

# クラスメンバ

ゲームプログラミングA #03

向井 智彦

# 今週の内容

- クラスメンバ
- カプセル化
  - アクセス指定子
  - constメンバ関数
- 続・メンバ関数
  - 引数付きコンストラクタ
  - 引数が異なるメンバ関数
  - 演算子オーバーライド

# クラス：複数の変数と関数の複合体

1. int a;
2. float b;
3. double c;
4. char d;

1. class Data
2. {
3. int a;
4. float b;
5. double c;
6. double Func();
7. };

# 復習に用いるクラス

```
class Vector2 // 2次元ベクトル
{
public:
    double x;
    double y;
};
```

# メンバ変数へのアクセス(復習)

1. `Vector2 v;` // Vector2変数vの宣言

2. `v.x = 10;`

// v + ピリオド + 要素名 x

3. `v.y = 20;`

// v + ピリオド + 要素名 y

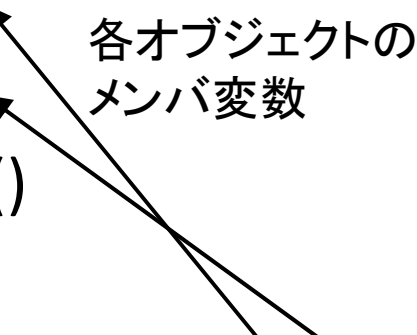
4. `printf(“%f, %f¥n”, v.x, v.y);`

```
class Vector2
{
public:
    double x;
    double y;
};
```

# メンバ関数 print の追加


```
class Vector2
{
public:
    double x;
    double y;
    void Print()
    {
        printf("%f, %f\n", x, y);
    }
};
```

各オブジェクトのメンバ変数




```
int main()
{
    Vector2 a, b;
    a.x = 10;  a.y = 5;
    b.x = 0;   b.y = 20;
    a.Print(); // 10, 5
    b.Print(); // 0, 20
    return 0;
}
```

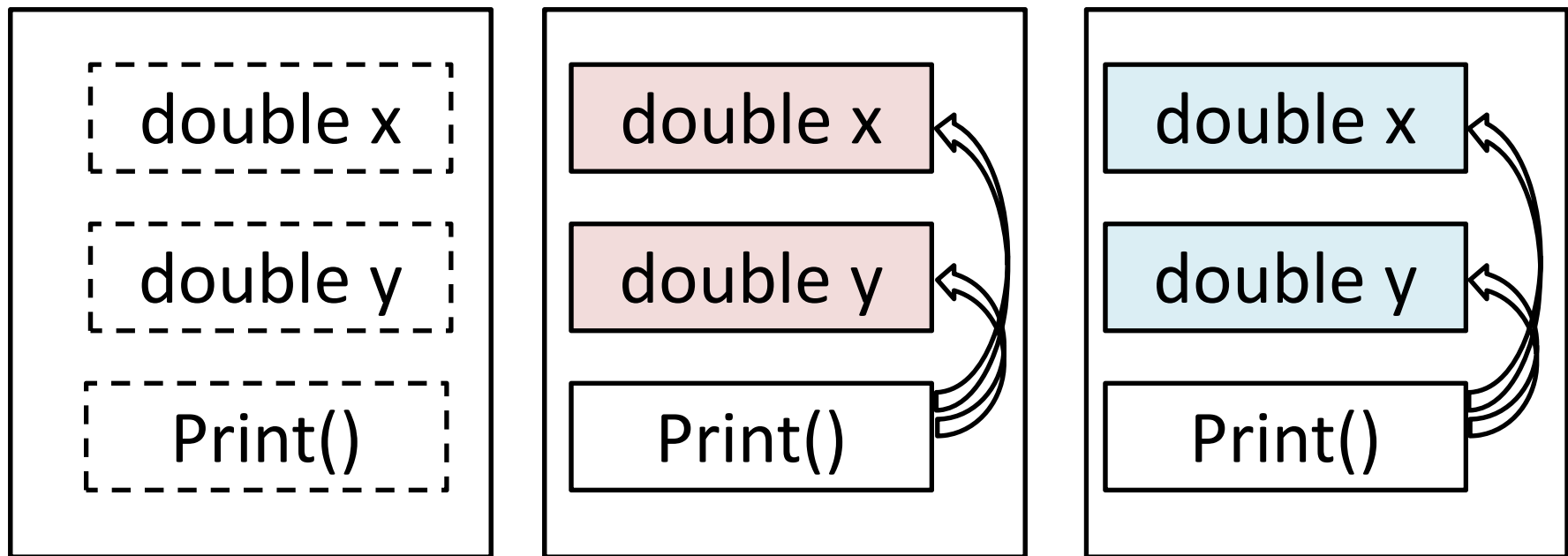
ベクトルaの成分x, yをprint



ベクトルbの成分x, yをprint



# クラス - オブジェクト - メンバ



class Vector2

クラスはメンバの数や  
種類を定義

Vector2 **a**;

クラス変数＝オブジェクト・インスタンスを宣言すると  
インスタンスごとにメンバ変数が用意される  
メンバ関数は、各インスタンスがもつメンバ変数に  
アクセスできる

Vector2 **b**;

# vector3class.cpp

```
class Vector3
```

```
{
```

```
public:
```

```
    double x, y, z;
```


```
    double Length() {
```

```
        return sqrt(x * x + y * y + z * z);
```

```
    }
```

```
};
```

(いずれ作成される)各クラスインスタンス、  
つまり各ベクトルのメンバ変数にアクセスして、  
それぞれのベクトルの長さを計算





# Quick Exercise

## ベクトルの足し算

```
class Vector3
{
public:
    double x, y, z;
    void Add(Vector3 c)
    {
        // 自身の x,y,z に
        // 引数 c の x,y,z を
        // それぞれ加算
    }
};
```

```
int main()
{
    Vector3 a, b;
    a.x = 1.0; a.y = 1.0; a.z = 1.0;
    b.x = 2.0; b.y = 1.0; b.z = 2.0;

    a.Add(b);
    // ベクトル a は (3.0, 3.0, 3.0) に
}
```

アクセス指定、constメンバ関数

# カプセル化

# クラス設計の基本的指針

- メンバー変数は全て隠す = privateにする
  - オブジェクトの内部状態を気にせず扱えるように
  - publicなメンバー関数を通じて読み書き
- インターフェースを介したオブジェクト操作

# パソコンを例に挙げると...

- メンバ変数：半導体集積回路（見たくない）
- インターフェース：キーボード、マウス、モニタ

```
class PC {
```

```
public:
```

```
    void TypeKey(); void ClickMouse();
```

```
private:
```

```
    CPU c; Memory mem; SSD s; GPU g;
```

```
}
```

各パーツのclassにもprivateなメンバーと  
publicなメンバーがあって、さらにその...

# アクセス指定子

見せたい  
メンバは  
publicに

隠したい  
メンバは  
privateに

この部分も  
publicに

```
class PC {  
    public:  
        void TypeKey();  
        void ClickMouse();  
    private:  
        CPU c;  
        Memory mem;  
        SSD s;  
        GPU g;  
    public:  
        ....  
}
```

# カプセル化 (スライド6の改変)

```
class Vector2
```

```
{
```

```
private:
```

```
double x;
```

```
double y;
```

```
public:
```

```
void Print()
```

```
{
```

```
    printf("%f, %f¥n", x, y);
```

```
}
```

```
private:
```

```
void PrivateFunc() { .... }
```

```
};
```

クラス内の他のメンバ関数から呼び出し可能  
一方、クラス外からのアクセスは不可

メンバ関数内では  
メンバ変数への  
アクセス制限無し

```
int main()
```

```
{
```

```
    Vector2 a, b;
```

```
    a.x = 10;  a.y = 5;
```

```
    b.x = 0;   b.y = 20;
```

```
    a.Print(); // OK
```

```
    b.Print(); // OK
```

```
    a.PrivateFunc(); // エラー
```

```
    return 0;
```

```
}
```

全てエラー:  
クラス外から  
privateメンバは  
アクセスできない

# Vector2クラス カプセル化演習

1. メンバ変数を全てprivateにする
2. 各成分x, y, zを取得するためのメンバ関数をそれぞれ追加
  - `double GetX()` , `GetY()` , `GetZ()`
3. 成分毎, および三成分を同時に設定するメンバ関数を追加
  - `void SetX(double)` , `SetY(double)` , `SetZ(double)`
  - `void Set(double, double, double)`

# 続・メンバ関数

- 引数付きコンストラクタ
  - インスタンス生成時に初期化内容を指定
- 引数の数や種類が異なる同名のメンバ関数を定義可能
- 演算子オーバーライド
  - 「+」「-」「==」「%」等の演算子をクラス毎に再定義



# 引数付きコンストラクタ

```
class Vector2
{
private:
    double x, y;
public:
    Vector2() {
        x = 0.0; y = 0;
    }
    Vector2(double initX,
            double initY) {
        x = initX; y = initY;
    }
};
```

デフォルト  
コンストラクタ

```
int main()
{
```

変数名だけの時は  
デフォルトコンストラクタ

```
    Vector2 a;
```

```
    Vector2 b(); // エラー
```

```
    Vector2 b(2.0, 3.0);
```

```
    Vector2 c = Vector2(0, 1.0);
```

```
}
```

引数付きの場合は  
数・種類が一致する  
コンストラクタが呼ばれる

# 引数が異なる同名のメンバ関数

```
class Vector2  
{  
    ....  
public:
```

```
void Init() { x = 0; y = 0; }
```

```
void Init(double a)
```

```
{ x = a; y = a; }
```

```
void Init(double a, double b)
```

```
{ x = a; y = b; }
```

```
int main()  
{
```

```
    Vector2 a;
```

```
    a.Init();
```

```
    a.Init(0.0);
```

```
    a.Init(1.0, 0.0);
```

```
}
```

# 演算子オーバーライド (operator)

```
class Vector2
{
    ....
public:
    Vector2 operator-() const {
        return Vector2(-x, -y);
    }
    Vector2 operator-(Vector2 a)
const {
    return Vector2(x - a.x, y - a.y);
}
    Vector2 operator=(Vector2 a) {
        { x = a.x;   y = a.y; }
    }
```

```
int main()
{
    Vector2 a(0.0, 1.0);
    Vector2 b(1.0, 0.0);

    Vector2 c = -a;
    Vector2 d = a - b;
    a = b;
}
```

# const メンバ関数

- メンバ変数の値を変更できない

```
class Vector2
{
private:
    double x, y;
public:
    double GetX() const
    {
        return x;
    }
};
```

「このメンバ関数はオブジェクトの  
内部状態を変更しない」ということを  
明確に指定する

# const 余談

```
const int x = 10; // xは読み取り専用変数  
int y = 0;
```

```
x += 2; // エラー
```

```
y += 2; // OK
```

```
y += x; //OK
```

```
int xarray[x]; // OK
```

```
int yarray[y]; // エラー
```

# 本日の演習課題

- Vector3クラスをカプセル化
- Vector3クラスにメンバー関数を追加
  - ベクトル長を求めるLength()
  - ベクトルをゼロクリアするZeroClear()
  - 代入演算「=」
  - ベクトルの加算「+」、減算「-」
  - ベクトルの拡大縮小「\*」「/」
  - その他、思いつくものを色々