情報工学演習I
第９回

C++の演習1（クラス）

授業の予定（後半）

#	月日	内容	担当者
7	11月13日	C言語の演習4 (ポインタの演算，列挙型)	内海
8	11月20日	C言語の演習課題	内海
9	11月27日	C++の演習1（クラス）	岩村
10	12月 4日	C++の演習2（クラスの継承）	岩村 (代理：谷川)
11	12月11日	C++の演習3（仮想関数）	岩村
12	12月18日	C++の演習4（インライン展開）	岩村
13	1月 8日	C++の演習5 (関数のオーバロード)	岩村
14	1月15日	C++の演習課題	谷川
15	1月22日	総合演習	谷川

参考書（追加）

- 参考書
  - (D)「新版 明解C++　入門編」
    - 柴田 望洋 (著)

今日の内容

- 第6回演習課題の解説
- Hello C++
- オブジェクト指向
- クラス
- クラスの分割コンパイル

第6回演習課題の解説

第６回演習課題

1. 第５回演習課題のプログラムのコンパイルをmakeを使って行え．そのとき用いたMakefile を提出せよ．

- コンパイルに必要な作業
  - 関数の定義が書かれた.c ファイル(arithmetic.c)をコンパイル
  - main関数のあるファイル(main.c)をコンパイル
  - main とarithmetic を統合

第６回演習課題

2. 引数を１次元配列とし，N 次元ベクトルの内積と外積を計算する関数を定義した.c ファイル，関数の宣言した.h ファイル，関数を利用するmain関数を含む.c ファイルを作成し，関数の挙動をmain 関数の中で確認せよ．ただし，ベクトルの次元数はmain 関数を含むファイルの中のグローバル変数として定義する.
   - extern を使うこと
   - 特徴ベクトルの次元数をmain 関数のファイル内(main.c)でグローバル変数として宣言
   - 関数の定義がある.c ファイル内でextern 変数として宣言

第６回演習課題（２）

3. N 人の学生がN 個の研究室にそれぞれ１人ずつ配属されるとする．学生は研究室を，研究室は学生を希望順に順位付けする．このとき，この順位表をもとにして最適な配属を見いだすプログラムを作れ．
   - ヒント）学生nが順位１位の研究室を希望し，研究室側は希望者がいない場合はとりあえずOK する．もし，研究室に希望者が既にいた場合は，研究室の学生の順位で新しく希望した学生の方が順位が高ければ，以前OK していた学生を手放し，新しく希望した学生にOK を出す．

課題３の基本方針

- 配列で学生，研究室の希望順位をそれぞれ指定
- 全学生が研究室を決定するまで，以下を行う
  - 学生のなかで，研究室が決まっていない学生のうち，順位を１つ下げて（初期の場合は第１志望）の研究室に志願する
  - 志願先の研究室の学生が決まっていないなら，無条件で研究室が決定
  - 既に決まっている学生がいる場合，研究室が作成した学生の順位で既に決まっている学生よりも上位であれば，新しく希望した学生が新たに研究室の学生として決定される．このとき，既に決まっていた学生は研究室未定となる．

Hello C++

C++

- C言語の拡張
  - クラス、仮想関数、多重定義、多重継承、テンプレート、例外処理などの導入
  - C++の当初の名前は「C with Classes」だった

- C++は、基本的にC言語の上位互換
  - ほとんどC言語で、一部だけC++のソースコードも書ける
  - C++で導入された拡張をうまく使えば、より便利にわかりやすく！
    - より直感的に
    - バグを減らす工夫
    - ソースコードの再利用可能性

C++のお約束

- ソースコードのファイル名
  - .ccまたは.cppで終わる
- コンパイル
  - gccの代わりにg++ を使う

- 例：g++ hello_c++.cc

Hello world of C++

Hello_c++.cc

```
#include <iostream> // Cのstdio.hに相当
using namespace std; // 名前空間stdを使う。とりあえずお約束

int main() {
  cout << "Hello C++!" << endl; // 出力の構文。 coutは標準出力

  int n; // 宣言は、使う前ならプログラムのどこにあってもよい
  cout << "n = ";
  cin >> n; // 入力の構文。 cinは標準入力
  int s = 0;
  for (int i=1; i<=n; i++) s+=i;
  cout << "sum(1.." << n << ") = " << s << endl;
  return 0;
}
```

C++の流儀 1

- C++の入出力
  - 出力
    - 標準出力：cout
    - 標準エラー出力：cerr
    - （改行：endl）

    例：cout << “hello” << endl;
  - 入力
    - 標準入力：cin

    例：int n; cin >> n;

標準出力と標準エラー出力の違いは、すぐに出力されるかどうか
（標準エラー出力はすぐに出力されるが、標準出力はバッファが一杯になるまで出力されない）

endlの代わりに¥nを使うことも出来る

coutやcinは変数の型を指定しなくてもいい

Cスタイルのprintf、scanfなども使える

C++の流儀２

- ヘッダのinclude
  - C++特有の関数を使いたければ、C++用のヘッダをincludeする

  例：#include <iostream>
  - Cの関数を使う場合は、C用のヘッダもincludeする

  例：#include <iostream>
  #include <stdio.h>

- 変数の宣言
  - C言語の場合：関数やブロックの最初でないと宣言できなかった
  - C++の場合：どこでも可能

C++の流儀３

using namespace std;
をお約束としてつけておこう

- 名前空間（Namespace）
  - 同名の変数や関数を区別する工夫
  - 標準関数は、名前空間stdに属している
    - coutの正式名称はstd::cout
    - いちいちstd::を付けるのが面倒な場合は、
      using namespace std;
      を付ければ省略可能

2つの京橋

東京の京橋

大阪の京橋

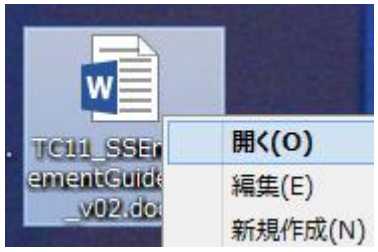
「これから話すことは全部東京のことだから」と宣言すれば、いちいち「東京の京橋と言わなくても一意に意味が通る」

- コメント
  - //が使える
  - Cスタイルの/* */も使える

オブジェクト指向

オブジェクト指向プログラミング

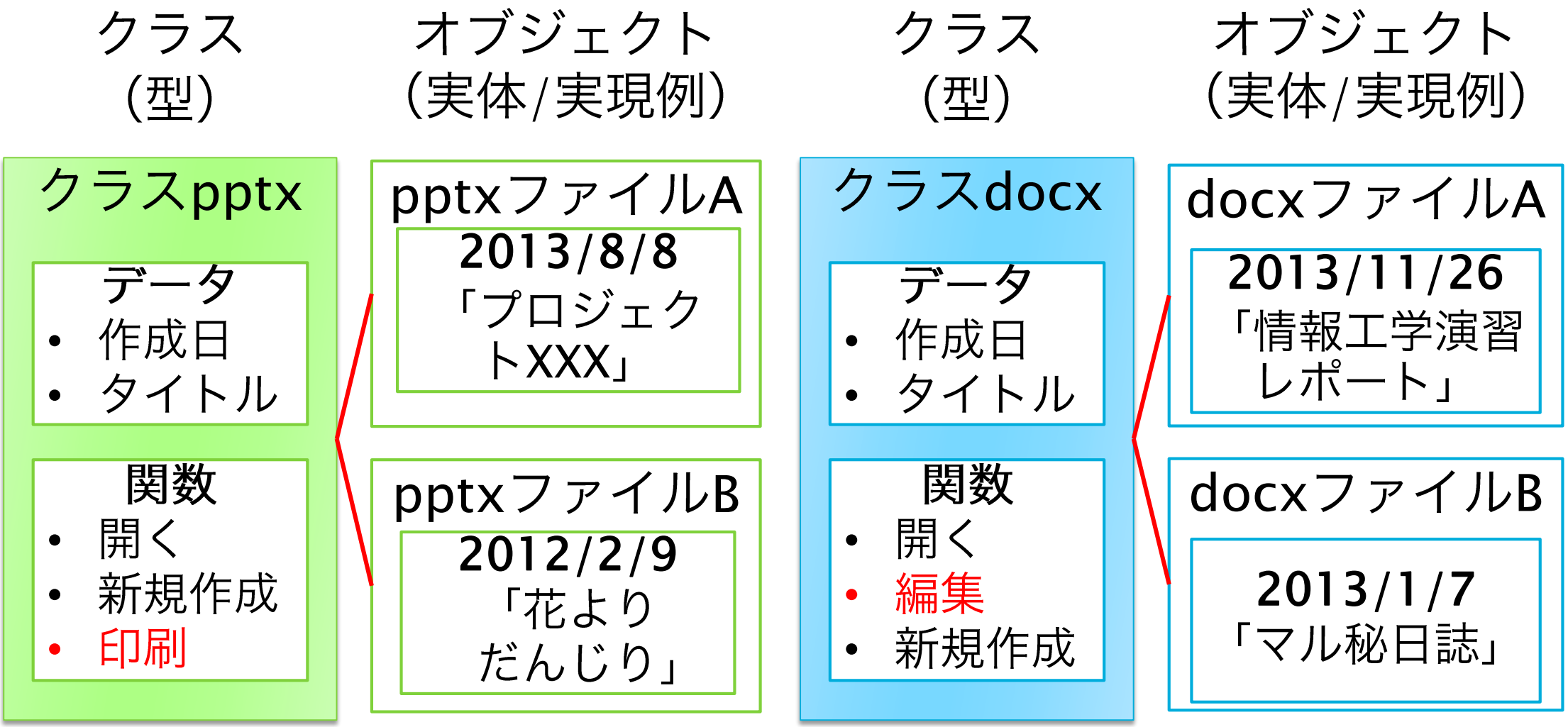
- オブジェクト指向とは？
  - 一言で言えば、データ中心の考え方
    - データと関数をひとまとまりにして扱い、同じ関数を呼び出した場合でも、データに応じて挙動が変わる
  - 例（プログラムではないけれど）
    - Windowsでファイルを選択し、右クリックして「開く」を選んだときの挙動はファイルによって異なる



オブジェクト指向プログラミング（続き）

- オブジェクト指向の例

プログラムの例ではないけれど



オブジェクト指向プログラミング（続き）

- どうしてオブジェクト指向が必要？
  - 大規模なプログラムは一人で作れない
    - 全員がプログラム全体を理解するのは大変！
      効率よく分業したい
    - そのために、各人が担当箇所をプログラムして、できたプログラムを簡単に組み合わせられるようにしたい
  - プログラムを少し修正しただけで、プログラム全体を見直すのは最悪
    - プログラムの修正は最小限度にしたい
    - 過去に作ったプログラムを使い回すときにも便利
  - バグがいっぱい起こる
    - プログラムが読み易くなれば、バグが起こりにくくなるし、バグを修正しやすくなる

クラス

残高照会プログラム（クラスを使わない場合）

ex1_wo_class.cc

```
#include <string> // stringを使うために必要
#include <iostream> // 入出力に必要
using namespace std; // お約束

int main() {
 string suzuki_name = "鈴木龍一"; // 鈴木さんの名前
  int suzuki_balance = 123000; // 鈴木さんの残高
  string tanaka_name = "田中恵美"; // 田中さんの名前
  int tanaka_balance = 256000; // 田中さんの残高

  suzuki_balance += 10000; // 鈴木さんの残高を10000円増やす
  tanaka_balance -= 2000; // 田中さんの残高を2000円減らす

  cout << suzuki_name << “様の残高は" << suzuki_balance << "円です．" << endl;
  cout << tanaka_name << "様の残高は" << tanaka_balance << "円です．" << endl;

  return 0;
}
```

残高照会プログラム（クラスを使う場合）

ex2_w_class.cc

```
#include <string>
#include <iostream>
using namespace std;

class Account {
public:
  string name; // 名前
  int balance; // 残高
};

int main() {
  Account suzuki; // 鈴木さんの口座のオブジェクト
  Account tanaka; // 田中さんの口座のオブジェクト
  suzuki.name = "鈴木龍一"; // 鈴木さんの名前
  suzuki.balance = 123000; // 鈴木さんの残高
  tanaka.name = "田中恵美"; // 田中さんの名前
  tanaka.balance = 256000; // 田中さんの残高

  suzuki.balance += 10000; // 鈴木さんの残高を10000円増やす
  tanaka.balance -= 2000; // 田中さんの残高を2000円減らす

  cout << suzuki.name << "様の残高は" << suzuki.balance << "円です。" << endl;
  cout << tanaka.name << "様の残高は" << tanaka.balance << "円です。" << endl;

  return 0;
}
```

クラス

- クラスは、特定の「もの」を表す型
- クラスの宣言
  - クラス名
    - ここでは銀行口座を意味する「Account」
  - データメンバ（クラスに属する変数）
    - ここでは、以下の2つ
      - string型のname
      - int型のbalance

実は構造体と同じ

クラスの宣言はclassから始まる

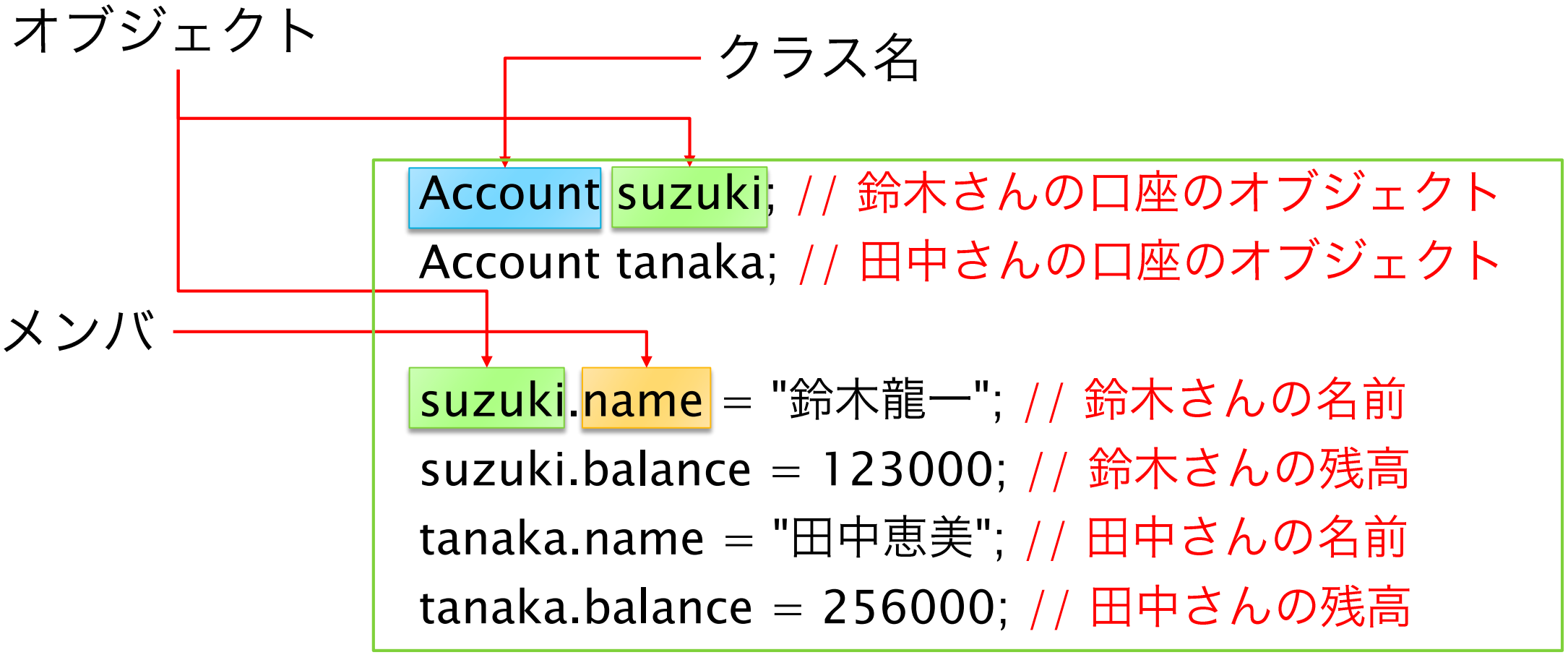
ひとまずお約束

```
class Account {
public:
  string name; // 名前
  int balance; // 残高
};
```

セミコロンを忘れない！

オブジェクト

- オブジェクト（インスタンスとも言う）
  - 特定のクラスの性質を持つ実現例



参考：classって実は構造体？

- classをstructに変えても
  同じ事ができる
  （試してみよう！）

```
struct Account {
public:
  string name; // 名前
  int balance; // 残高
};
```

- C言語では、構造体を宣言するとき、
  struct Account suzuki;
  のようにした
- C++では、structを省略して、
  Account suzuki;
  とできる

残高照会プログラム（メンバ関数を使う場合）

ex3_w_class2.cc

```
#include <string>
#include <iostream>
using namespace std;

class Account {
private:
  string name; // 名前
  int balance; // 残高

public:
  // コンストラクタ
  Account(string _name, int _balance) {
    name = _name; // 名前を初期化
    balance = _balance; // 残高を初期化
  }
  // 名前を調べる
  string get_name() {
    return name;
  }
  // 残高を調べる
  int get_balance() {
    return balance;
  }
  // 預ける
  void deposit(int amnt) {
    balance += amnt;
  }
  // おろす
  void withdraw(int amnt) {
    balance -= amnt;
  }
};
```

残高照会プログラム（メンバ関数を使う場合）

続き

ex3_w_class2.cc

```
int main() {
  Account suzuki(“鈴木龍一”, 123000); // 鈴木さんの口座のオブジェクト作成
  Account tanaka("田中恵美", 256000); // 田中さんの口座のオブジェクト作成

  suzuki.deposit(10000); // 鈴木さんの残高を10000円増やす
  tanaka.withdraw(2000); // 田中さんの残高を2000円減らす

  cout << suzuki.get_name() << "様の残高は"
       << suzuki.get_balance() << "円です．" << endl;
  cout << tanaka.get_name() << "様の残高は"
       << tanaka.get_balance() << "円です．" << endl;

  return 0;
}
```

クラス・その２

- クラスはデータメンバだけでなく、
  メンバ関数も持てる

- クラスの宣言again
  - メンバ関数
    （クラスに属する関数）
    - ここでは、以下の4つ
      - get_name
      - get_balance
      - deposit
      - withdraw

```
 // 名前を調べる
  string get_name() {
    return name;
  }
 // 残高を調べる
  int get_balance() {
    return balance;
  }
  // 預ける
  void deposit(int amnt) {
    balance += amnt;
  }
  // おろす
  void withdraw(int amnt) {
    balance -= amnt;
  }
};
```

クラス・その２（続き）

- データメンバ、メンバ関数へのアクセス制御

これ以降に宣言されるものは
クラス外からもアクセス可能

```
class Account {
public:
  string name; // 名前
  int balance; // 残高
};
```

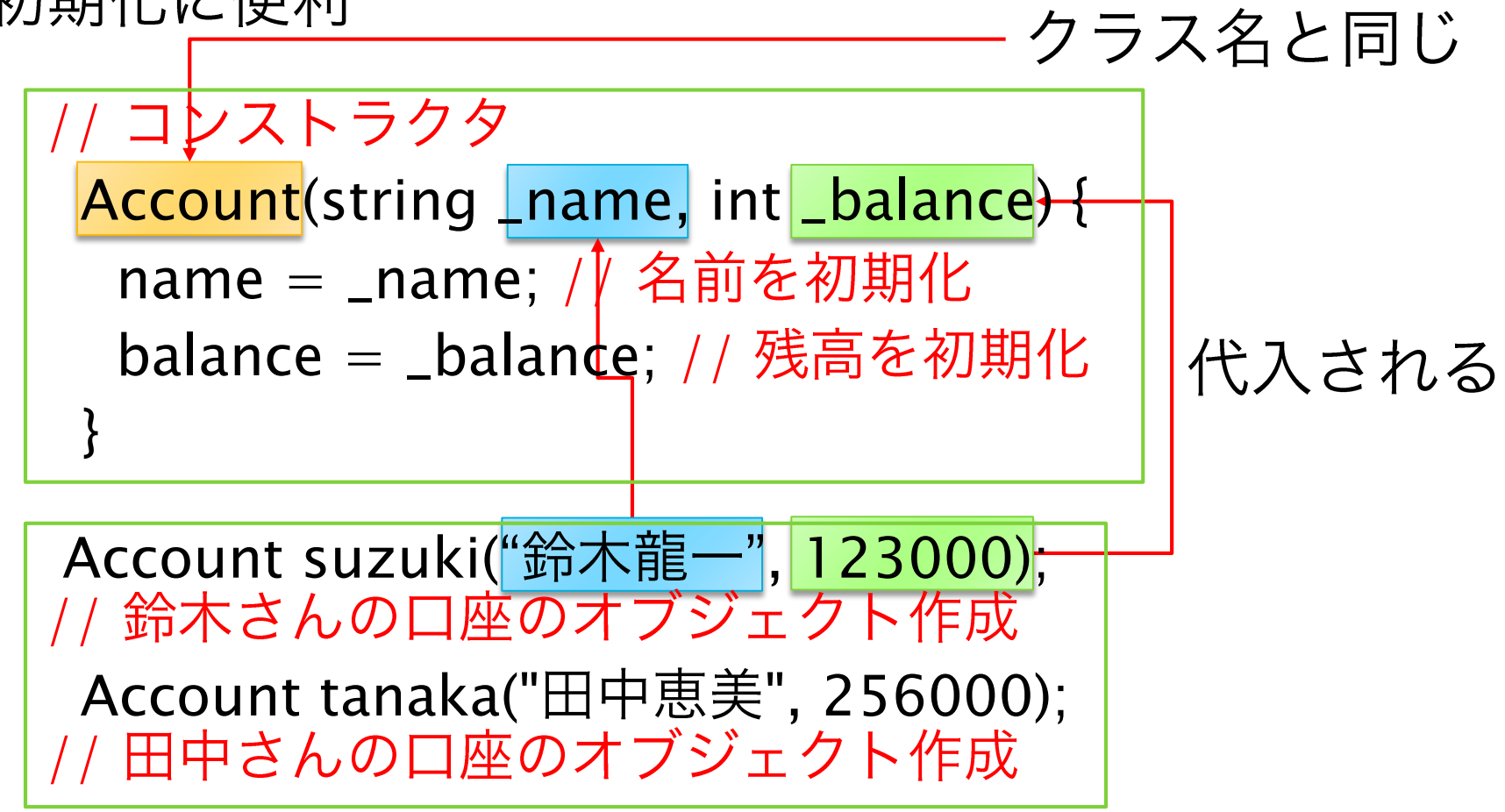
これ以降に宣言されるものは
クラス外からアクセス不可能

```
class Account {
private:
  string name; // 名前
  int balance; // 残高
};
```

試しに、ex2_w_class.ccのpublicを
privateにしてみると、エラーになる

クラス・その２（さらに続き）

- コンストラクタ
  - オブジェクト作成時に呼ばれる関数
  - 変数の初期化に便利



クラス・その２（さらにさらに続き）

- デフォルトコンストラクタ
  - 引数なしで呼び出せるコンストラクタ
  - 明示的に定義しないときは、自動的に作られる

```
// デフォルトコンストラクタの例
  Account() {
}
```

クラスの分割コンパイル

残高照会プログラム（分割コンパイル、タイプ１）

ex4_account.h

```
#ifndef ex4_account_h
#define ex4_account_h
#include <string>

class Account {
private:
  std::string name; // 名前
  int balance; // 残高

public:
  // コンストラクタ
  Account(std::string _name, int _balance) {
    name = _name; // 名前を初期化
    balance = _balance; // 残高を初期化
  }
  // 名前を調べる
  std::string get_name() {
    return name;
  }
  // 残高を調べる
  int get_balance() {
    return balance;
  }
  // 預ける
  void deposit(int amnt) {
    balance += amnt;
  }
  // おろす
  void withdraw(int amnt) {
    balance -= amnt;
  }
};
#endif
```

残高照会プログラム（分割コンパイル、タイプ１）

続き

ex4_main.cc

```
#include <iostream>
#include "ex4_account.h" // Accountクラスの定義を読み込む
using namespace std; // ヘッダには書かない方がいい

int main() {
  Account suzuki("鈴木龍一", 123000); // 鈴木さんの口座のオブジェクト作成
  Account tanaka("田中恵美", 256000); // 田中さんの口座のオブジェクト作成
  suzuki.deposit(10000); // 鈴木さんの残高を10000円増やす
  tanaka.withdraw(2000); // 田中さんの残高を2000円減らす

  cout << suzuki.get_name() << "様の残高は"
       << suzuki.get_balance() << "円です．" << endl;
  cout << tanaka.get_name() << "様の残高は"
       << tanaka.get_balance() << "円です．" << endl;

  return 0;
}
```

クラスの分割コンパイルの方法１（クラスの定義ファイルを分割しない場合）

- g++ ex4_main.ccでコンパイル

- クラスの定義はヘッダファイル（.h）に書く
- ヘッダファイルの中に
  using namespace std;
  は書かない方がいい

- クラスを使うプログラムでは、ヘッダファイルをincludeする
- ヘッダファイルを２回includeするとエラーになるので、それを避けるおまじないをヘッダファイルに書いておくと安心

```
#ifndef ex4_account_h
#define ex4_account_h


#endif
```

残高照会プログラム（分割コンパイル、タイプ２）

ex5_account.h

```
#ifndef ex5_account_h
#define ex5_account_h


#include <string>


class Account {
private:
  std::string name; // 名前
  int balance; // 残高
```

```
public:
  // コンストラクタ
  Account(std::string _name,
int _balance);
  // 名前を調べる
  std::string get_name();
  // 残高を調べる
  int get_balance();
  // 預ける
  void deposit(int amnt);
  // おろす
  void withdraw(int amnt);
};


#endif
```

残高照会プログラム（分割コンパイル、タイプ２）

ex5_account.cc

```
#include "ex5_account.h" // Account クラスの定義を読み込む


// コンストラクタ
Account::Account(std::string _name, int _balance) {
  name = _name; // 名前を初期化
  balance = _balance; // 残高を初期化
}
// 名前を調べる
std::string Account::get_name() {
  return name;
}
```

```
// 残高を調べる
int Account::get_balance() {
  return balance;
}
// 預ける
void Account::deposit(int amnt) {
  balance += amnt;
}
// おろす
void Account::withdraw(int amnt) {
  balance -= amnt;
}
```

残高照会プログラム（分割コンパイル、タイプ２）

続き

ex5_main.cc
（２行目以外は
ex4_main.ccと同じ）

```
#include <iostream>
#include "ex5_account.h" // Accountクラスの定義を読み込む
using namespace std; // ヘッダには書かない方がいい

int main() {
  Account suzuki("鈴木龍一", 123000); // 鈴木さんの口座のオブジェクト作成
  Account tanaka("田中恵美", 256000); // 田中さんの口座のオブジェクト作成
  suzuki.deposit(10000); // 鈴木さんの残高を10000円増やす
  tanaka.withdraw(2000); // 田中さんの残高を2000円減らす

  cout << suzuki.get_name() << "様の残高は"
       << suzuki.get_balance() << "円です．" << endl;
  cout << tanaka.get_name() << "様の残高は"
       << tanaka.get_balance() << "円です．" << endl;

  return 0;
}
```

クラスの分割コンパイルの方法２（クラスの定義ファイルを分割する場合）

- コンパイルの方法（以下を順番に）
  - g++ -c ex5_main.cc
  - g++ -c ex5_account.cc
  - g++ ex5_main.o ex5_account.o

ex5_main.oができる

ex5_account.oができる

- ヘッダファイル（.h）の中には関数の宣言のみ書く

```
// コンストラクタ
Account(std::string _name, int _balance);
// 名前を調べる
std::string get_name();
```

- 関数の定義は別ファイル（ex5_account.cc）に書く
  - 関数の前にクラス名が必要

```
// コンストラクタ
Account::Account(std::string _name, int _balance) {
…
}
```

演習課題

第９回演習課題（１）

- １. 以下の仕様を満たすプログラムを作れ
  - 以下の仕様を満たすクラスを持つ
    - 人の名前を保存することができる
    - 数学、理科、英語の点数を保存することができる
    - ３教科の点数を変更（上書き）できる関数と照会できる関数がある
    - ３教科の点数の平均を計算して返す関数がある
  - 上記のクラスを用いて、２人分のデータ（名前、３教科の点数）を順次コマンドラインから入力できる
  - 全員分の情報を入力した後、一人ずつ名前と平均点を表示する

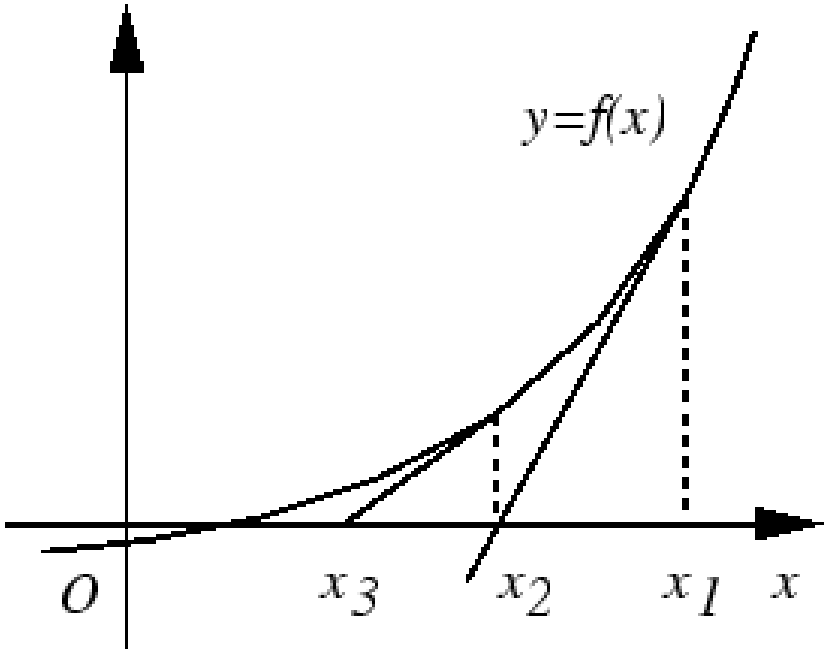
第９回演習課題（２）

- 2. 以下の仕様を満たすプログラムを作れ
  - 以下の仕様を満たすクラスを持つ
    - ２次元の座標を保持することができる
    - 座標の初期値はオブジェクト作成時に与える
    - 現在位置を更新（上書き）する関数を持つ
    - 現在位置を返す関数を持つ
    - 呼び出されると上下左右にそれぞれ移動する関数を持つ
  - コマンドラインから上下左右に移動する命令を受け取り、上記クラスの関数を呼び出して、移動させる
    - 命令の例：up, down, left, right
  - コマンドラインから現在位置を照会する命令を与えたときは、現在位置を返す

第９回演習課題（３）

- 3. ニュートン法で方程式を解くプログラムを作れ。ただし、以下の仕様を満たすものとする
  - 方程式は$ax^3+bx^2+cx+d=0$とし、a, b, c, dはコマンドラインから入力

ニュートン法：
曲線の接線とx軸の交点を求める処理を何度も繰り返して解を導く方法



何かを参考にした場合は、その旨を述べる。書いてない場合は０点

提出に関して

- 提出するもの
  - ソースファイル(.ccまたは.cpp ファイル)
    - ファイル名はkadai1127_学籍番号_課題番号.cc（.cpp）
    - (Visual Studioの場合)
      ファイル名はkadai1127_学籍番号_課題番号_v.cc
      （.cpp）
  - 実行結果の出力と講義に関するコメント
    - .txt ファイルで、学籍番号、氏名を含む
    - ファイル名はreport1127_学籍番号.txt とする

提出に関して（続き）

- 提出期限
  - 12月11日（水）　００：００
- 提出方法
  - 授業支援システムから提出
- 注意点
  - ファイル名の命名規則が間違っているものは採点しない
  - コンパイルの通らないものは採点しない