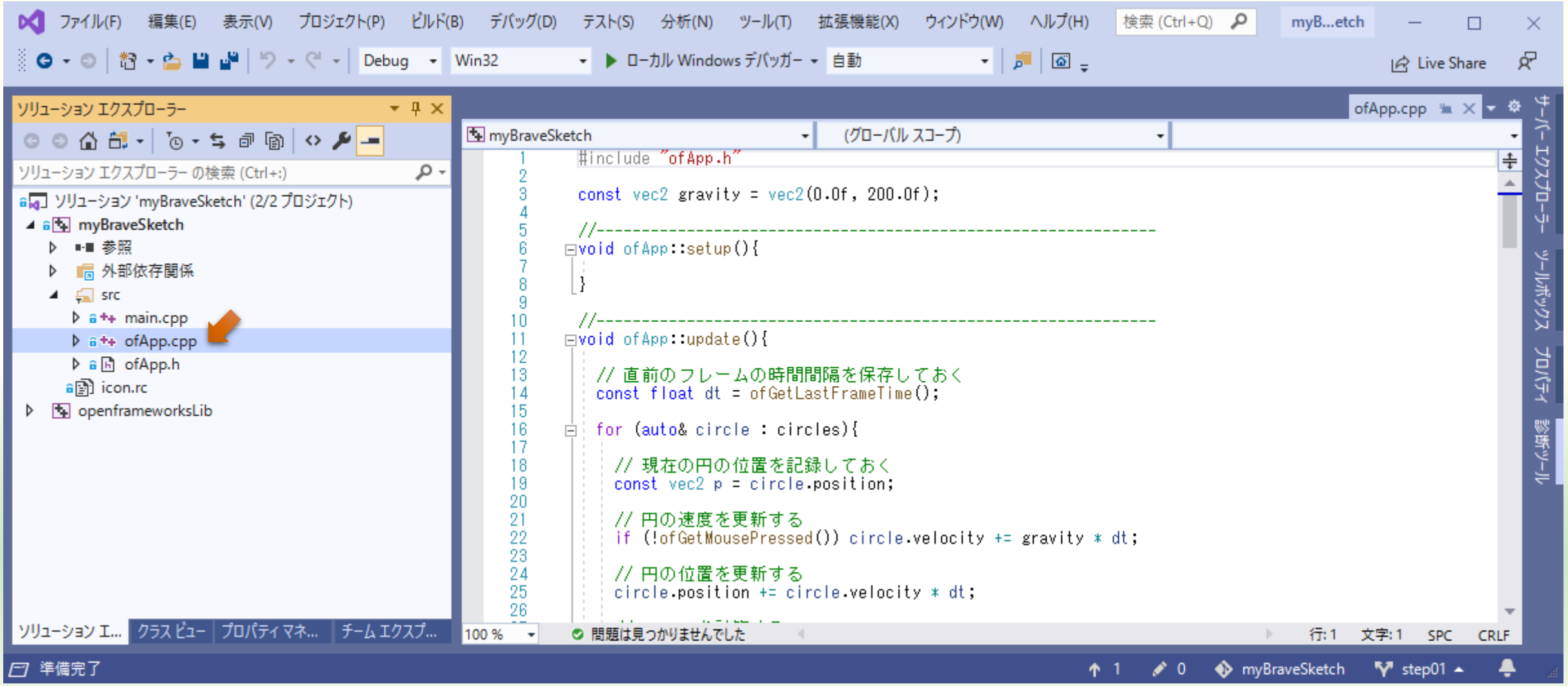

ファイル(F)
編集(E)
表示(V)
プロジェクト(P)
ビルド(B)
デバッグ(D)
テスト(S)
分析(N)
ツール(T)
拡張機能(X)
ウィンドウ(W)
ヘルプ(H)
検索 (Ctrl+Q)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myBraveSketch' (2/2 プロジェクト)
myBraveSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.h
myBraveSketch
(グローバル スコープ)
#pragma once
#include "ofMain.h"
using namespace glm;
class Circle{
public:
vec2 position;
vec2 velocity;
float radius;
ofColor color;
};
class ofApp : public ofBaseApp{
vector<Circle> circles;
vec2 startPosition;
float startTime;
public:
void setup();
void update();
void draw();
void keyPressed(int key);
100 %
問題は見つかりませんでした
行: 1
文字: 1
タブ
CRLF
準備完了
1
0
myBraveSketch
master

ofAppクラスに音声再生のメンバ変数を追加する

```
（以上略）

class ofApp : public ofBaseApp{
  vector<Circle> circles;
  vec2 startPosition;
  float startTime;
  ofSoundPlayer sound;

public:
  void setup();
  void update();
  void draw();

（以下略）
```

- ofSoundPlayer
  - サウンドファイルの読み込みと再生を行うクラス
    - ボリューム、パン、スピード、シーク、マルチプレイのコントロールが可能
    - プラットフォームごとに異なるサウンド再生機能に対して統一したインターフェイスを与えたもの

ofApp.cpp を開く



サウンドファイルを読み込む

```
#include "ofApp.h"

const vec2 gravity = vec2(0.0f, 200.0f);

//--------------------------------------------
void ofApp::setup(){
  sound.load("sound.mp3");
}

//--------------------------------------------
void ofApp::update(){
  ofSoundUpdate();

   （途中略）
}

（以下略）
```

自分で音声ファイルを用意しても構わない

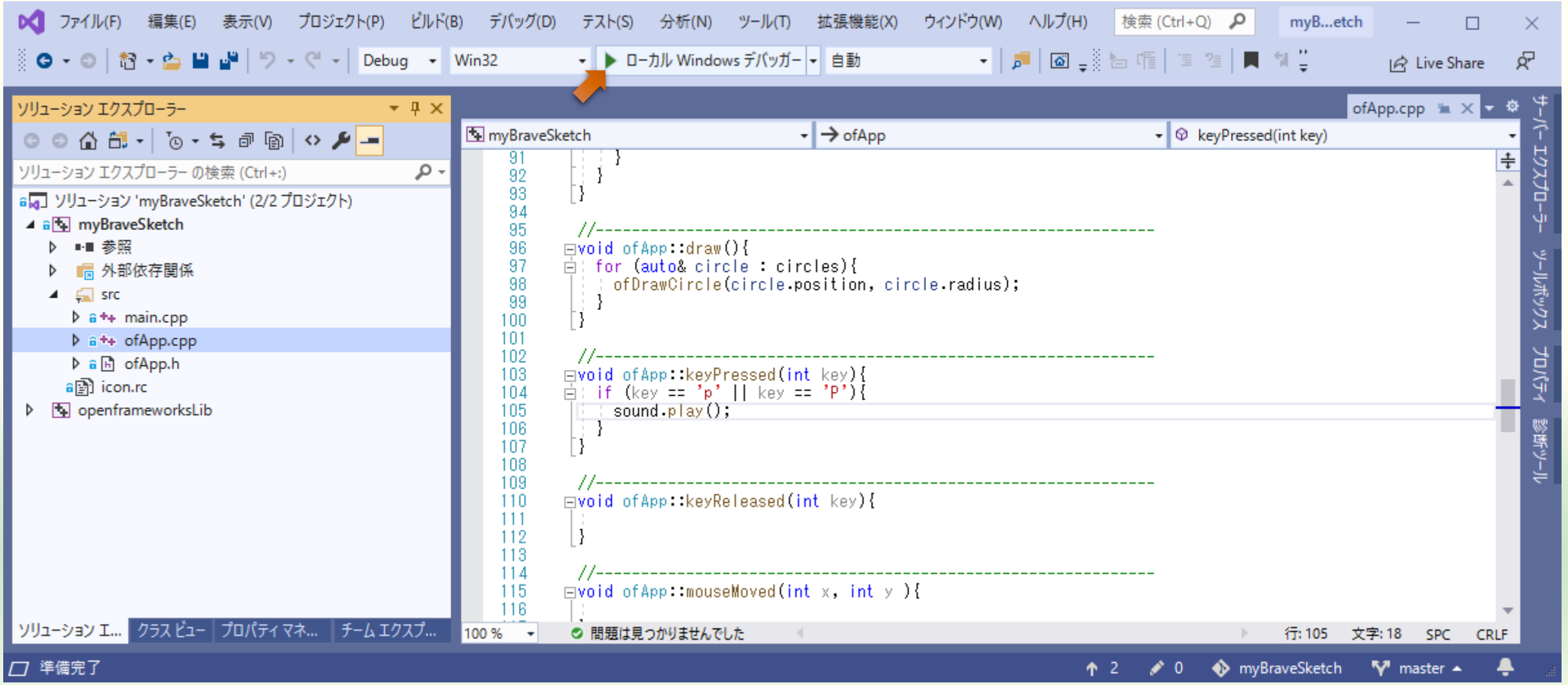
- sound.load("sound.mp3");
  - プロジェクトのフォルダの bin の data の中にある sound.mp3 という音声ファイルを sound に読み込む
- “sound.mp3” は音声ファイル名
  - どんな音声ファイルが読み込めるかはプラットフォーム（Windows, macOS, Linux, …）依存
- ofSoundUpdate();
  - 音声エンジンの更新、毎フレーム呼び出す必要がある

‘p’ または ‘P’ キーのタイプでサウンドファイルを再生する

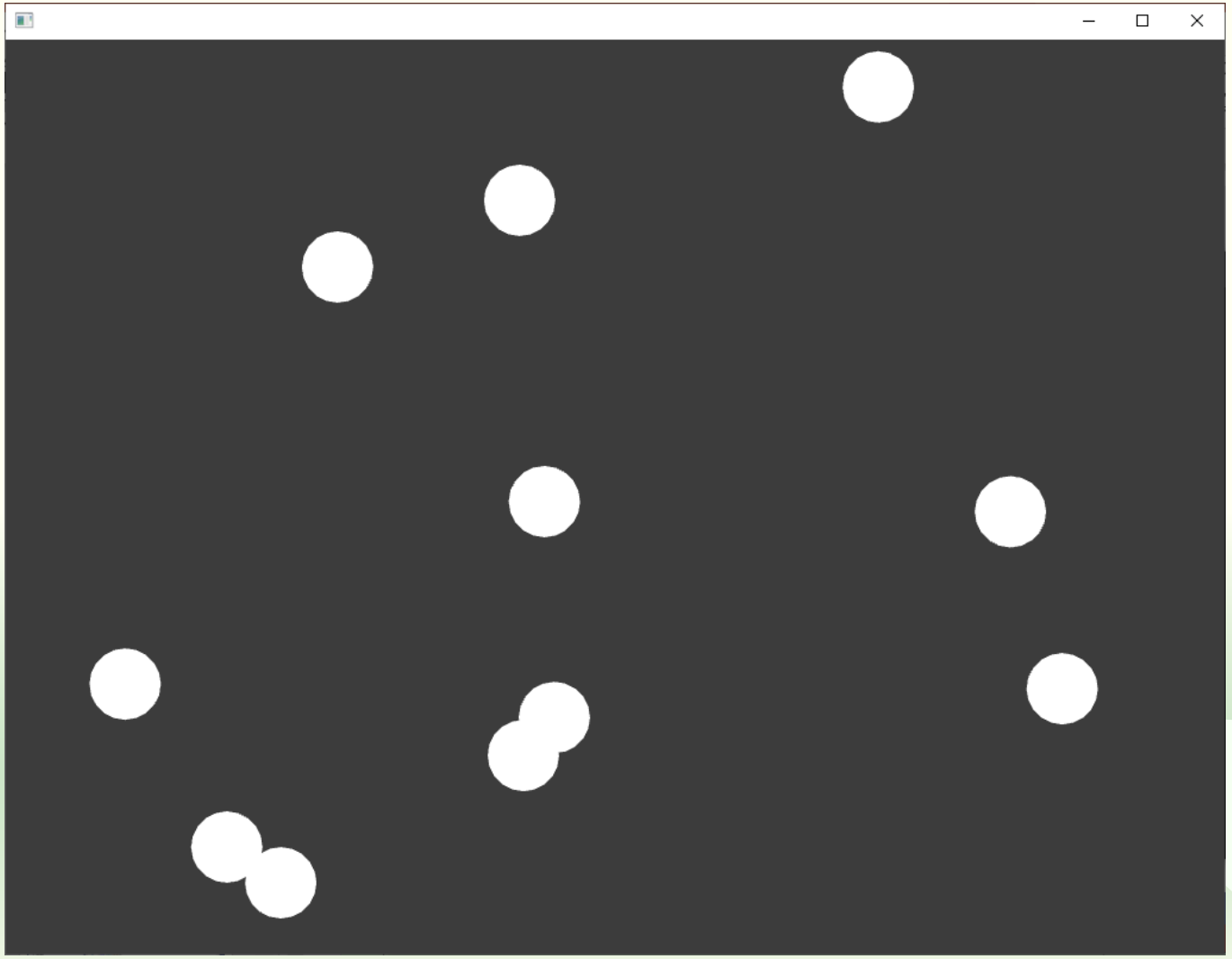
```
（以上略）

//--------------------------------------------
void ofApp::draw(){
   （途中略）
}

//--------------------------------------------
void ofApp::keyPressed(int key){
  if (key == 'p' || key == 'P’){
    sound.play();
  }
}

//--------------------------------------------
void ofApp::keyReleased(int key){

}

（以下略）
```

- sound.play();
  - sound に読み込んだ（load した）音声ファイルを再生する
  - これを setup() で実行するとパソコンによってはうまく再生されないことがある

ビルドと実行


ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) テスト(S) 分析(N) ツール(T) 拡張機能(X) ウィンドウ(W) ヘルプ(H)
Debug
Win32
ローカル Windows デバッガー
自動
Live Share
ソリューション エクスプローラー
ソリューション エクスプローラー の検索 (Ctrl+;)
ソリューション 'myBraveSketch' (2/2 プロジェクト)
myBraveSketch
参照
外部依存関係
src
main.cpp
ofApp.cpp
ofApp.h
icon.rc
openframeworksLib
ofApp.cpp
myBraveSketch
ofApp
keyPressed(int key)
91 }
92 }
93 }
94
95 //--
96 void ofApp::draw(){
97 for (auto& circle : circles){
98 ofDrawCircle(circle.position, circle.radius);
99 }
100 }
101
102 //--
103 void ofApp::keyPressed(int key){
104 if (key == 'p' || key == 'P'){
105 sound.play();
106 }
107 }
108
109 //--
110 void ofApp::keyReleased(int key){
111
112 }
113
114 //--
115 void ofApp::mouseMoved(int x, int y){
116
100 %
問題は見つかりませんでした
行: 105 文字: 18 SPC CRLF
ソリューション エ...
クラス ビュー
プロパティ マネ...
チーム エクスプ...
サーバー エクスプローラー
ツールボックス
プロパティ
診断ツール
準備完了
2
0
myBraveSketch
master

‘p’ か ‘P’ をタイプしてサウンドを再生してみる



課題６－１

跳ね返るときに音を出す

円が壁で跳ね返るときに効果音を再生しなさい

- bin > data に置いた 0.wav ～ 6.wav は 1 秒未満の短い音声ファイルである
- 円が壁で跳ね返るときにこれらを再生するようにしなさい
- 音声ファイルは自分で用意しても構わない
  - Windows の「ボイスレコーダー」アプリや macOS / iOS / iPadOS の「ボイスメモ」アプリで作成した AAC ファイル（拡張子 .m4a）は Windows の ofSoundPlayer クラスでは多分再生できない
  - mp3 か wav に変換する
- 同じ音を同時に鳴らすには load() した後に setMultiPlay(true)

サウンドファイルをループ再生する

```
（以上略）

//--------------------------------------------
void ofApp::setup(){
  sound.load("sound.mp3");
  sound.setLoop(true);
}

（途中略）

//--------------------------------------------
void ofApp::keyPressed(int key){
  if (key == 'p' || key == 'P'){
    sound.play();
  }
  else if (key == 's' || key == 'S'){
    sound.stop();
  }
}

（以下略）
```

- sound.setLoop(true);
  - 音声ファイルをループ再生（エンドレス再生）するようにする
  - 引数の true が false の場合はループ再生しない
- sound.stop();
  - 音声ファイルの再生を停止する

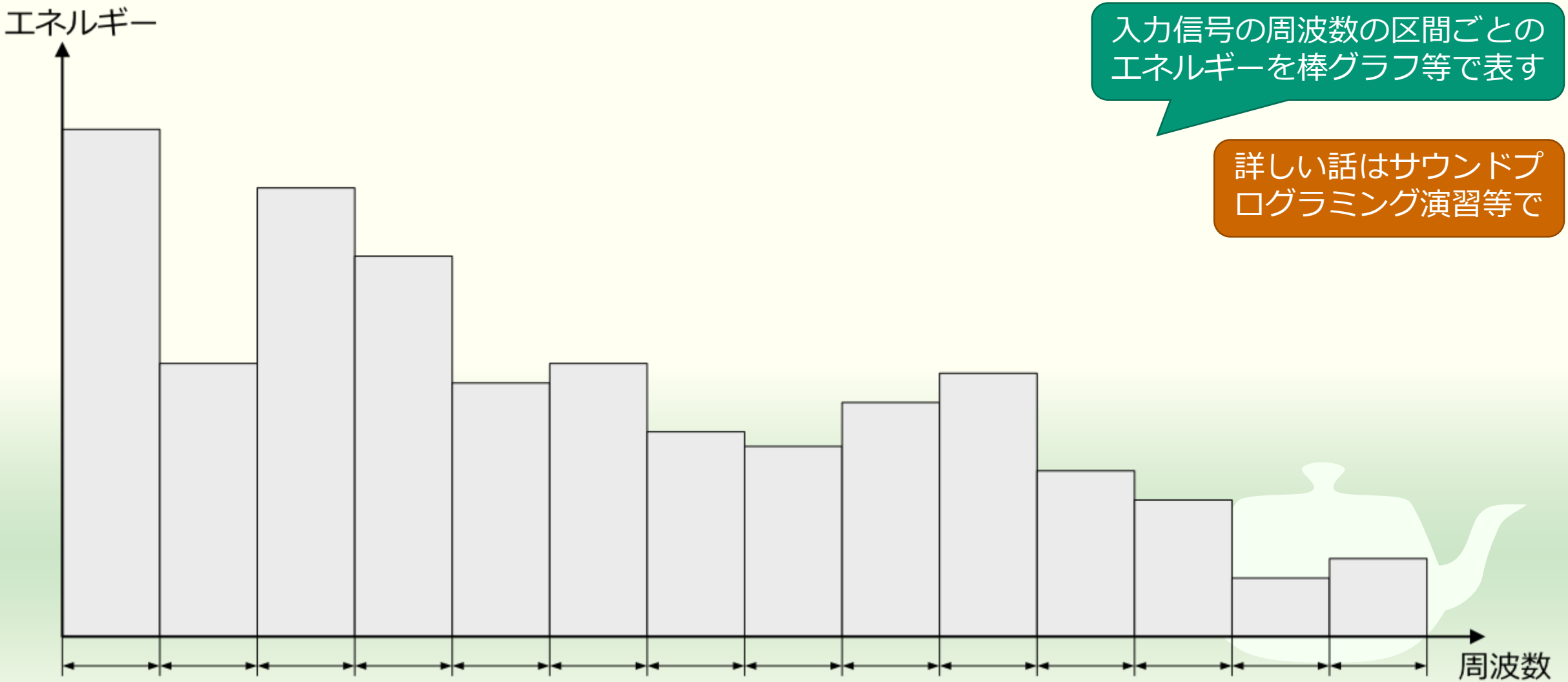
課題のアップロード

- 作成したプログラムの実行中のウィンドウを**５秒以内**で動画キャプチャして、**6-1.mp4** というファイル名で Moodle の第６回課題にアップロードしてください
  - 動画のキャプチャができないときはスクリーンショットを撮って 6-1.png というファイル名でアップロードしてください

スペクトル表示

音の周波数分布

スペクトル分布



入力信号の周波数の区間ごとのエネルギーを棒グラフ等で表す

詳しい話はサウンドプログラミング演習等で

スペクトル分布を格納するメンバ変数を追加

```
（以上略）

class ofApp : public ofBaseApp{
  vector<Circle> circles;
  vec2 startPosition;
  float startTime;
  ofSoundPlayer sound;
  array<float, 64> spectrum{};

public:
  void setup();
  void update();
  void draw();

（以下略）
```

- array<float, 64> spectrum{};
  - array は**固定長**配列
    - サイズ（要素の数）を指定して宣言することによりメモリを確保する
    - vector のように後からデータを追加したり削除したりすることはできない
    - この場合のサイズは 64、spectrum[0] ～spectrum[63] の要素を持つ
- array の初期化
  - 例）array<int, **5**> x{ 3, 1, 2 };
    - x[0] は 3、x[1] は 1、x[2] は 2 に初期化
    - 初期値が指定されていない x[3], x[4] は 0 で初期化される
    - {} だと全部 0 で初期化される

スペクトラム分布の抽出

```
（以上略）

//--------------------------------------------
void ofApp::update(){

  ofSoundUpdate();

  const size_t nBands = spectrum.size();
  const float *val = ofSoundGetSpectrum(nBands);

   （次ページに続く）
```

- spectrum.size()
  - spectrum の要素数を返す
  - これを周波数の区間 nBands に使う
- float *ofSoundGetSpectrum(int nBands)
  - 再生中の音声から高速フーリエ変換 (Fast Fourier Transform, FFT) を用いてスペクトル分布を求める
  - 結果が格納されたメモリへのポインタを返す
  - nBands は周波数の区間の数

スペクトラム分布のグラフデータの作成

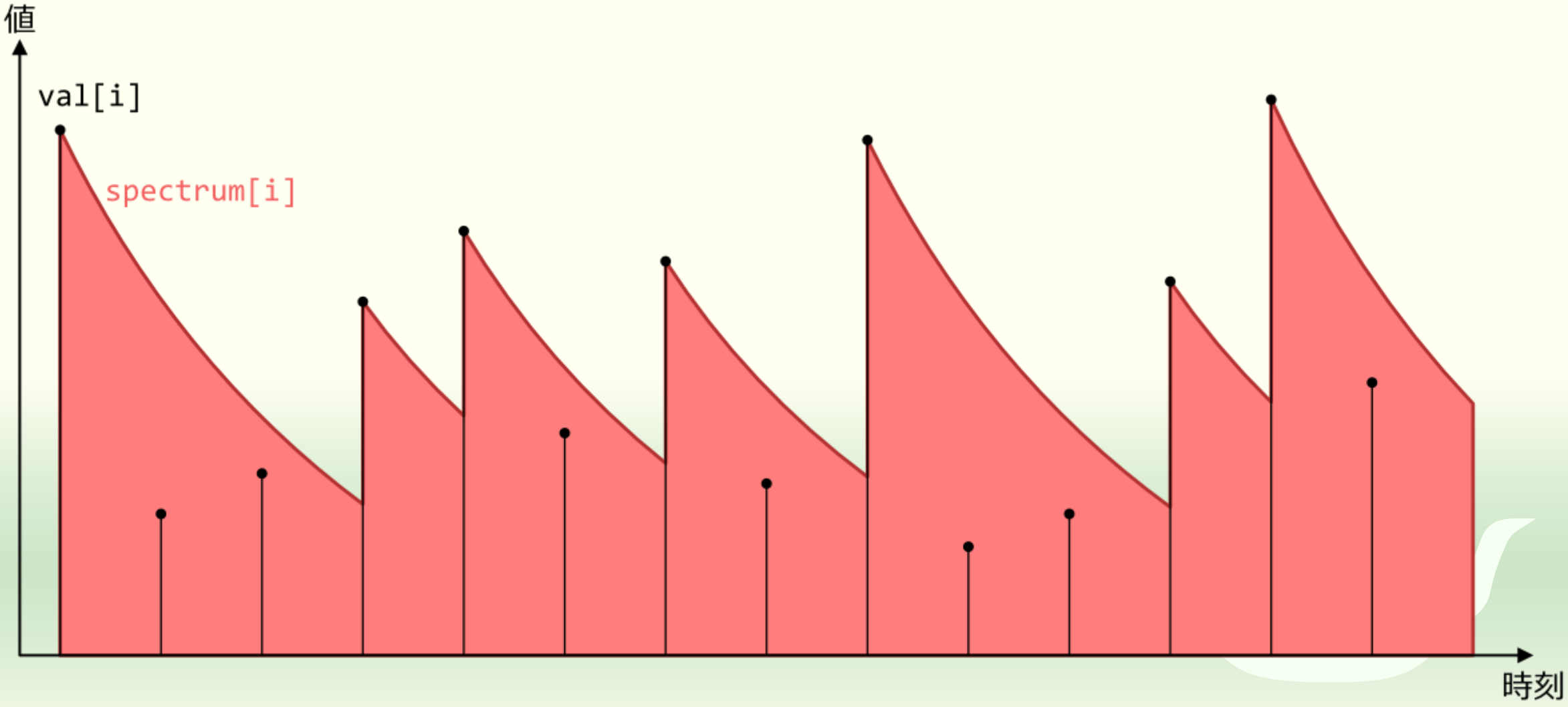
（全ページからの続き）

```
  for (size_t i = 0; i < nBands; ++i) {
    spectrum[i] *= 0.96f;
    if (spectrum[i] < val[i]){
      spectrum[i] = val[i];
    }
  }
```

（以下略）

- for (size_t i = 0; i < nBands; ++i) {
  - i を 0 から nBands - 1 まで変化させながら {} 内を繰り返す
  - spectrum[i] *= 0.96f;
    - 以前の値を 0.96 倍することで時間の経過に伴い値が指数関数的に減少する
  - if (spectrum[i] < val[i]){
  - もし入力信号の値 val[i] が現在の値 spectrum[i] を超えていたら
    - spectrum[i] = val[i];
    - spectrum[i] を入力信号の値 val[i] に更新する

スペクトル分布のグラフの変化



課題６－２

スペクトル分布のグラフを描く

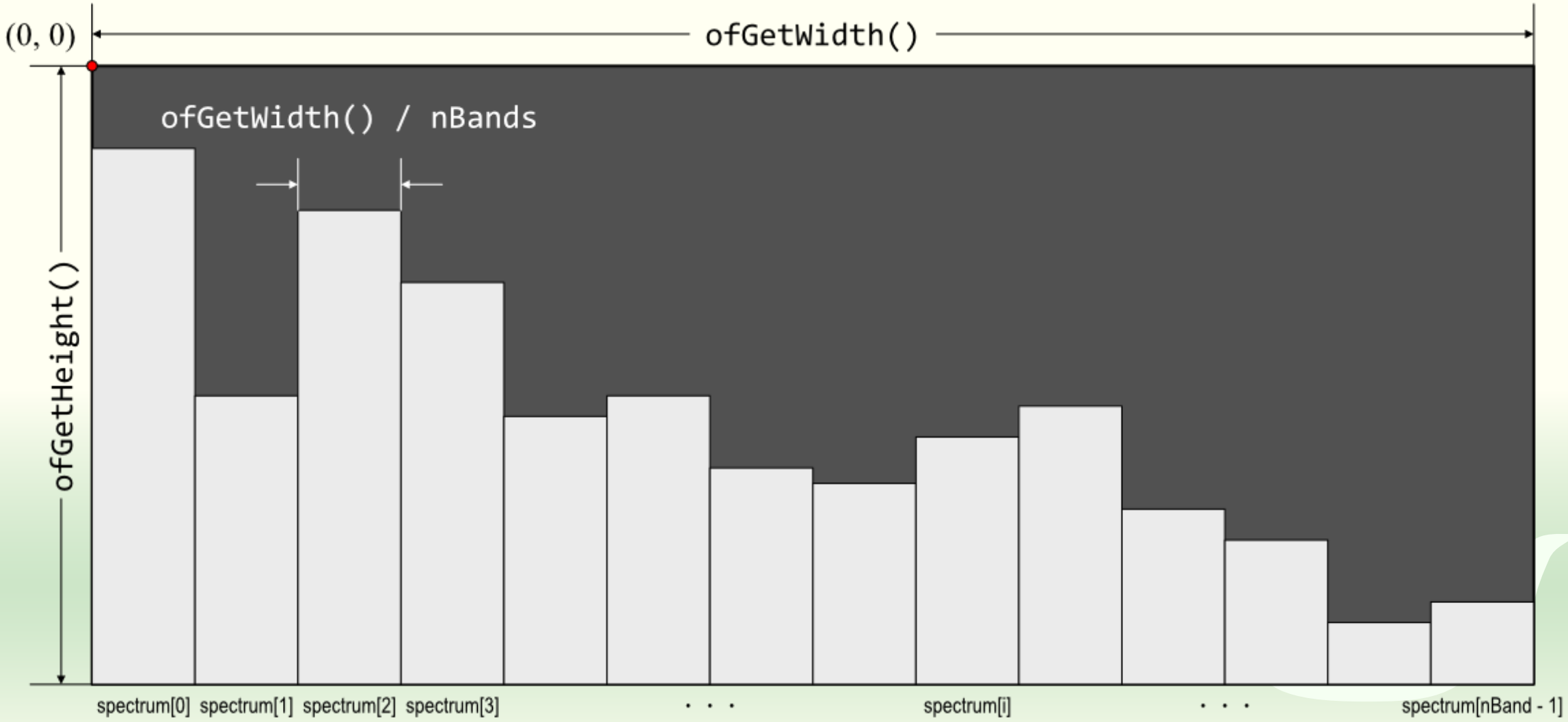
スペクトル分布の棒グラフを描きなさい

- sectrum の値を使ってウィンドウ上にスペクトル分布の棒グラフを円の下に描きなさい
- spectrum[i] には 0～1 の値が入っている
  - したがって棒グラフの高さを spectrum[i] * ofGetHeight() にすれば最大値がウィンドウの高さのグラフになる
  - ただし原点がウィンドウの上端にあるのでそのままではグラフの上下が反転してしまう
- 棒グラフの棒の数は nBands である
  - したがって 1 本の棒グラフの幅は ofGetWidth() / nBands になる

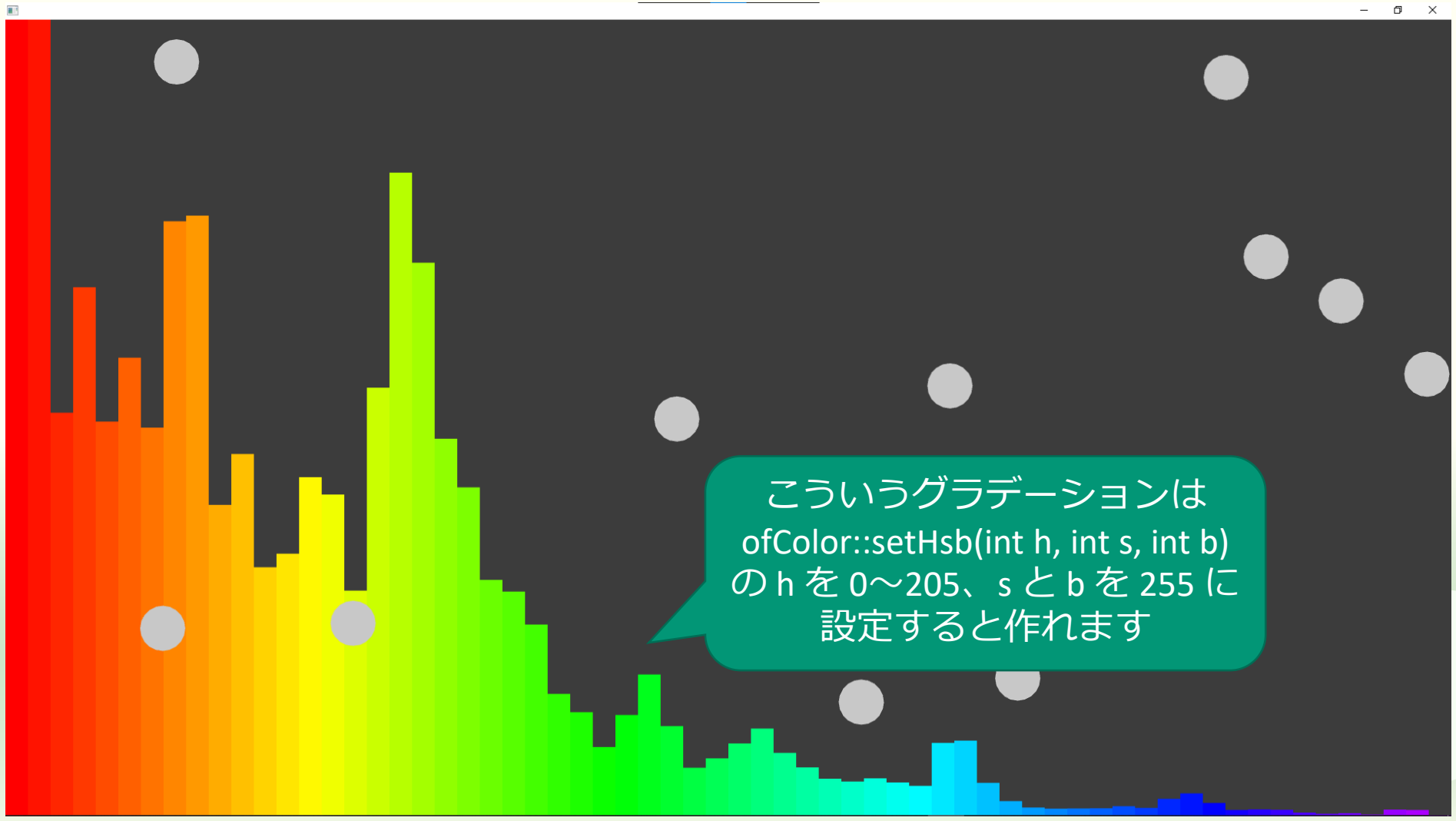
矩形の描画

- void ofDrawRectangle(const glm::vec2 &p, float w, float h)
  - p を左上に幅 w 高さ h の矩形を描く
- void ofDrawRectangle(float x1, float y1, float w, float h)
  - (x1, y1) を左上に幅 w 高さ h の矩形を描く

スペクトル分布の棒グラフのレイアウト



結果の例


こういうグラデーションは
ofColor::setHsb(int h, int s, int b)
の h を 0～205、s と b を 255 に
設定すると作れます

課題のアップロード

- 作成したプログラムの実行中のウィンドウを**５秒以内**で動画キャプチャして、**6-2.mp4** というファイル名で Moodle の第６回課題にアップロードしてください
  - 動画のキャプチャができないときはスクリーンショットを撮って 6-2.png というファイル名でアップロードしてください

音声の入力

サウンドデータの取り扱い

「設定」でマイクの設定を確認する

「システム」→「サウンド」

「プライバシー」→「マイク」

ofApp クラスに音声入力のメンバ変数を追加する

```
（以上略）

class ofApp : public ofBaseApp{
  vector<Circle> circles;
  vec2 startPosition;
  float startTime;
  ofSoundPlayer sound;
  vector<ofSoundPlayer> effect;
  array<float, 64> spectrum{};
  ofSoundStream soundStream;
  array<float, 256> buffer{};
  float volume;

public:
  void setup();
  void update();
  void draw();
  void audioIn(ofSoundBuffer &input);

（以下略）
```

- ofSoundStream
  - リアルタイムに音声の入出力を行うためのクラス
- void audioIn(ofSoundBuffer &input);
  - input に音声データが、CD 品質なら 44,100Hz で取得される
  - update() や draw() は画面表示のタイミングで実行される（60Hz のディスプレイなら 60秒間に1回）
  - タイミングが合わないので音声は画面表示と並行して処理する

サウンド入力の設定

```
#include "ofApp.h"

const vec2 gravity = vec2(0.0f, 200.0f);
const int channels = 2;

//--------------------------------------------------------------
void ofApp::setup(){
  sound.load("sound.mp3");
  sound.setLoop(true);
   （途中略）

  ofSoundStreamSettings settings;
  settings.setInListener(this);
  settings.sampleRate = 44100;
  settings.numOutputChannels = 0;
  settings.numInputChannels = channels;
  settings.bufferSize = buffer.size() * channels;
  soundStream.setup(settings);
}

 （以下略）
```

2 で音声が入らないときは
1 で試してください

- settings.setInListener(this);
  - このオブジェクト (ofApp) の audioIn() を使って音声データを受け取る
- settings.sampleRate = 44100;
  - サンプリングレートを 44,100Hz（CD 品質）に設定する
- settings.numOutputChannels = 0;
  - このプログラムでは出力しないので出力チャネル数は 0 にする
- settings.numInputChannels = channels;
  - 入力チャネル数は 2（ステレオ）にする
- settings.bufferSize = buffer.size() * channels;
  - 取り出し用のメモリ (buffer) のチャネル数のバッファ（一時メモリ）を確保する

ofApp.cpp に追加するサウンド入力関数

```
//--------------------------------------------------------------
void ofApp::audioIn(ofSoundBuffer &input){

  // 二乗和
  float square = 0.0f;

  for (size_t i = 0; i < input.getNumFrames(); i += channels){

    // 左チャネルだけを保存する
    buffer[i / channels] = input[i];

    // 二乗和を求める
    square += input[i] * input[i];
  }

  // RMS (root mean square, 二乗平均平方根)
  volume = sqrt(square * channels / input.getNumFrames());
}
```

- void ofApp::audioIn(ofSoundBuffer &input){
  - バッファに入力音声データが満たされたら実行される
  - 入力音声データは input に格納されている
- input.getNumFrames()
  - input に格納されている入力音声データの数
    - 音声データはチャネルごとに順番に入っている
    - チャネル数が 2 なら、input[0] は左、input[1] は右、 input[2] は左、input[3] は右、…
  - buffer[i / channels] = input[i];
    - 左チャネルだけ使うので偶数番号のデータだけを buffer にコピーする
  - square += input[i] * input[i];
    - 入力データの二乗和を求めておく
- volume = sqrt(square * channels / <省略>);
  - volume にはバッファの中に格納されている音声データの**音量** (0～1) が入る

音量で円の大きさを制御する

```
#include "ofApp.h”

 （途中略）

//--------------------------------------------
void ofApp::draw(){
  const float cx = ofGetWidth() * 0.5f;
  const float cy = ofGetHeight() * 0.5f;
  const float cr = (cx < cy ? cx : cy) * volume;
  ofSetColor(150, 150, 150);
  ofDrawCircle(cx, cy, cr);

   （途中略）
}

 （以下略）
```

- const float cx = ofGetWidth() * 0.5f;
  - cx はウィンドウの横方向の中心
- const float cy = ofGetHeight() * 0.5f;
  - cy はウィンドウの縦方向の中心
- cx < cy ? cx : cy
  - cx < cy なら cx、でなければ cy
    - cx と cy の小さい方に volume を掛ける
    - 条件 ? 式１ : 式２
      - 条件が true なら式１の値を求め、そうでなければ式２の値を求める
      - **３項演算子**

課題６－３

声で円を追加する

一定以上の音量で円を追加するようにしなさい

- マイクに向かって一定以上の声で叫ぶと円が追加されるようにしてください
- ヒント：update() で volume と適当な閾値を比較して volume が閾値を超えたら円を生成します
  - 初期位置はウィンドウの中心にするといいでしょう
  - 初速度の大きさを声の大きさで決定するのも面白いと思います
  - 初速度の方向を疑似乱数で決定するといろんな方向に移動します
    - float ofRandom(float max), float ofRandom(float v0, float v1)
      - それぞれ 0～max、v0～v1 の疑似乱数を返す

課題のアップロード

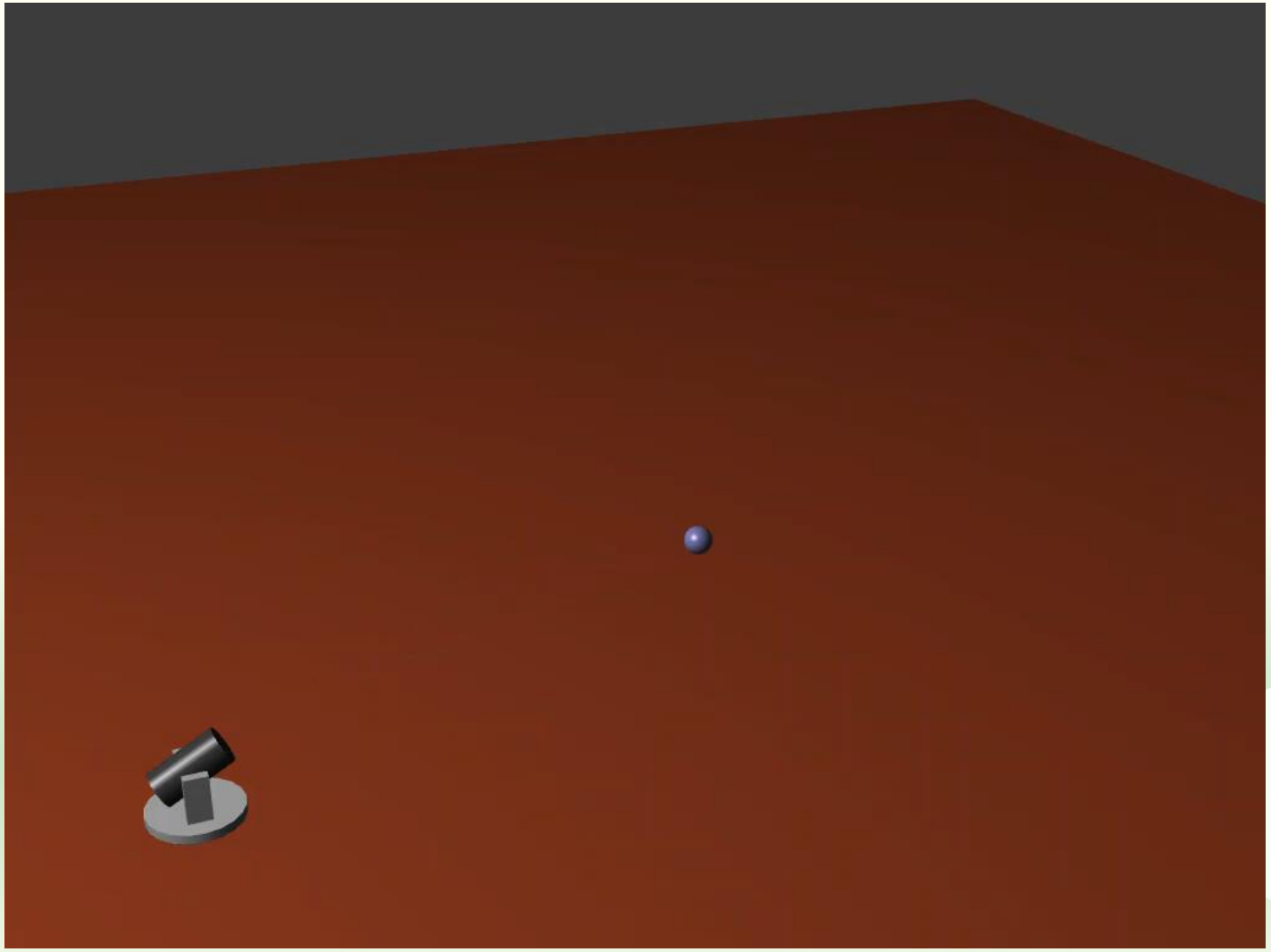
- 作成したプログラムの実行中のウィンドウを**５秒以内**で動画キャプチャして、**6-3.mp4** というファイル名で Moodle の第６回課題にアップロードしてください
  - 動画のキャプチャができないときはスクリーンショットを撮って 6-3.png というファイル名でアップロードしてください
- ソースプログラム **ofApp.h** と **ofApp.cpp** を Moodle の第６回課題にアップロードしてください

メディアプログラミング演習

第7回

今回は最終日なので９０分２コマ（１６時２０分終了）です

本日は簡単な的当てゲームの作成

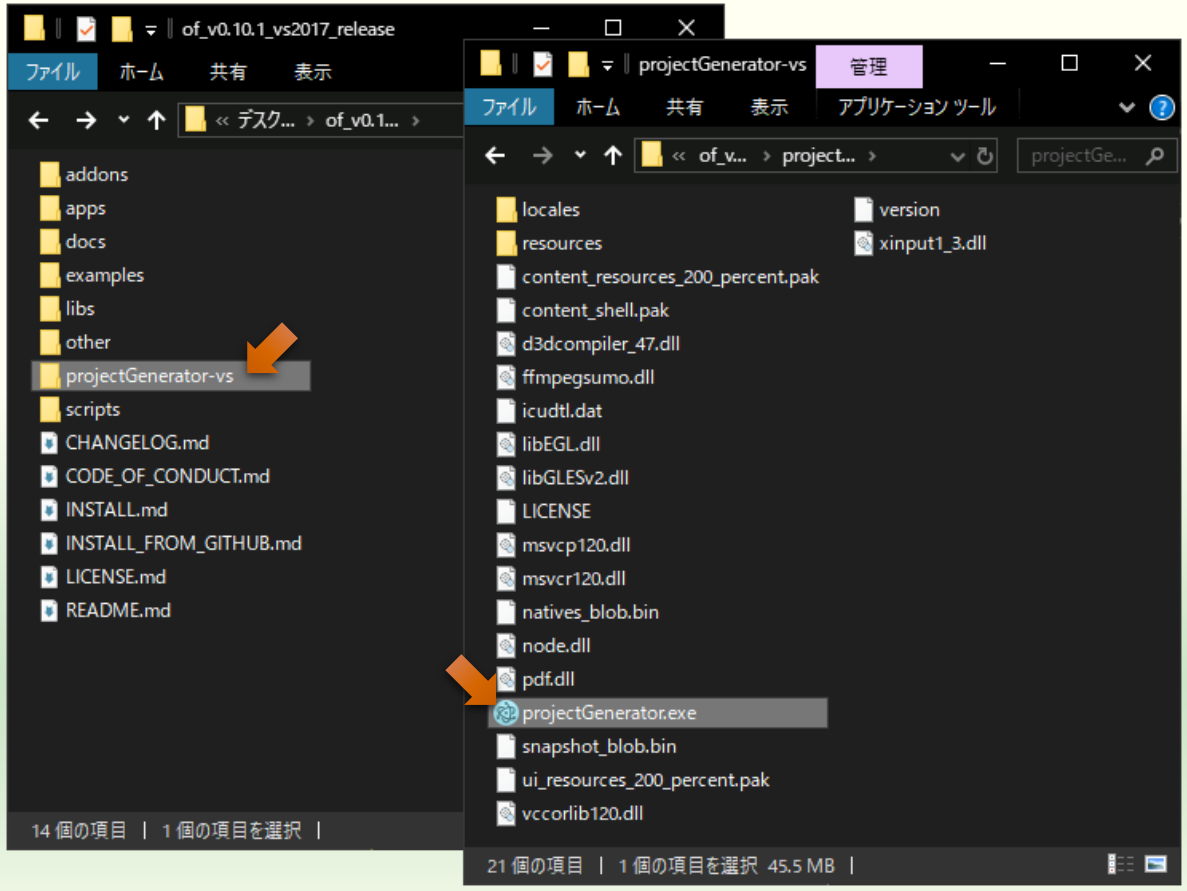


準備

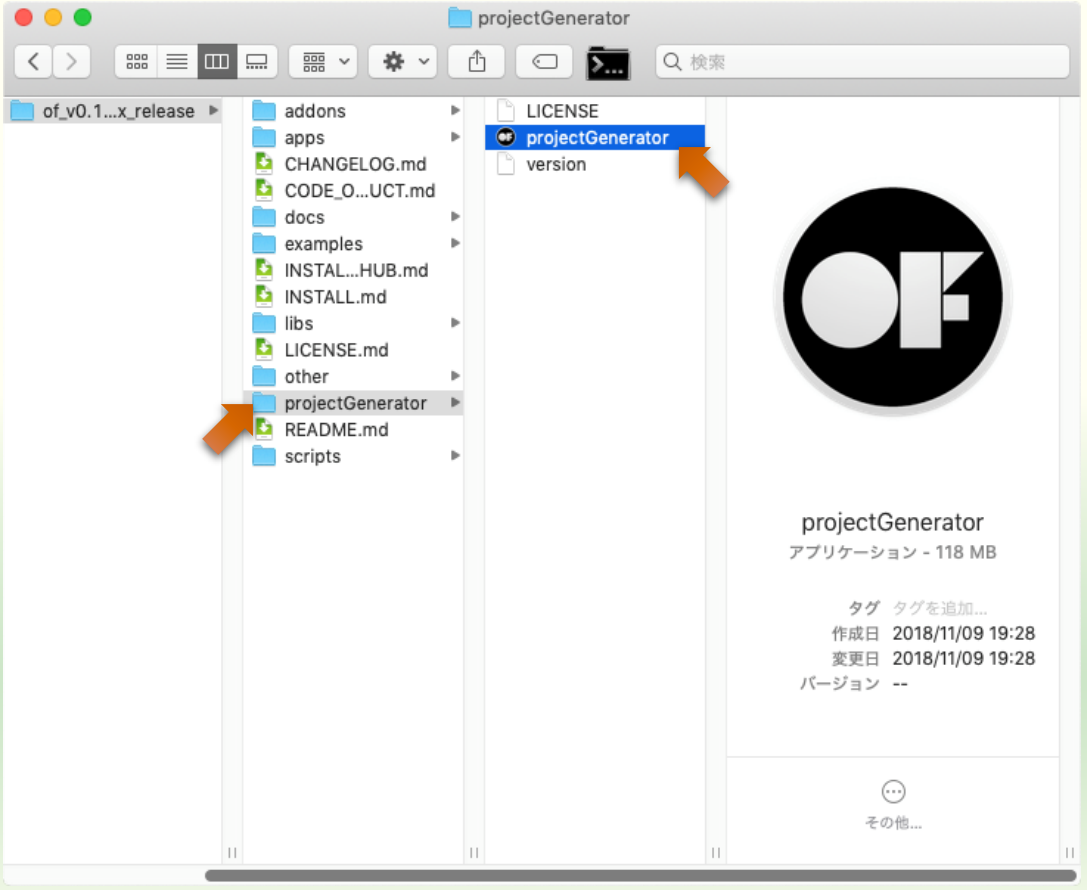
プロジェクトの作成

projectGenerator を起動する

windows 版のパッケージ



macOS 版のパッケージ

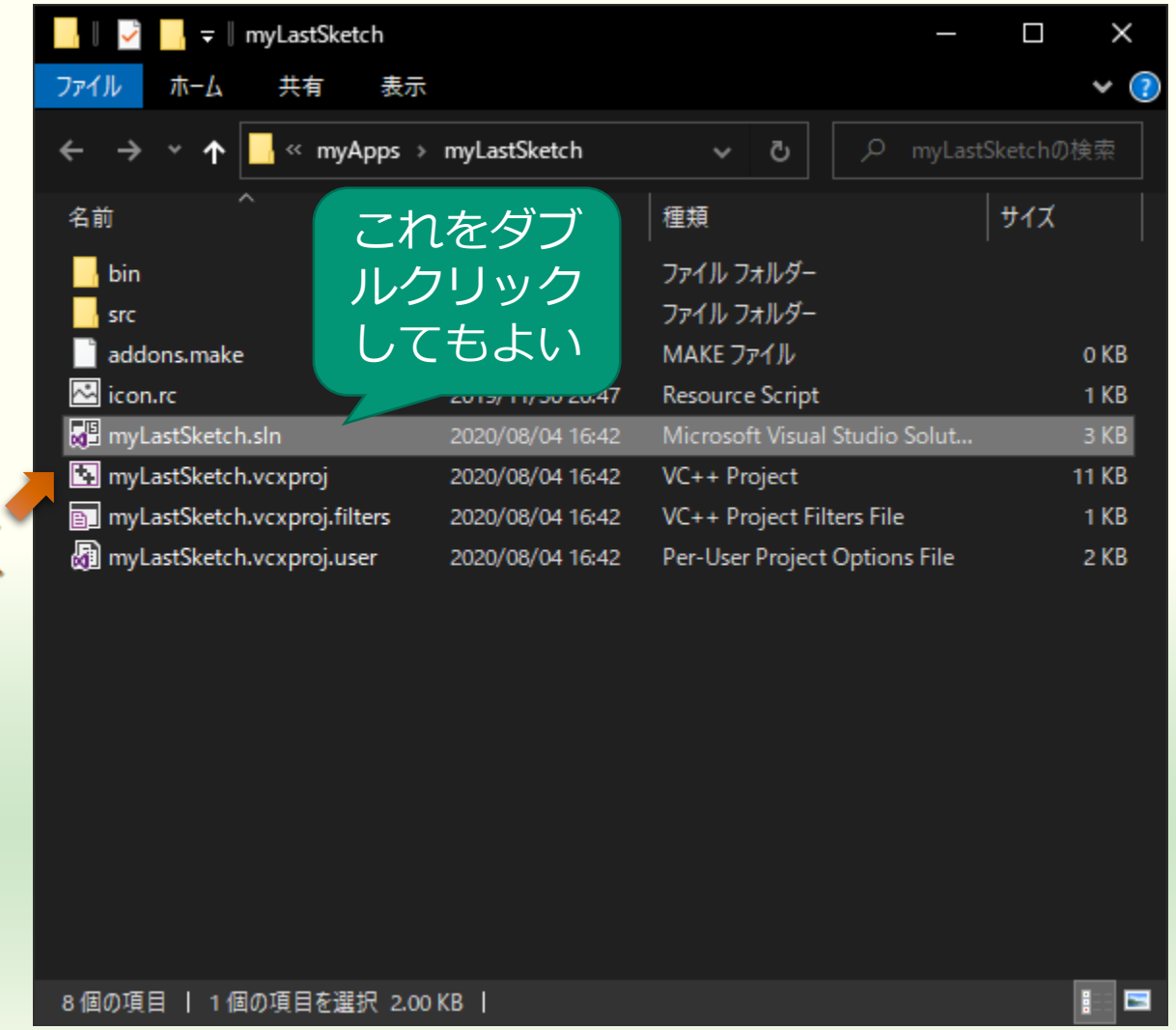


空のプロジェクトの作成

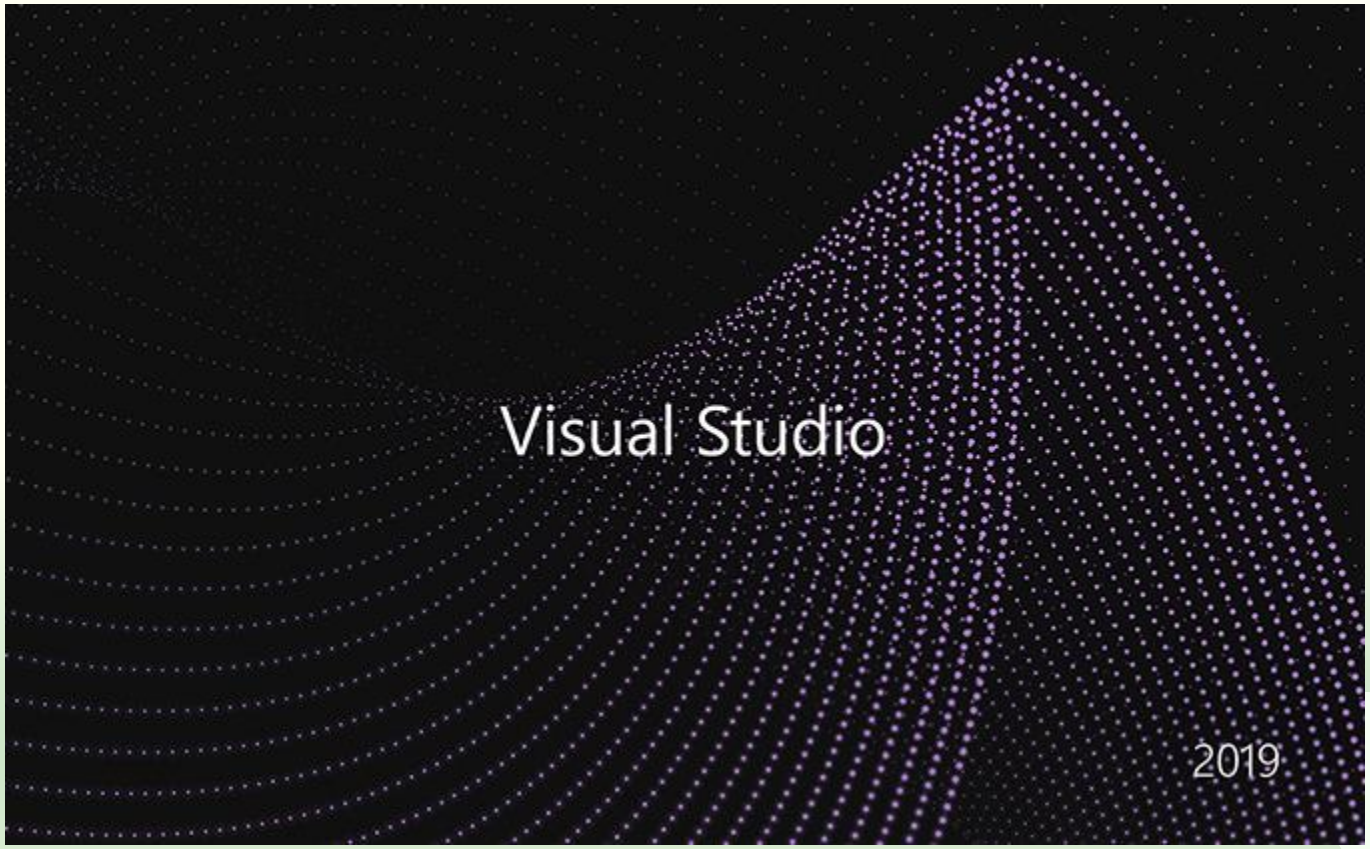
- Project name:
  - 作成するプロジェクト（プログラム）の名前
- Project path:
  - 作成するプロジェクトのファイルを置く場所
  - openFrameworks のパッケージを展開した場所の中の apps¥myApps

プロジェクトの作成成功

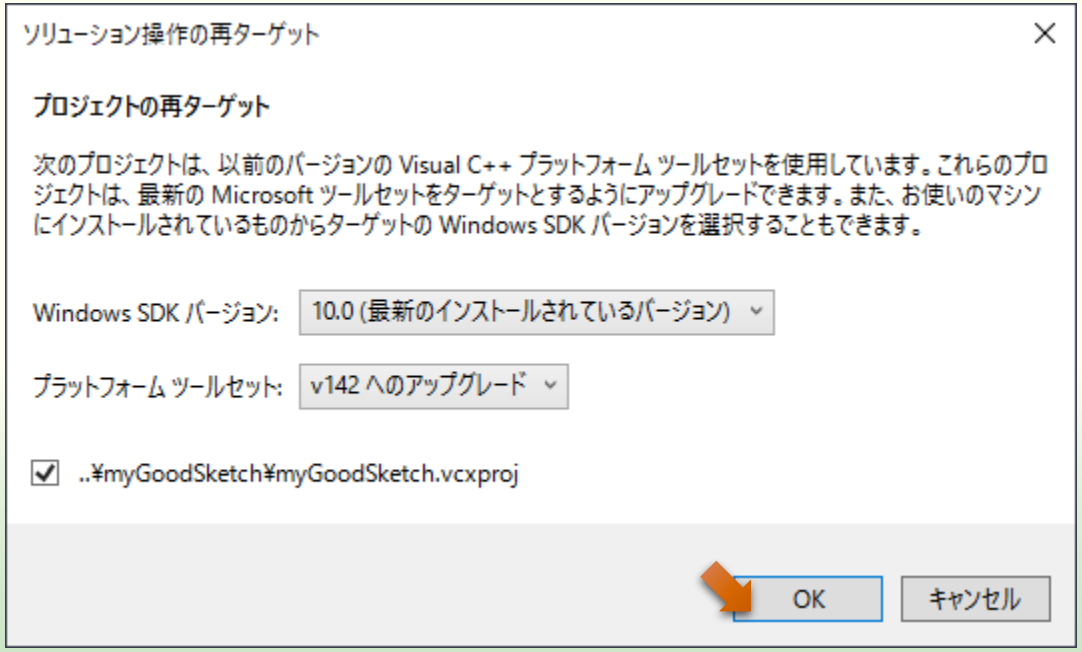
クリックすると開く



Visual Studio 2019 が起動する

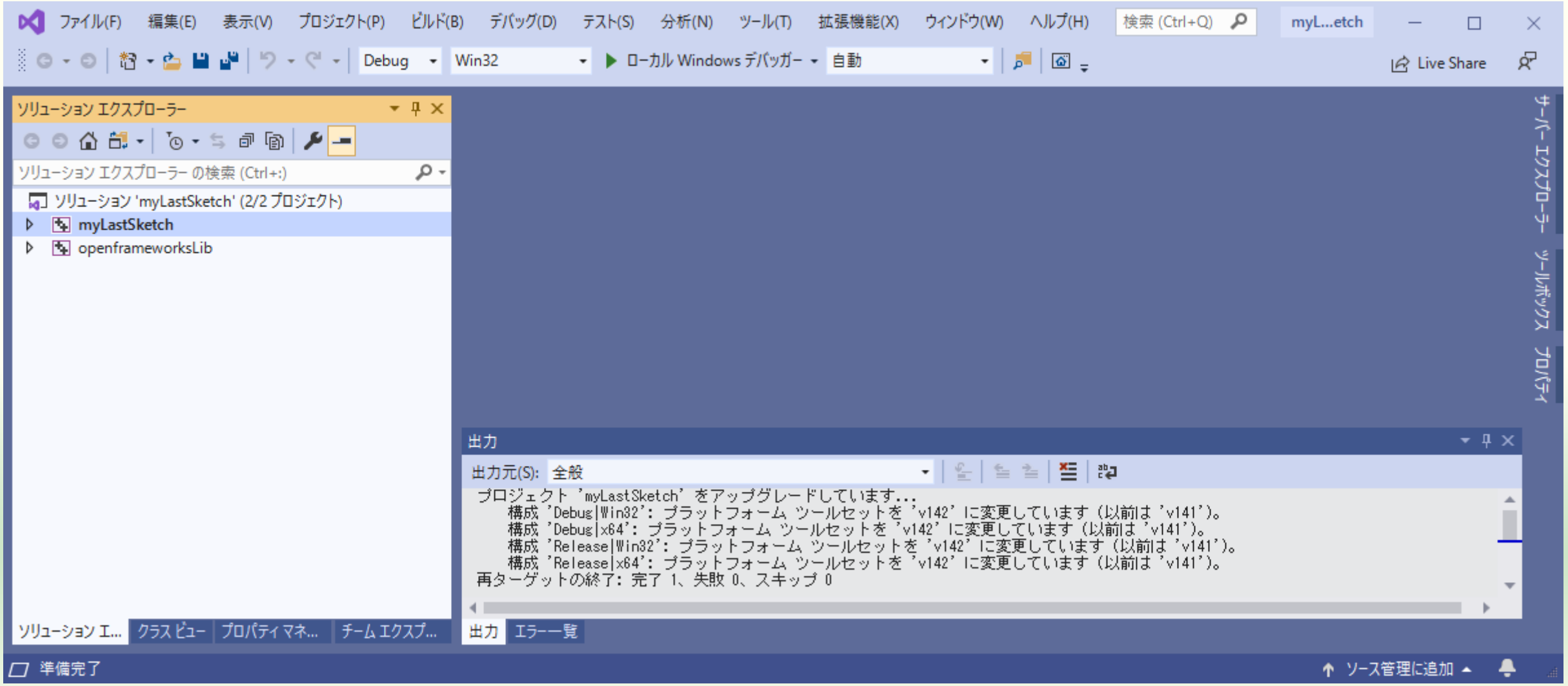


ソリューションの再ターゲット



Visual Studio は頻繁に更新しているので皆さんがお使いの Visual Studio SDK のバージョンと合わない場合がある

Visual Studio 起動



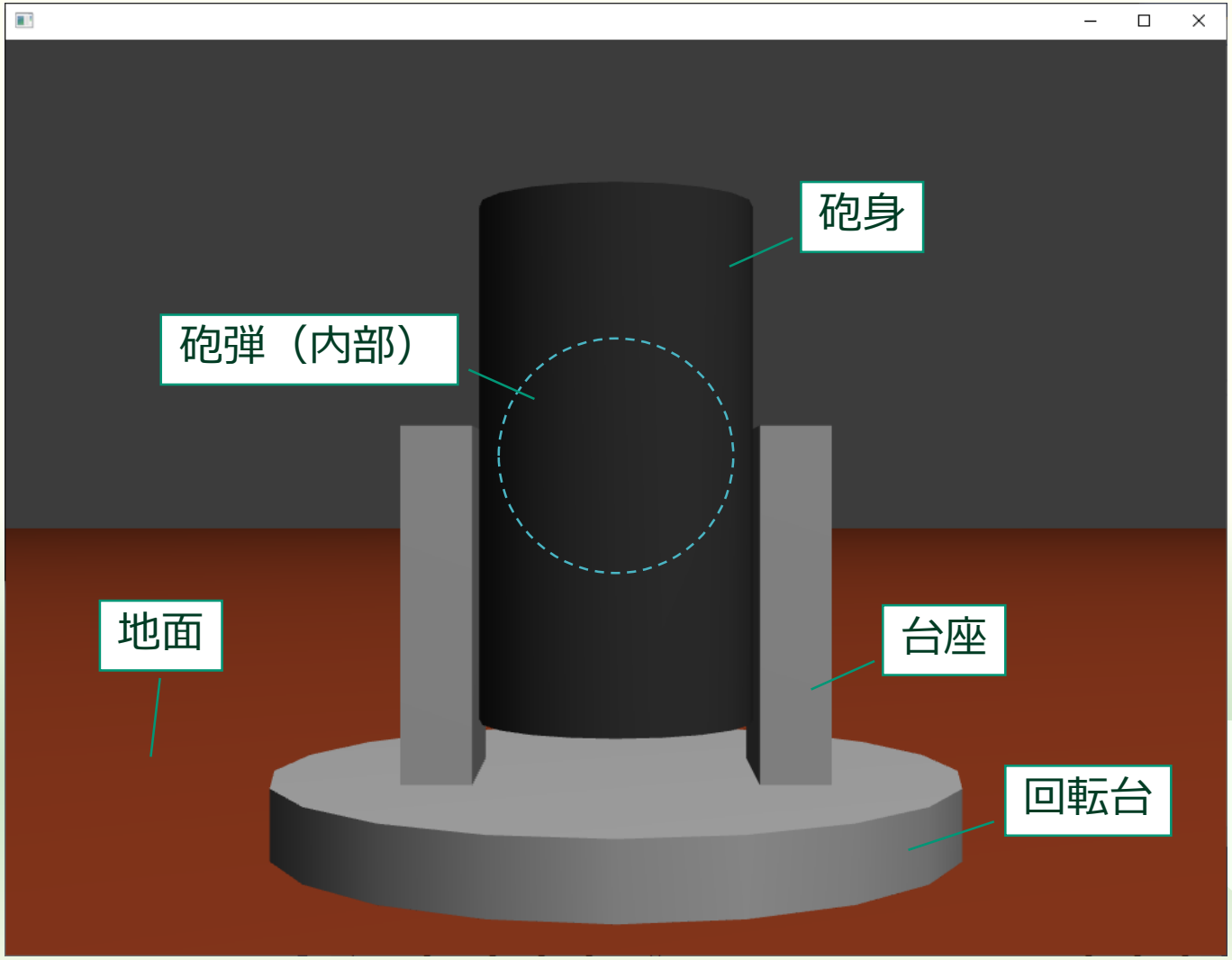
課題 7

矢印キーで砲台の方位角や砲身の仰角を設定しスペースキーでその方向に砲弾を射出する

ofApp クラスに以下のメンバを追加しなさい

- カメラのオブジェクト
  - ofCamera もしくは ofEasyCam クラスのオブジェクトを作成する
    - ofEasyCam クラスはマウスでカメラを操作するクラス
    - ofEasyCam を使うときはキーボード等マウス以外でシーンを操作する
- ライトのオブジェクト
  - ofLight クラスのオブジェクトを作成する
- 砲弾の速度と加速度
  - それぞれ glm::vec3 クラスのオブジェクトを作成する
- 図形のオブジェクト（回転台、台座、砲身、砲弾、地面）

図形のオブジェクト



ofApp::setup() に以下の処理を追加しなさい

- カメラの位置、方向、画角の設定
- ライトの位置の設定と有効化
- 隠面消去処理の有効化
- 砲弾の速度と加速度の初期値の設定
- 図形のオブジェクトの設定
  - 回転台、台座、砲身、砲弾、地面の大きさと位置
  - 砲弾の初速度、加速度は 0 にする

ofApp::update() に以下の処理を追加しなさい

- 左右の矢印キーで砲台の方位角 (heading / pan) の変更
- 上下の矢印キーで砲身の仰角 (pitch / tilt) の変更
- 砲弾の位置と速度の更新
- 砲弾の着弾判定
  - 着弾は目標との衝突か地面への着地で判定する
  - 着弾後の処理は任意
    - 音を出す、視覚効果を表示する、ほか

ofApp::draw() に以下の処理を追加しなさい

- カメラの使用開始
- 図形のオブジェクトの描画
  - 回転台、台座、砲身、砲弾、地面
- カメラの使用終了

ofApp::keyPressed() に以下の処理を追加しなさい

- スペースキーで砲弾の発射速度の設定
  - 砲台の方位角と砲身の仰角から射出方向の速度ベクトルを決定
  - 速度の設定
  - 加速度（重力加速度）の設定

課題のアップロード

- 作成したプログラムの実行中のウィンドウを **5 秒以内**で動画キャプチャして、**7.mp4** というファイル名で Moodle の第 7 回課題にアップロードしてください
  - 動画のキャプチャができないときはスクリーンショットを撮って 7.png というファイル名でアップロードしてください
- ソースプログラム **ofApp.h** と **ofApp.cpp** を Moodle の第 7 回課題にアップロードしてください