第4章「プログラムとデータの構成」

主なトピック

- 構造体の定義の方法（クラスを定義するための準備段階）
- リファレンス（参照）型

プログラムの基本要素

- データ構造
  - 構造体
- 関数
  - 関数の定義，複数ファイルと分割コンパイル
- （クラス＝データ構造＋関数）

- 前回の演習は，データを処理しただけ
- 今回では，それを「構造化」していく
  - データ構造の定義と処理を関数として表現

データ構造

- 各学生のデータ
  - 学生番号: std::string ID;
  - 姓: std::string SurName;
  - 名: std::string GivenName;
  - 中間試験の成績: double Midterm;
  - 期末試験の成績: double Final;
  - 演習の成績: std::vector<double> Exercise;

構造体

- いくつかのデータをまとめて，新しい「型」をつくる
- 構造体の例

```
struct Student_info            {
      std::string  ID;             //  学生番号:
      std::string  SurName;  //  姓
      std::string  GivenName;           //  名
      double Midterm;          //  中間試験の成績:
      double Final;              //  期末試験の成績:
      std::vector<double> Exercise;   //  演習の成績:
};
```

構造体の定義

- struct 構造体の名前　　{
  - 変数の型　変数名;
  - これは，構造体のメンバ変数と呼ばれる
  - メンバ変数は，あるだけ続く
- };
- 構造体の定義の最後の「;」（セミコロン）を忘れずに
- 構造体の各メンバ変数へのアクセスの方法は
  - 構造体の変数名.メンバ変数名
  - これで普通の変数と同じように利用可能

構造体の使い方

- Student_info　record;
  - Sutudent_infoという型（構造体）のrecordとい名前の変数
  - 学生一人分のデータ
- record.ID = “s0001”;
  - recordのIDというメンバ変数に”s0001”を代入
- std::cout << record.Final;
  - recordのFinalというメンバ変数の値を出力

学生全体のデータ

- std::vector<Student_info> students;
- 学生数が何人かは事前にはわからない
  - だからvectorを使う
- vectorの各要素が何を示すかというと
  - Student_info
- 使い方の例
  - students[0].ID
  - students.push_back(record);

関数の利用

- 処理のまとまりを関数（サブルーチン）にする
- 例えば，
  - read: データの読み込み
  - median: メジアンの探索
  - grade: 総合得点の計算
- 構造体と関数で，プログラムを見通しよくする（「構造化」）

read 関数の設計

- 概要
  - 学生1人分のデータを標準入力ストリーム(cin)から読み取り，Student_info 型の変数に保存する
- 関数の汎用性（他のプログラムでも使えるように）を考えて
  - データの入力はcin専用ではなく，他の入力ストリーム（例えばファイル入力）も使えるように，関数の引数にする

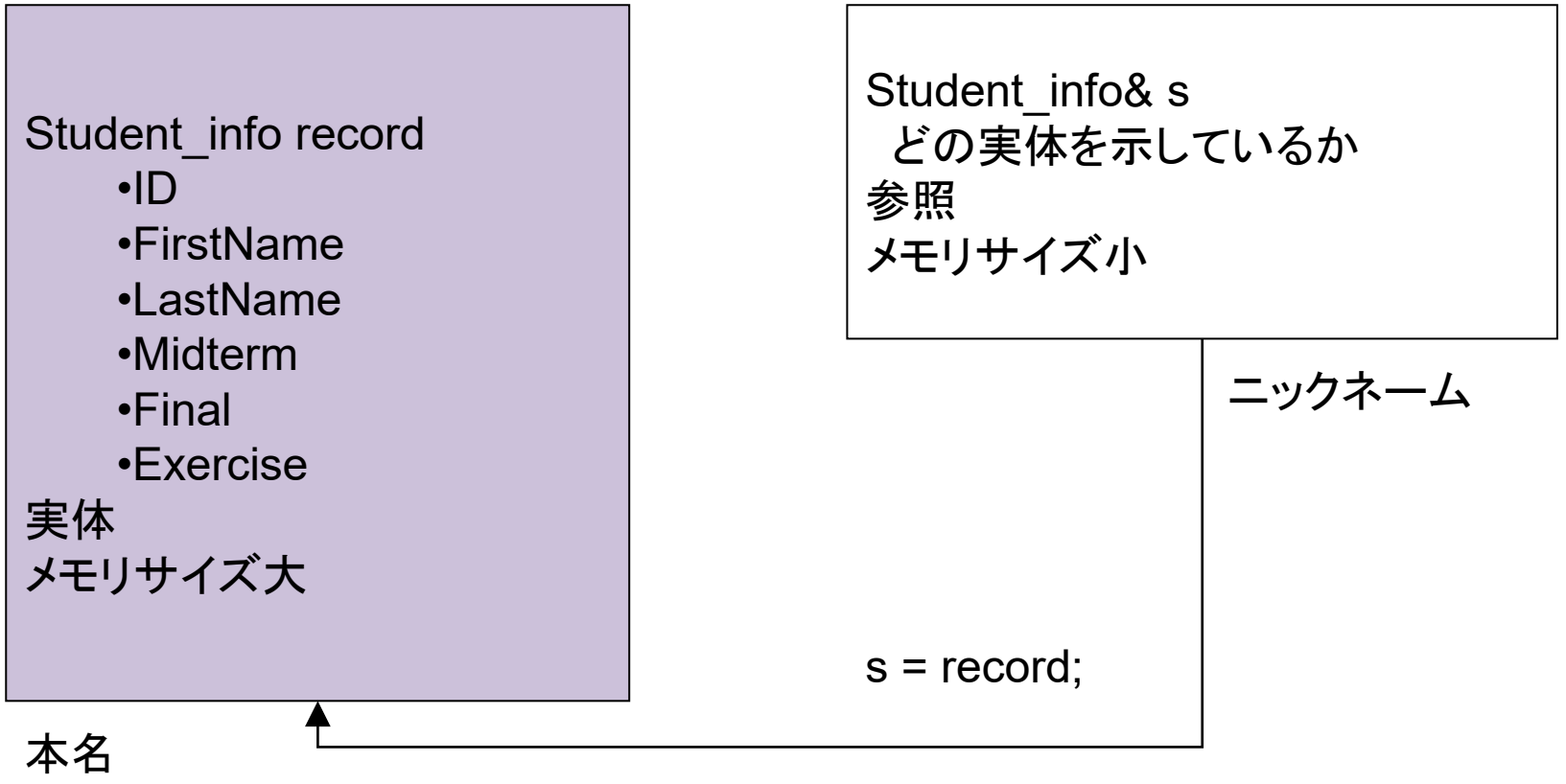
read 関数の設計

- std::istream& read(std::istream& is, Student_info& s)
- 引数
  - std::istream& is: 入力ストリーム(istream)の参照(&)型で，変数名がis
  - Student_info& record: Student_info型の参照(&)で，変数名がs
- 戻り値
  - std::istream&: 入力ストリーム，while文やif文で評価ができるように

参照（リファレンス）

- 参照
  - 別のところに実体がある変数を，他の名前で指し示す（「参照」する）
  - 使う時は，元の型に「&」をつける
- Student_info record;
  - 通常の変数，実体
- Student_info& s;
  - 参照，必ず実体を指し示す必要がある

参照（リファレンス）


Student_info record
•ID
•FirstName
•LastName
•Midterm
•Final
•Exercise
実体
メモリサイズ大
Student_info& s
どの実体を示しているか
参照
メモリサイズ小
ニックネーム
s = record;
本名

参照（リファレンス）

- 参照の使い方
- s = record;
  - これで s は record を指し示すことになる
  - s.ID = “s0002”; とすると record の内容が変更
  - record.ID = “s0003”; とすると s.ID も変化する
  - なぜなら s と record は同じものを表しているから

参照（リファレンス）

- 注意
  - 参照型の変数を宣言して，実体を示さずに使うことはできない
- 間違った例
  - Student_info& s; s.ID = “s0003”;
  - コンパイルエラー
  - これだと，sは実体を表していない
  - sは他の変数を参照しなければ使えない
  - 本名（実体）とニックネーム（参照）の関係

read 関数の使い方（ループの例）

```
int main()  {
   vector<Student_info> students;  //  学生全体のデータ
   Student_info record;  //  学生一人分のデータ
   //  標準入力ストリームからデータの読み込み
   //  読み込んだデータはrecordという変数に一時的に格納
   while( read( cin, record ) )   {
        //  recordをvectorに追加
        students.push_back( record );
   }
   //  読み込みデータがなくなるとループが終了
   //  read 関数の戻り値が偽： falseになるから
   //  データがないときにcinを評価すると false
}
```

read 関数の内容

```
std::istream& read( std::istream& is, Student_info& s )
{
    is >> s.ID >> s.GivenName >> s.SurName;
    is >> s.Midterm >> s.Final;
    //  演習の得点の読み込み，別の関数にする
    read_hw( is, s. Exercise );
    return is;
}
```

- 演習の得点の読み込みは，まとまった処理なので，別関数で行うことにする
- 引数のisを戻り値にすることで，関数を使いやすくしている

read_hw 関数の設計

- 概要
  - 学生1人の演習の成績（回数不明）を入力ストリームから読み取り，vector<double>&型の変数に保存する
- 今回のデータの形式
  - s10000001 Taro Aizu 80 20 100 100 100
  - s10000002 Jiro Aizu 100 90 90
  - 演習の最後に -1がない

データの読み込み方法

- データの末尾に-1がないのに，どうやって各学生のデータの終わりを見つけるか？
  - 各学生のデータの先頭は学生番号，string型（文字列）
  - 各学生のデータの末尾は期末試験か演習の成績，double型（数値）
  - 演習の成績（double型）を読んでるつもりで，次の学生の学生番号（string型）を読み込むとエラー
- このエラーを利用

read_hw 関数の内容

```
std::istream&  read_hw(std:: istream& is, std::
   vector<double>& v)
{
   if( is )   {
      double x;  //  毎回の演習の成績
      v.clear();
      while( is >> x ){
          v.push_back( x );
      }
      is.clear();
    }
    return is;
}
```

read_hw 関数の解説

- std::istream& read_hw(std::istream& is, std::vector<double>& v)
  - 第1引数: 入力ストリームの参照
    - この関数の中で内容を更新するので参照型
  - 第2引数: 演習の成績保存するvectorの参照
    - この関数の中で内容を更新するので参照型
  - 戻り値: 入力ストリーム
    - 他の関数から使いやすくするように

read_hw 関数の解説

- if(is) {}
  - もし入力ストリームが空とかエラー状態だったら何もしない
- v.clear();
  - 初期化，前の学生の処理でvectorにデータが残っているかもしれない
- while(is >> x) { v.push_back(x); }
  - データをdouble型のxに読み込み，vectorに追加
  - 読み込みが失敗したときに，ループから脱出
  - ループが終わるのは，
    - 入力がなくなったときか
    - double型に数値以外のデータを代入しようとしたとき

read_hw 関数の解説

- is.clear();
  - 入力ストリームのエラーを解除
  - とくにdouble型に数値以外を代入のケース
- return is;
  - 使いやすいように関数の戻り値に入力ストリーム

median 関数の設計

- 概要
  - 演習の成績(std::vector<double>)を与えて，メジアンの値を返す
- double median(std::vector<double> v)
  - 引数は，参照型ではないことに注意
  - メジアンの計算では演習の成績をソートする必要
  - ここでは，元のデータの並びを変更しないようにして，元のデータのコピーを関数に渡す
  - 引数に「&」がないことに注意

関数の引数と参照

- 参照をよく使うのは，関数の引数
  - 参照型：関数の中で値を変更し，それを関数の外でも反映させたいときに使う
  - 通常：関数の中で値を変更しても，関数の外では元のままであって欲しいときに使う
  - 引数がコピーされ，コピーが関数に渡される

median 関数の中身

```
double median(std::vector<double> v)
{
   std::vector<double>::size_type size = v.size(), mid;
   if( size == 0 )
       throw domain_error("要素数が0のメジアン");
   std::sort( v.begin(), v.end() );
   mid = size / 2;
   if( size % 2 == 0 )
       return (v[ mid ] + v[ mid - 1 ]) / 2;
   else
       return v[ mid ];
}
```

例外処理

- 演習の回数が0だったとき
  - メジアンや平均点が計算できない
  - 暫定的にメジアンや平均点を0とできるが,
  - できれば例外として扱いたい
- C++における例外処理（詳しくは，また後で）
  - #include <stdexcept>が必要
  - throw: 例外が起こったときに,「例外を投げる」
  - try: 例外を見つける範囲を決める
  - catch: 投げられた例外を捕まえたときに，どう処理するか

grade 関数の設計

- 概要
  - 中間試験，期末試験，演習の成績を与えて，演習メジアン，演習平均，演習合計，総合得点を計算する
- double grade(double midterm, double final, const std::vector<double>& hw, double& ex_med, double& ex_avg, double& ex_sum)
  - constは，変数を定数として扱い，値を変更しない
  - const std::vector<double>& hw は，hwのデータを見るだけで，データは変更しない

grade 関数の設計

- 関数の内部で値を変更したくないときに，便利
- double& ex_med などは値を書き換え，関数の外側でもそれを使うので，参照型

grade 関数の中身

```
double grade(double midterm, double final, const std::vector<double>& hw,
    double& ex_med, double& ex_avg, double& ex_sum)
{
    if( hw.size() == 0)
        std::throw std::domain_error(“演習回数が0”);
    ex_med = median( hw );
    ex_sum = 0;
    for( std::vector<double>::size_type i = 0; i != hw.size(); ++i){
        ex_sum += hw[i];
    }
    ex_avg = ex_sum / hw.size();
    return 0.2 * midterm + 0.4 * final + 0.4 * ex_med;
}
```

学生を名前順にソートして出力

- GivenName順にソート
  - もしGivenNameが同じだったら，次はSurNameでソート
- std::vector<Students_info> students　をソートする
  - 演習のソート，vector<double>のときは，大小関係は明らか
  - しかし，Student_info型の大小は決められていない
  - ソートのときにStudent_info用の大小関係を指定

vector<Students_info>のソート

- std::vector<Students_info> students
- std::sort(students.begin(), students.end(), compare);
  - students.begin() から students.end() までをcompareという関数に記述される大小関係に従ってソート
  - bool compare (const 型名& x, const 型名& y)
  - x が y より小さいときに真(true)を返す関数

compare 関数の内容

```
bool compare( const Student_info& x, const Student_info&
   y )
{
   //   xと y の GivenName が異なるときは GivenName を比較
   if( x. GivenName != y. GivenName )
        return( x. GivenName < y. GivenName );
   //  xと y の GivenName が同じときは SurName を比較
   else
        return( x. SurName < y. SurName );
}
```

main 関数

```
int main()
{
    std::vector<Student_info>  students;
    Student_info  record;
    //  まずデータの読み込み
    while( read( cin, record ) )          {
            students.push_back( record );
    }
    //  学生データを名前順にソート
    std::sort( students.begin(), students.end(), compare);
```

main 関数

```
// 学生のデータを出力
for( std::vector<Student_info>::size_type i = 0; i !=
        students.size(); ++i)          {
        // 総合得点，演習メジアン，演習平均，演習合計
        double total, ex_med, ex_avg, ex_sum;
        // 計算
        // grade関数の中で例外が発生するかもしれないので
        try         {
            total = grade( students[ i ].Midterm,
            students[ i ].Final, students[ i ].Exercise,
            ex_med, ex_avg, ex_sum);
            // 画面にデータを出力（ここでは省略）
        }
```

main 関数

```
        catch(std:: domain_error)  {
            //  ここに例外時の処理を書く
            //  今回はID, FirstName, LastNameの後に,
            //  演習回数が0だから総合得点が計算できない
            //  というエラーメッセージを出力
        }
    }
```