

[prog2] Programming C++ (C6) Exercise Guide (Ex05)

10/19, Thursday 3rd period.

クラスをはじめよう

- ▶ クラスは本質的には構造体と同じようなもの
 - ⇒ 構造体のメンバーには **変数**しか含まなかった
 - ⇒ クラスのメンバーには **関数** も含めることができる

構造体のプログラム例:

Test構造体

メンバー変数: int x1, x2;

```
1  #include <stdio.h>
2
3  double center(int,int);
4
5  typedef struct{
6      int x1;
7      int x2;
8  }Test;
9
10 int main(){
11     Test a,b;
12
13     scanf("%d",&a.x1);
14     scanf("%d",&a.x2);
15
16     printf("a: %f\n",center(a.x1,a.x2));
17     printf("b: %f\n",center(b.x1,b.x2));
18
19     return 0;
20 }
21
22 double center(int x1,int x2){
23     return (x1+x2)*0.5;
24 }
```

クラスをはじめよう

- ▶ 関数をメンバーに取り込んだ「クラス」を用いた簡単なプログラム(構造体バージョンとほぼ同じもの)を書く
- ▶ クラスの場合、「**アクセス指定子**」というものが登場
- ▶ Ex05-1の段階では、とりあえず**public** としておく

ex05-1.cc

```
1  #include <iostream>
2  using namespace std;
3
4  class Test{
5  public:
6  /* [Complete Here!!] */
7  };
8
9  int main(){
10
11  /* [Complete Here!!] */
12
13
14  return 0;
15 }
```

```
1  #include <stdio.h>
2
3  double center(int,int);
4
5  typedef struct{
6      int x1;
7      int x2;
8  }Test;
9
10 int main(){
11     Test a,b;
12
13     scanf("%d",&a.x1);
14     scanf("%d",&a.x2);
15
16     printf("a: %f\n",center(a.x1,a.x2));
17     printf("b: %f\n",center(b.x1,b.x2));
18
19     return 0;
20 }
21
22 double center(int x1,int x2){
23     return (x1+x2)*0.5;
24 }
```

クラスをはじめよう

- ▶ クラス定義の中に、変数 **x1, x2** ,そして関数 **center()** を入れる
- ▶ main関数は、**C++ 様式**で書き直す
- ▶ クラス定義は、構造体定義と同様にデータ構造の「テンプレート」である
- ▶ 構造体の使用と同じように、使いたい場所で**実体(オブジェクト、インスタンス)**を宣言する

ex05-1.cc

```
1  #include <iostream>
2  using namespace std;
3
4  class Test{
5  public:
6  /* [Complete Here!!] */
7  };
8
9  int main(){
10
11  /* [Complete Here!!] */
12
13
14  return 0;
15 }
```

```
1  #include <stdio.h>
2
3  double center(int,int);
4
5  typedef struct{
6  int x1;
7  int x2;
8  }Test;
9
10 int main(){
11     Test a,b;
12
13     scanf("%d",&a.x1);
14     scanf("%d",&a.x2);
15
16     printf("a: %f\n",center(a.x1,a.x2));
17     printf("b: %f\n",center(b.x1,b.x2));
18
19     return 0;
20 }
21
22 double center(int x1,int x2){
23     return (x1+x2)*0.5;
24 }
```

クラスをはじめよう

- ▶ 外部からのアクセス方法は、構造体と同じで **.** (ドット)を使う(関数も)
 - ▶ クラス内の関数は、そのクラスのメンバー変数に直接アクセス可能
 - ▶ Testクラスのメンバーとなっている center() に**引数は不要**
- ⇒ center() の中の処理には、何の修正を加えなくても動作するはず

ex05-1.cc

```
1  #include <iostream>
2  using namespace std;
3
4  class Test{
5  public:
6  /* [Complete Here!!] */
7  };
8
9  int main(){
10
11  /* [Complete Here!!] */
12
13
14  return 0;
15 }
```

```
1  #include <stdio.h>
2
3  double center(int,int);
4
5  typedef struct{
6  int x1;
7  int x2;
8  }Test;
9
10 int main(){
11     Test a,b;
12
13     scanf("%d",&a.x1);
14     scanf("%d",&a.x2);
15
16     printf("a: %f\n",center(a.x1,a.x2));
17     printf("b: %f\n",center(b.x1,b.x2));
18
19     return 0;
20 }
21
22 double center(int x1,int x2){
23     return (x1+x2)*0.5;
24 }
```

クラスをはじめよう：オブジェクト指向

▶ ex05-1.cc（とりあえずクラスを使ってみた）

```
1  #include <iostream>
2  using namespace std;
3
4  class Test{
5  public: //以下のメンバーは、どこからでも参照可能
6      int x1,x2;//メンバー変数 x1, x2
7      double center(){//メンバー関数 center (中点を計算する。x1,x2はTestクラスのメンバー変数)
8          return (x1+x2)*0.5;
9      }
10 };
11
12 int main(){
13     Test a, b; // Testクラスのオブジェクト（実体）を生成
14
15     cin >> a.x1 >> a.x2;// a のメンバー x1,x2 に値を入力
16     cout << "a: " << a.center() << endl;// aに対するメンバー関数center()を呼び出す
17     cout << "b: " << b.center() << endl;// bに対するメンバー関数center()を呼び出す（出力値不定）
18     return 0;
19 }
```

- ▶ クラスは、変数だけでなく関数もメンバーにできる
- ▶ 関数本体をクラス定義の中に入れる
- ▶ メンバー変数と同様に、関数も .（ドット）でアクセス

クラスをはじめよう：オブジェクト指向

オブジェクト指向に沿った書き方へ

- ▶ その前に、アクセス指定子：**private**
- ▶ **public** のときは「どこからでも参照できた」
⇒ **構造体を作るときの意識と同じまま、特別なことは不要**

```
1  #include <iostream>
2  using namespace std;
3
4  class Test{
5  private:
6      int x1,x2;
7      double center(){
8          return (x1+x2)*0.5;
9      }
10 };
11
12 int main(){
13     Test a, b;
14
15     cin >> a.x1 >> a.x2; //a.x1やa.x2はクラスの外からは参照できない
16     cout << "a: " << a.center() << endl; //a.center()はクラスの外からは参照できない
17     cout << "b: " << b.center() << endl; //b.center()はクラスの外からは参照できない
18     return 0;
19 }
```


クラスをはじめよう：オブジェクト指向

アクセス指定子： **private**

▶ **そのクラスのオブジェクトからしか**参照できなくなる

⇒main関数を含む他の場所から **a.x1** のような形でアクセスできない

▶ 前回の **public** をただ、**private** に変えた例 (コンパイル×)



```
1  #include <iostream>
2  using namespace std;
3
4  class Test{
5  private:
6      int x1,x2;
7      double center(){
8          return (x1+x2)*0.5;
9      }
10 };
11
12 int main(){
13     Test a, b;
14
15     cin >> a.x1 >> a.x2; //a.x1やa.x2はクラスの外からは参照できない
16     cout << "a: " << a.center() << endl; //a.center()はクラスの外からは参照できない
17     cout << "b: " << b.center() << endl; //b.center()はクラスの外からは参照できない
18     return 0;
19 }
```


クラスをはじめよう：オブジェクト指向

オブジェクト指向： 基本的な構造

メンバー変数 ⇒ **private**

メンバー関数 ⇒ **public**

```
1  class Test{  
2  private:    //外からアクセスできない  
3      int x1,x2;  
4  public:    //外からアクセスできる  
5      double center(){  
6      /*  
7       center()はTestクラスの内部構造の1つであるので、  
8       Testクラスのメンバー変数 x1,x2 は参照できる  
9       */  
10         return (x1+x2)*0.5;  
11     }  
12 };
```

外の関数からは `a.center()`; のような関数呼び出しは **可能**

`a.x1;` や `a.x2;` といった変数の参照は **不可能**

クラスのメンバー関数 `center()` の中からは **`x1;` `x2;` の形で参照可能**

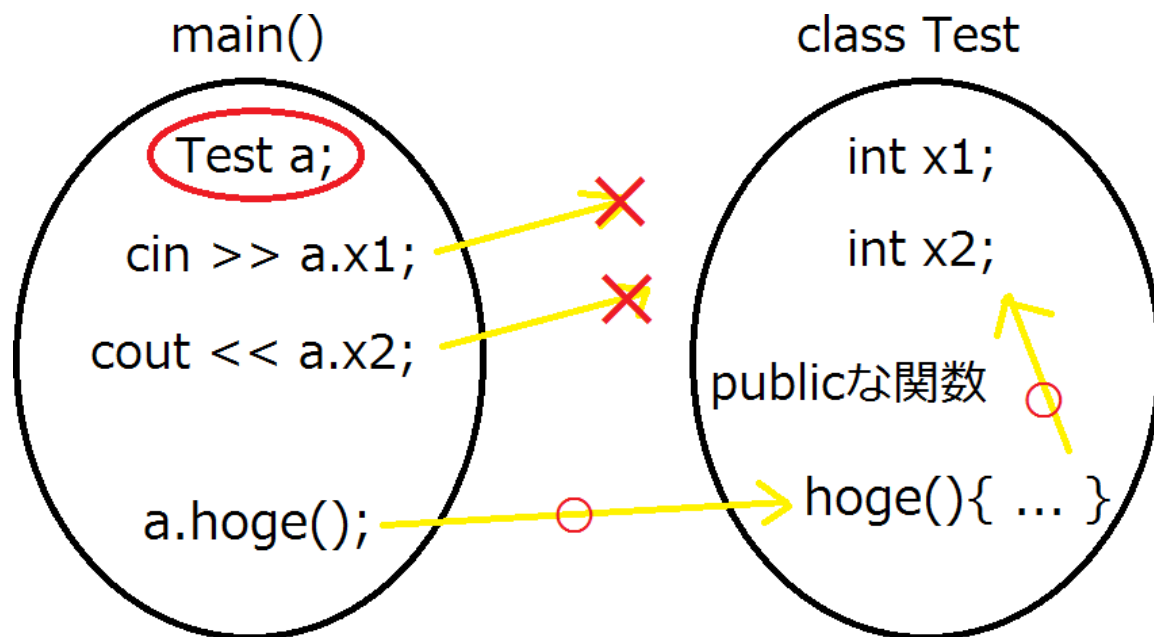
クラスをはじめよう：オブジェクト指向

main関数を含むクラス外の関数から、

クラスのメンバー変数へアクセスしたいとき

⇒ クラスのメンバー関数を必ず通す

メンバー関数を経由すれば、メンバー変数の値を参照できる状態

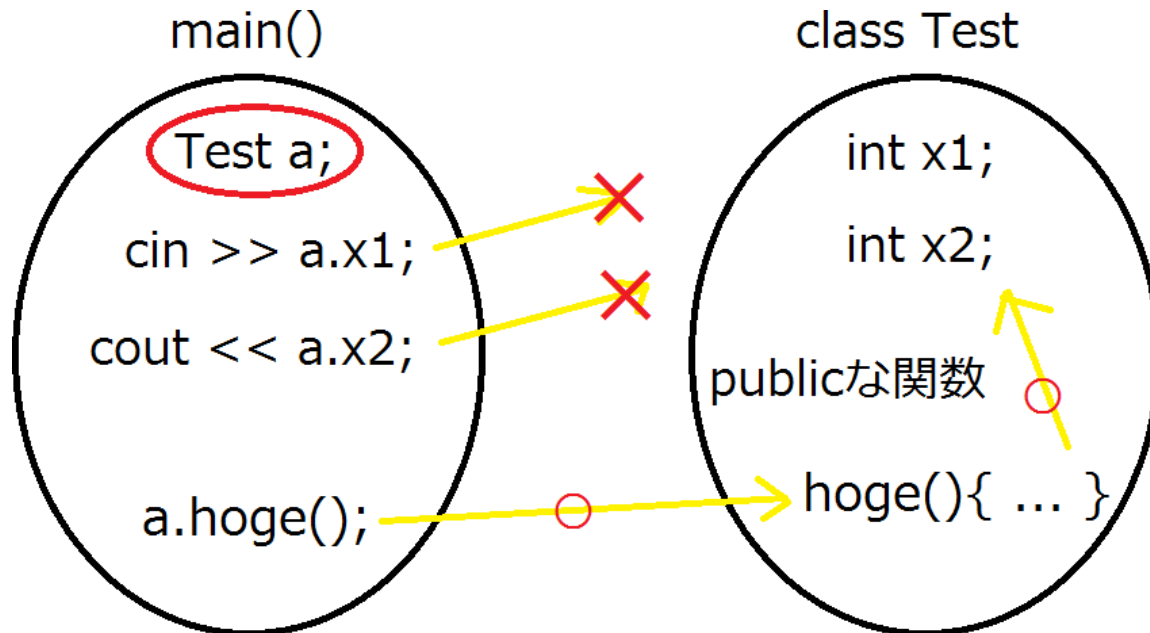


オブジェクト指向：private変数へのアクセス

アクセス関数（と一般に呼ばれるただの関数）を作る

（**クラス外の関数と、privateなメンバー変数を繋ぐ役割**）

- ・ゲッター：あるメンバー変数をreturn するだけの関数
- ・セッター：あるメンバー変数にクラス外の関数から値を代入する
だけの関数



オブジェクト指向：private変数へのアクセス

・値を1個読み取り、クラスのメンバーに代入した後、それを読み出し表示

```
Class TestClass{  
private:  
    int a;  
public:  
    int get (){ //ゲッター  
        return a;  
    }  
    void set(int x){ // セッター  
        a=x;  
    }  
}
```

```
int main(){  
    TestClass obj;  
    int val, result;  
  
    cin >>val;  
    obj.set( val );  
  
    result=obj.get();  
    cout <<result;  
    return 0;  
}
```

オブジェクト指向は面倒くさい？

直感的には、操作が2度手間になっている

こういう考え方（概念）

- ◀ クラスのユーザーはクラスの詳細な設計を知る必要がない
 - ▶ **メンバー関数（必要な機能）の使い方だけ**知ればよい
 - ▶ ⇒ 細部に手を出して、無用なバグを生むリスクを減らす
 - ▶ ⇒ この演習のEx04までの基本スタイル(string, STL)
- ◀ クラスの設計者にとってはメンテナンスがしやすい（保守性）
 - ▶ ⇒ 変数の名前を変えないといけなくなった
 - ▶ ⇒ 細かい仕様変更が発生した
 - ▶ ⇒ **クラスの利用者側の変更が最小限で済む**

オブジェクト指向の諸概念

このように、クラスのメンバー変数は
外部からは隠蔽し、アクセスできなくする

⇒ **カプセル化、情報の保護**

アクセス関数を始めとする、
ユーザーが実際にそのクラスを利用時に使用する関数

⇒ **インターフェース**

内部の詳細な構造ではなく、
インターフェースでクラスの特徴を定義できる

⇒ **データの抽象化**