第9章「新しい型を定義する」

主なトピック

- クラスの定義

クラスの作り方

クラスの使い方

Student_info 構造体再掲

- 以前定義した構造体

```
struct Student_info {
   std::string FirstName, LastName, ID;
   double Midterm, Final;
   std::vector<double> Homework;
}
```

- 完全修飾型 `std::string` や `std::vector` を使う理由
  - 色々なプログラムやユーザに使われる
  - 構造体内で使われる `string` や `vector` は内部仕様で，この構造体を使うユーザに `using` を強要できないため

構造体のメンバ関数

- 構造体のデータ(メンバ変数)を操作する関数

```
struct Student_info {
   std::string FirstName, LastName, ID;
   double Midterm, Final, Ex;
   std::vector<double> Homework;

   //  入力ストリームからデータを読み込み,
   //  この構造体に格納する関数
   std::istream& read(std::istream&);
   //  最終成績を計算する関数
   double grade() const;
}
```

read関数の例

- メンバ関数のread関数

```
std::istream& Student_info::read( std::istream& in )  {
   in >> FirstName >> LastName >> ID;
   read_hw( in, Homework );
   return in;
}
```

- 従来の通常関数のread関数

```
std::istream& read( std::istream& in, Student_info& s ) {
   in >> s.FirstName >> s.LastName >> s.ID;
   read_hw( in, s.Homework );
   return in;
}
```

メンバ関数と通常の関数の違い

- 関数名
  - read から Student_info::read
  - クラス名::メンバ関数名
- 引数
  - read(in, s) から read(in)
  - 読み込み結果は，このオブジェクト（変数）に格納
- 関数の定義の内部
  - s.FirstName から FirstName
  - オブジェクトの要素に直接アクセス可能

メンバ関数のread関数の使用例

```
//   データを標準入力ストリームから読み込み
vector<Student_info> students;
Student_info record;

//   recordというオブジェクト（構造体変数）に
//   readというメンバ関数を適用
//   その結果はrecordに反映される

while( record.read( cin ) ) {
    students.push_back( record );
}
```

通常関数のread関数の使用例

```
//   readという通常関数を実行し，
//   その結果を参照型の引数で与えられた
//   recordというオブジェクトに格納

while( read( cin, record, ) ) {
    students.push_back( record );
}

//   メンバ関数でも通常関数でも，結果は同じ
//   recordというオブジェクトに読み込みデータが格納される
//   プログラムの考え方（オブジェクト指向）の違い
```

grade関数の例

```
//  メンバ関数の grade 関数
double Student_info::grade() const {
   return ::grade(Midterm, Final, Homework);
}

//  通常関数の grade 関数
double grade( double midterm, double final, const
   vector<double>& hw )     {
   double ex;
   //  hw(演習)のメジアンを求め, ex に代入
   return 0.2 * midterm + 0.4 * final + 0.4 * ex;
}
```

grade関数の解説

- `double Student_info::grade() const`
  - このメンバ関数はオブジェクトの値を変更しないことを宣言
  - 引数に与える `const` と同じこと
  - `const` メンバ関数
- `::grade()`
  - メンバ関数の`grade`ではなく，（普通の）関数の`grade`を呼び出す

構造体(struct)からクラス(class)へ

```
class  Student_info {
public:
   double grade() const;
   std::istream& read(istream&);
private:
   std::string  FirstName, LastName, ID;
   double Midterm, Final;
   std::vector<double>  Homework;
}
```

データ保護

- public:
  - ユーザが（クラスの外部から）使用可能
  - インターフェース（外部からの操作）
- private:
  - クラスのメンバ関数からのみ利用可能
  - 実装（データなど）
- class と struct の違いは，ほとんどない
  - ラベルがない時のデフォルトは，classはprivate, structはpublicのアクセス制限

アクセス関数

- アクセス制限でデータメンバ（例えば，FirstNameはprivate）を隠蔽
- では，データメンバにアクセスしたいときは，どうする？
- データメンバにアクセスするためのメンバ関数（アクセス関数）を作る
  - メンバ変数を直接操作するのではなく，アクセス関数を経由して操作
  - 不用意にデータ更新されないので，安全

アクセス関数の例

```
class  Student_info {
public:
   //  追加
   std::string  first_name() const { return FirstName; } ;
   std::string  last_name() const { return LastName; } ;
   std::string  id() const { return ID; } ;
   bool valid() const { return !Homework.empty(); };
   //  省略
private:
   std::string  FirstName, LastName, ID;
   double Midterm, Final;
   std::vector<double>  Homework;
}
```

アクセス関数の例の解説

- std::string first_name() const { return FirstName; } ;
  - クラスの内部で，関数の定義を含めることが可能（インライン展開）
  - 関数が戻すのは string型なので，FirstNameのコピーが返される
  - さらに，const が付いているのでメンバ変数は変更されない
  - これによりデータが保護（書き換えられない）

アクセス関数の利用例

```
bool compare( const Student_info& x, const Student_info&
   y)
{
   if( x.first_name() == y.first_name() )
        return x.last_name() < y.last_name();
   else
        return x.first_name() < y.first_name();
}
```

- compare 関数というクラス外部から，アクセス関数を利用してメンバ変数の値を利用可能

チェック用の関数

- bool valid() const;
  - 演習のメジアンの計算をするときに，演習の提出回数が0だと処理できない
    - 以前のプログラムでは例外にしていた
  - クラスの再利用性を考えると，演習が0回のときの処理は，ユーザやプログラムにより様々．
  - そこで，演習が0回かどうかを判定する関数を用意し，実際の処理はユーザにまかせる

コンストラクタ

- 必要となる初期化処理
  - オブジェクトを保持するための，メモリの確保（割付）
  - オブジェクト（データメンバの値）が初期化
- 初期化処理を行う特別なメンバ関数がコンストラクタ
  - コンストラクタの名前は，クラス名と同じ
  - 引数の型や個数に応じて，複数のコンストラクタを定義可能

Student_infoのコンストラクタ

- `Student_info s1;`
  - 変数を宣言するだけ，データは不定
- `Sudent_info s2(cin);`
  - 変数を宣言し，`cin`からデータを読み込み，その値で初期化

```
class  Student_info {
public:
     //   追加
   Student_info(); //   変数宣言だけ
   Student_info( std::istream& ); // データで初期化
     //   省略
}
```

デフォルトコンストラクタ

- 引数を取らないコンストラクタ
- 例えば,

```
Student_info::Student_info()      {
   Midterm = 0;
   Final = 0;
}
```

- MidtermとFinalを0で明示的に初期化
- FirstName, LastName, IDはstringクラスの
- Homeworkはvectorクラスのコンストラクタにより非明示的に初期化

デフォルトコンストラクタ

- 別の書き方，こちらがお勧め
- Student_info::Student_info( ) : Midterm(0), Final(0) { }
- コンストラクタ・イニシャライザ（初期化子），「:」と「{」の間
  - データメンバの値をカッコの中のもので初期化
  - その後に，{}の内部が実行される
  - 同じ処理は{}内部でも行えるが，初期と代入で2度手間になる

引数を取るコンストラクタ

- Student_info::Student_info(istream& is) { read(is); }
- この例では，コンストラクタ・イニシャライザはない

コンストラクタによる値の初期化

- コンストラクタが定義されているクラスなら
  - コンストラクタに指定されている手続きに従って初期化
- 組み込み型なら
  - 値なら0，それ以外は不定値に初期化
- コンストラクタが定義されていないクラスなら
  - データメンバそれぞれのコンストラクタに従って初期化

今回の演習のポイント (1)

```
class Student_info{
public:
  //  インターフェース
  Student_info();
  Student_info( std::istream& );
  std::string first_name() const { return FirstName; };
  std::string last_name() const { return LastName; };
  std::string id() const { return ID; };
  double midterm() const { return Midterm; };
  double final() const { return Final; };
  double ex() const { return Ex; };
  double total() const { return Total; };
```

今回の演習のポイント (2)

```
  std::vector<double> homework() const { return
   Homework; };
  bool valid() const { return !Homework.empty(); };
  std::istream& read( std::istream& );
  double grade();
private:
  //  実装
  std::string FirstName, LastName, ID;
  double Midterm, Final, Ex, Total;
  std::vector<double> Homework;
};
bool compare(const Student_info&, const Student_info&);
```

今回の演習のポイント (3)

```
//   メンバ関数の定義
Student_info::Student_info( ) : Midterm(0), Final(0) { }
Student_info::Student_info( istream& is) { read(is); }
std::istream& Student_info::read( std::istream& is ){
   //   適切なコードを書く
}
double Student_info::grade(){
   //   適切なコードを書く
}
bool compare(const Student_info&, const Student_info&){
   //   適切なコードを書く
}
```

今回の演習のポイント (4)

```
int main()    {
   vector<Student_info> students;
   Student_info record;
   while( record.read( cin ) ) {
        students.push_back( record );
   }
   sort( students.begin(), students.end(), compare);
   for(vector<Student_info>::size_type i = 0; i !=
   students.size(); ++i)    {
        students[i].grade();
        //  結果の出力などvaild()
   }
}
```