# C から入る C++

担当: 金丸隆志

第 7 回

## 1 はじめに

今回から C++ を用いたオブジェクト指向プログラミングの演習を行う.

C++ は膨大な機能を持った言語であり, その使い方は人によって様々である. 例えば, 前回まで C++ 固有の機能として「参照引数」,「for 文内での int i の定義」,「STL」,「iostream (cin や cout)」などを用いてきたが, これらの C++ の便利な機能のみを使って C 言語的なプログラミングを行う (C++ をベター C として用いる) ことも可能である. 個人的な意見を言えば, 前回までの知識しかなくても卒論などに必要なプログラミングはできると考えている.

しかし, 本演習では C++ をオブジェクト指向プログラム言語 (OOPL: Object-Oriented Programing Language) としてとらえ, その技術を習得する事を目指している. なぜなら大規模なアプリケーションの作成にはオブジェクト指向的な考え方が必要不可欠であると考えられるからである [1]. (もちろん, 演習ではそれほど大きなプログラムは作れないが, 演習で得るであろう知識の延長線上に大規模ソフトウェア開発がある, ということ)

なお, C++ の OOPL としての側面を身につければ, Java や C# などの言語の習得も容易になるであろう.

## 2 クラスとは何か

### 2.1 クラスとは型である

C 言語におけるプログラミングでは関数を中心に考えることが多かった. つまり, 行わせたいタスクからいくつかの機能を抽出し, それを関数として実装していく, というプログラミング手法である.

それに対し, オブジェクト指向プログラミングでは「**クラス**」を中心にしたプログラミングを行うことになる.

クラスを理解するために, まず「クラスを作るとは新たな型を定義することである [2]」という立場で考えてみよう. 型とは, char, int, float, double などのことであり, これに新たな型を自分で追加する, ということである.

ここでは簡単な例として複素数 c=x + $i$ y ($i$ は虚数単位) を表す myComplex クラスを作成してみよう. ただし, x (実部: real part), y (虚部: imaginary part) は double 型で実装するものとし, 複素数のノルム $|c| = \sqrt{x^2 + y^2}$ を求める機能をつけるものとする.

ここで, myComplex クラスの実装例を図 1 のリストに示す.

myComplex クラスの宣言部 (インターフェイス部) は myComplex.h (ヘッダファイル), myComplex クラスの実現部は myComplex.cpp, 実際に myComplex クラスを呼び出して利用する部分は complex_test.cpp のようにファイルが分割されていることに注意しよう.

このようにファイルを分割してプログラムを作成することは C++ では標準的に行われるので, 今後も採用することにする [3].

図 1 のリストにおけるポイントを以下に列挙する.

**[(a) myComplex.h]**

- 「public:」は,「以下のデータメンバやメンバ関数がどこからでも参照できる」ことを意味するアクセス指定子である.
- 実部 x と虚部 y は myComplex クラスの**データメンバ** (または**メンバ変数**) として, double 型で実装されている.
- ノルムを求める関数「double norm(void);」を**メンバ関数** として定義している.

**(a) myComplex.h**

```
class myComplex{
    public:
        /* データメンバ */
        double x; /* real part */
        double y; /* imaginary part */
        /* メンバ関数 */
        double norm(void);
};
```

**(b) myComplex.cpp**

```
#include <math.h> /* for sqrt */
#include "myComplex.h"

double myComplex::norm(void){
    return(sqrt(x*x + y*y));
}
```

**(c) complex_test.cpp**

```
#include <iostream>
#include <math.h> /* for sqrt */
#include "myComplex.h"
using namespace std;

int main(void){
    myComplex c;
    myComplex* cp;

    c.x = 3;
    c.y = 4;

    cp = new myComplex; /* ヒープ領域に確保 */
    cp->x = 0.5;
    cp->y = sqrt(3)/2;

    cout << "norm of c is " << c.norm() << endl;
    cout << "norm of cp is " << cp->norm() <<
endl;

    delete cp; /* 確保した領域を開放 */
    return 0;
}
```

図 1: myComplex クラス (第 1 版) とその利用.

**[(b) myComplex.cpp]**

- メンバ関数の定義の多くは cpp ファイルで行う.
- メンバ関数とは, クラスのデータメンバに対する操作を行う関数である.
- メンバ関数の定義には「:: (スコープ解決演算子)」を用いて, その関数が myComplex のメンバ関数であることを明示する.
- メンバ関数の内部では, クラスのデータメンバ (ここでは x, y) にアクセスできる.
- sqrt (square root) は sqrt(x) = $\sqrt{x}$ を求める関数である.

**[(c) complex_test.cpp]**

- myComplex 型の変数 c および myComplex 型のポインタ cp を定義している.
- myComplex 型の変数 c については, データメンバおよびメンバ関数に対して「. (ドット演算子)」でアクセスする.
- myComplex 型のポインタ cp については, データメンバおよびメンバ関数に対して「-> (アロー演算子)」でアクセスする.

なお, complex_test.cpp において, myComplex 型の変数としてスタック領域に変数 c, ヒープ領域に *cp が確保されたが, これらのことを「myComplex 型の**オブジェクト** (または**インスタンス**)」と呼ぶ.

## 2.2 クラスとは物である

前節では複素数クラスを取り扱い,「クラスを作成することは新しい型を作ることである」と見なせることをみた. 複素数は数学的な概念であるから, 型と見なすことに違和感はないであろう.

しかし, クラスはより広い概念であり, もっと一般的なものを取り扱える. 例えば, 文献 [2] は「名詞で表されるものはすべてクラスの候補となり得る」という考えに基づいて書かれている.

わざとらしい例としては「テレビ」クラスなどが考えられる. テレビクラスのデータメンバとしては, テレビの状態を表す「電源の状態」,「現在のチャンネル」,「現在の音量」などが候補になるし, テレ

ビクラスのメンバ関数としては「電源 ON/OFF」,「チャンネル変更」,「音量調節」などが考えられる.

より研究らしい例を出すと,「遺伝的アルゴリズム (GA: Genetic Algorithm)」クラスが考えられる. 遺伝的アルゴリズムとは,「ある系の状態を染色体 (0/1 からなる遺伝子の列) で表現し, 染色体集団の確率的な操作 (交叉, 突然変異, etc) により系の最適な状態を求める」アルゴリズムである.

まず GA クラスを実現するには, 0/1 の列からなる「染色体 (chromosome)」クラスが必要である. 染色体クラスは int 型の配列 (実際にはポインタ) をデータメンバとしてもち,「配列のランダム初期化」などのメンバ関数を持てば良い. これを元にすれば, GA クラスは染色体クラスの配列 (実際にはポインタ) をデータメンバとしてもち,「交叉」,「突然変異」などの機能をメンバ関数として持つものとして実装できる.

GA クラスの例を見るとわかるように, オブジェクト指向プログラミングにおいては,「クラスの記述」がプログラミングの大部分を占めることになる. (C 言語によるプログラミングでは, 関数の記述がメインであった).

そして, main 関数は作成したクラスを呼び出して利用したり, あるいは作成したクラスにバグがないかどうか検証するために用いられることが多くなるだろう.

[**課題 1**]
身近なもので, クラスの候補になるものを一つ挙げよ. その際, 何がデータメンバ, メンバ関数として考えられるかも書くこと. なお, プログラム化できるかどうかは考えても考えなくてよい. (文中の「テレビ」クラスのようなものでも良い, ということ)

## 3 データ隠蔽 (data hiding)

前章での例ではクラスのデータメンバに対して「c.x」や「cp->x」などとアクセスしていた. これは C 言語の構造体で用いられた方法である.

しかし, オブジェクト指向プログラミングでは, クラスのデータメンバに直接アクセスするのを避け, データメンバへのアクセスにメンバ関数を経由させることが多い. これを**データ隠蔽** (data hiding), あるいは**情報隠蔽** (information hiding) と呼ぶ.

データ隠蔽を行う理由を簡潔に説明するのは難しいが, 一つには「データメンバに不用意にアクセスさせないことでバグを減らす」ためである.

**(a) myComplex.h**

```
class myComplex{
    private:
        double x; /* real part */
        double y; /* imaginary part */

    public:
        void setReal(double xx);
        void setImag(double yy);

        double getReal(void);
        double getImag(void);

        double norm(void);
};
```

**(b) myComplex.cpp**

```
#include <math.h> /* for sqrt */
#include "myComplex.h"

void myComplex::setReal(double xx){
    x = xx;
}
void myComplex::setImag(double yy){
    y = yy;
}
double myComplex::getReal(void){
    return x;
}
double myComplex::getImag(void){
    return y;
}
double myComplex::norm(void){
    return(sqrt(x*x + y*y));
}
```

図 2: データ隠蔽にもとづいた myComplex クラス (第 2 版)

この考えのもとには「クラスの作成者 (前節の例では myComplex を書く人) とクラスの利用者 (complex_test.cpp を書く人) が異なる」, という前提がある. このような状況は, 大規模なアプリケーショ

ンのプログラミングでは日常的に起こっていることである.

また, 商用コンパイラ付属のクラスライブラリを用いてみると, クラスのデータメンバに直接アクセスすることはほとんどないことに気づくであろう. これは, クラスライブラリがデータ隠蔽の考えに基づいて設計されているからである.

さて, 実際にデータ隠蔽を行うには, 図 2 のリストにあるように, データメンバをアクセス指定子「private:」のもとで定義すればよい. これにより, そのデータメンバはそのクラス内からしかアクセスできなくなる.

それだけではデータメンバ x, y の値を変更することも, 参照することもできなくなるので, 新たに値変更用のメンバ関数 (setReal(), setImag()) および値参照用のメンバ関数 (getReal(), getImag()) を定義している.

**complex_test.cpp の内容**

```
#include <iostream>
#include <math.h> /* for sqrt */
#include "myComplex.h"
using namespace std;

int main(void){
    myComplex c;
    myComplex* cp;

    c.setReal(3);
    c.setImag(4);

    cp = new myComplex;
    cp->setReal(0.5);
    cp->setImag( sqrt(3)/2 );

    cout << "norm of c is " << c.norm() << endl;
    cout << "norm of cp is " << cp->norm() << endl;

    delete cp;
    return 0;
}
```

図 3: myComplex クラス (第 2, 3 版) の利用例.

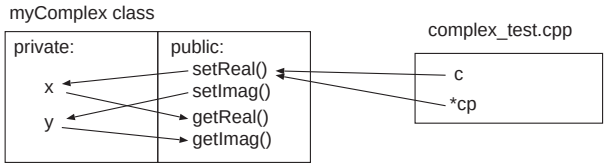


図 4: データ隠蔽の模式図.

この myComplex クラス (第 2 版) の利用例を図 3 のリストに示した.

この時の myComplex クラスの利用状況の模式図を図 4 に示す. データメンバに直接アクセスするのではなく, メンバ関数を介していることがわかるであろう.

## 4 インライン関数

前章の myComplex クラス (第 2 版) では, 値の変更/参照用の関数を myComplex.cpp 内に記述していた (図 2).

実は, 関数の呼び出しには負荷 (関数呼び出しのオーバーヘッドという) がかかるため, 値の変更/参照程度の簡単な機能を関数化するのはあまり好ましいことではない.

このように, 「簡単な機能のみを行う関数」を効率を下げることなく実現するために C++ では「**インライン関数**」という仕組みを用意している. インライン関数の詳細は C++ の教本 (文献 [5] など) を参照して欲しい.

インライン関数の実現にはいくつかの方法があるが, 本演習では図 5 のように「関数をヘッダファイルのクラス宣言内部に記述する」という方法をとる. 以後の演習ではこの記法を主に使用することになるので, しっかり理解しておいてほしい.

関数をクラス内部で定義すると, コンパイラがその関数をインライン関数として実現しようと試みる. うまくいかない場合 (ある程度複雑な処理をする関数の場合) はその関数は通常の関数として実現されるだろう.

**(a) myComplex.h**

```
class myComplex{
    private:
        double x; /* real part */
        double y; /* imaginary part */

    public:
        void setReal(double xx){ x = xx; }
        void setImag(double yy){ y = yy; }

        double getReal(void){ return x; }
        double getImag(void){ return y; }

        double norm(void);
};
```

**(b) myComplex.cpp**

```
#include <math.h> /* for sqrt */
#include "myComplex.h"

double myComplex::norm(void){
    return(sqrt(x*x + y*y));
}
```

図 5: データ隠蔽にもとづき, さらにインライン関数を用いた myComplex クラス (第 3 版)

[**課題 2**]

(1) 図 1 のリストを書き写して実行し, 意味を理解せよ.

(2) myComplex.h, myComplex.cpp を図 5 のものに変更せよ. ただし, complex_test.cpp は図 1 (c) のリストのものをそのまま用いるとする. コンパイルをするとどのようなエラーメッセージが表示されるか. そのエラーは何故出たのだろうか.

(3) complex_test.cpp を 図 3 のものに変更し, 実行してみよ (図 3 および 図 5 のリストを実行することに相当). これはデータ隠蔽にもとづく myComplex クラスを実現し, 利用したことになる.

(4) (前問のソースを用いる) complex_test.cpp を編集し, "c=3+4i", "cp=0.5+0.86i" などと表示させる命令を追加せよ. ("0.86" の桁数は違ってもよい) (ヒント: getReal(), getImag() を用いる)

## 参考文献


[1] オブジェクト指向開発を行うことによるメリットを知りたければ, 例えば文献 [2] が参考になる.

[2] Tucker!, "憂鬱なプログラマのためのオブジェクト指向開発講座," 翔泳社 (1998).

[3] もちろん, C 言語でも規模が大きくなればファイル分割は普通に行われる. その際, ヘッダファイルには関数のプロトタイプや構造体を, 実体のファイルには関数本体を記述し, main 関数はさらに別のファイルに記述することが多い.

[4] Visual C++ 付属の MFC (Microsoft Foundation Classes) や Borland C++ Builder 付属の VCL (Visual Component Library) など. クイックソートのところで触れた C++ 標準ライブラリにも便利なクラスが多く含まれる.

[5] 柴田望洋, "C プログラマのための C++ 入門," ソフトバンク (1992).