Programming C++
Lecture Note 9
新しいクラス型を定義する（一部復習）

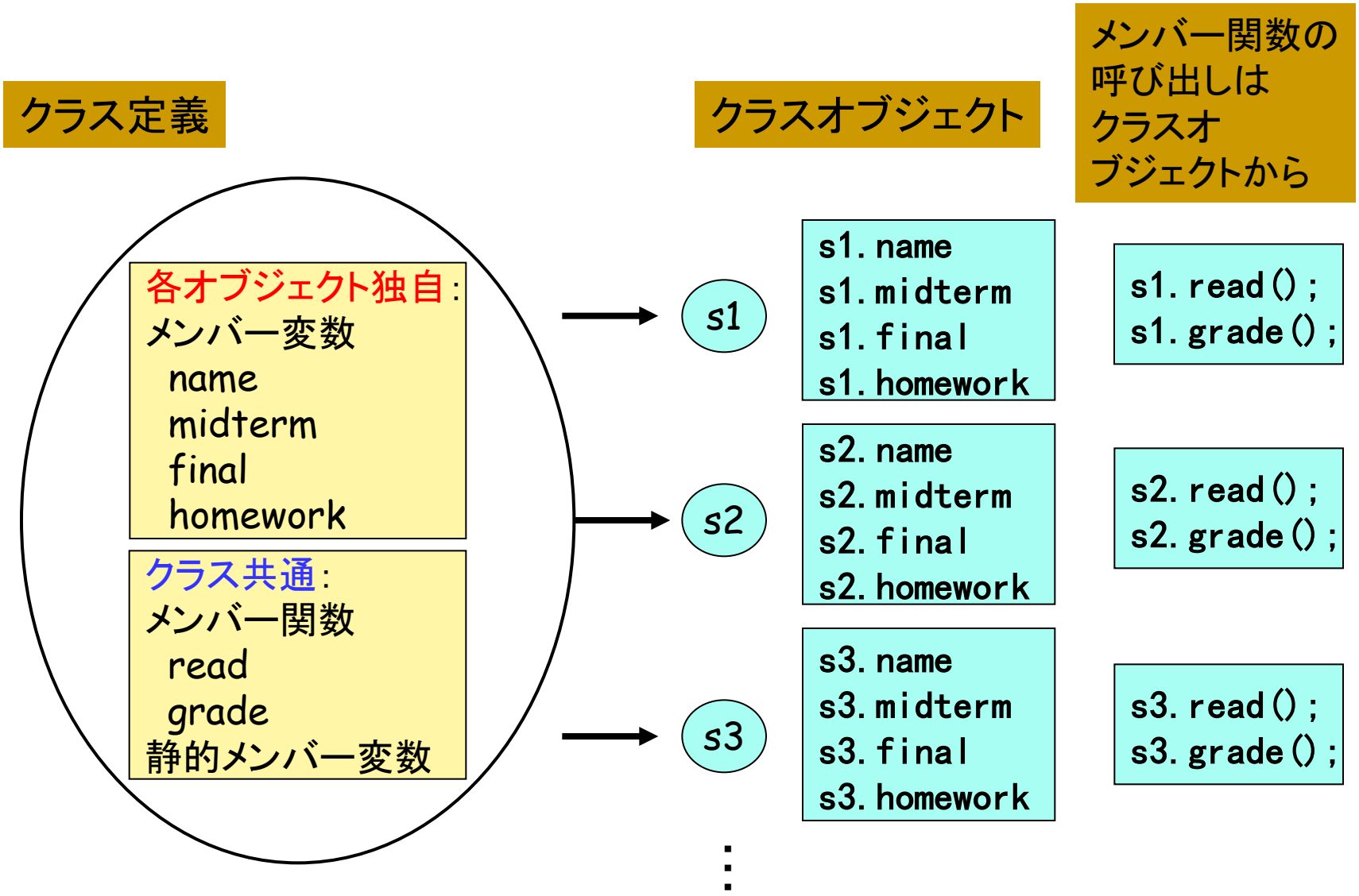
Jie Huang

クラス型はデータと関数を含む

- structを使ったクラス定義の例

```
struct Student_info {
    //メンバー変数
    std::string name;
    double midterm, final;
    std::vector<double> homework;
    //メンバー関数
    std::istream& read(std::istream&);
    double grade() const;
};
```

クラスとクラスオブジェクト



メンバー関数のデフォルトの引数

- 一般関数

  read(s1, cin);

- クラスメンバー関数

  s1.read(cin);

  ここで、クラスオブジェクトのs1は明示的に引数として渡す必要がない。

クラスメンバー関数の定義

- クラス内でのメンバー関数定義

```
struct Student_info {
    //メンバー変数
    std::string name;
    double midterm, final;
    std::vector<double> homework;
    //クラス内でのメンバー関数定義
    std::istream& read(std::istream& in){
      in >> name >> midterm >> final;
      read_hw(in, homework);
      return in;
    }
     double grade() const;
};
```

クラスメンバー関数の定義

- スコープ演算子を使ったクラス外でのメンバー関数定義

```
//クラス外でのメンバー関数定
std::istream& Student_info::read(std::istream& in){
  in >> name >> midterm >> final;
  read_hw(in, homework);
  return in;
}
```

クラスメンバー関数と一般関数

- クラスメンバー関数にするべき関数
  - クラスオブジェクトの内容を変更する関数

    例えば:

    ```
    std::istream& Student_info::read(std::istream&);
    ```
  - クラスのインターフェースを提供する関数

    例えば:

    ```
    double Student_info::grade() const {…}
    s1.grade(); // s1は変更されない
    ```

    参考:一般関数としてのgrade関数

    ```
    double grade(const Student_info&){…}
    ```

クラス内部のデータ保護

- クラス内で外部に公開する部分と隠す部分に分ける

```
class Student_info {
  private:
    //ここは私のプライベート
    std::string name;
     double midterm, final;
     std::vector<double> homework;
  public:
     //この部分はどうぞ使ってください
    std::istream& read(std::istream&);
    double grade() const;
};
```

クラス外で以下の使い方はエラーとなる

```
s1.name, s1.midterm, s1.final, s1.homework
```

- classはデフォルトでprivateであるのに対して、structはデフォルトでpublicとなる。

クラスのデータについてのアクセス

- クラスへのアクセス方法を定義するインターフェース
  - アクセス関数

```
class Student_info {
   public:
     //このインターフェース部分はどうぞ使ってください
     std::istream& read(std::istream&);
     double grade() const;
     std::string name() const { return n; }
     bool valid() const { return !homework.empty(); }
   private:
     //ここは私のプライベート
     std::string n; double midterm, final;
     std::vector<double> homework;
};
```

クラスについての情報はアクセス関数を通して取り出す。

例えば、s1.nameの代わりにs1.name()となる。

クラスオブジェクトの初期化

- コンストラクタはクラス名と同じの初期化専用関数で、
- クラスオブジェクトが生成される時に自動的に呼び出される。
- デフォルト(引数を取らない)コンストラクタ

  Student_info::Student_info():midterm(0), final(0) {}

  このコンストラクタは以下の様な場合に呼び出される

  Student_info s;

  ここで、midterm(0)とfinal(0)は初期化子と呼ばれ、データメンバーの初期化に使われる。なお、homeworkは自身のコンストラクタを持っているので、非明示的に初期化される。
- 引数を取るコンストラクタ

  Student_info::Student_info(std::istream& is){ read(is);}

  このコンストラクタは以下の様な場合に呼び出される

  Student_info s(cin);

クラスオブジェクトの初期化過程

- オブジェクトのメモリ領域が確保される
- コンストラクタの初期化子に従って、クラスメンバーの初期化が行われる。
- コンストラクタが実行され、初期化が行われる。

コンストラクタはオープンインターフェース

- コンストラクタはクラスオブジェクトの生成に使われるので、オープンで、publicラベルの下に置く必要がある

```
class Student_info {
public:
   Student_info();
   Student_info(std::istream&);
   …
};
```

- もし、コンストラクタをprivateに下においてしまったら、オブジェクトの生成ができなくなるので、意図的にオブジェクトを生成できないようにする以外では、コンストラクタはpublicの部分に置く必要がある。

新しいクラスを使う

```
int main(){
   vector<Student_info> students;
   Student_info record; string::size_type maxlen = 0;
   while (record.read(cin)) {
        maxlen = max(maxlen, record.name().size());
        students.push_back(record);}
   sort(students.begin(), students.end(), compare);
   for (vector<Student_info>::size_type i = 0;
     i != students.size(); ++i) {
     cout << students[i].name() <<
       string(maxlen + 1 - students[i].name().size(), ' ');
     try {double final_grade = students[i].grade();
       streamsize prec = cout.precision();
       cout << setprecision(3) << final_grade
                       << setprecision(prec) << endl;
     } catch (domain_error e) { cout << e.what() << endl;}
   }
   return 0;}
```

静的メンバー変数

- 静的メンバー変数を含むクラス

```
class S {
public:
  static int num;
};
int S::num = 0;
```

ここで、int型の変数numは静的メンバー変数で、その実体はクラスにおいてただ1つだけで、クラスオブジェクト共通のものとなる。
静的メンバー変数の実体の定義と初期化はクラスの外にて行う。

- 静的メンバー変数の参照は普通のメンバー変数と同じくオブジェクトからメンバー演算子で参照するか、クラススコープ演算子を使って参照することができる。

```
int main() {
  S a; a.num = 1;
  cout << S::num << endl;
}
```

静的メンバー関数

- 静的メンバー変数とメンバー関数を含むクラス

```
class S {
public:
  static int num;
  static int get_num () { return num;}
};
int S::num = 0;
```

ここで、関数int get_num()は静的メンバー関数である。

- 静的メンバー関数はクラスオブジェクトではなく、クラスそのものに付随する。従って、静的メンバー関数はクラスオブジェクトが1つも生成されていない状態でも呼び出すことができる。
- 静的メンバー関数はthisポインタを使うことはできないし、普通(静的でない)のメンバー変数に直接アクセスすることもできない。
- 静的メンバー関数の呼び出しはクラスオブジェクトからメンバー演算子で行うか(thisポインタは受け継がない)、スコープ演算子を付けて外部関数と同じように呼び出す。

静的メンバー関数

- 静的メンバー変数とメンバー関数を含むクラス

```
class S {
public:
  static int num;
  static int get_num () { return num;}
};
int S::num = 0;

int main() {
  cout << S::get_num() << endl;
  S a; a.num = 1;
  cout << a.get_num() << endl;
}
```

- この仕組みを利用すれば、例えばクラスのオブジェクトをいくつ生成されたかを管理することができるようになる。