# C から入る C++

担当: 金丸隆志

第 8 回

## 1 はじめに

前回からクラスを用いたプログラミングの基礎を学んでいる. 今回は「**コンストラクタ**」,「**デフォルトコンストラクタ**」,「関数や演算子の**多重定義**(**オーバーロード**)」を取り扱い, 次回は「**デストラクタ**」,「**コピーコンストラクタ**」などを扱う予定である.

覚えることばかりが多くてつまらないかもしれないが, いずれも重要な内容なので理解するようにして欲しい.

## 2 コンストラクタ

前回, myComplex クラスを作成し, complex_test.cpp ファイルの main 関数から呼び出した.

その際, 図 1 のリストの様に変数 c を定義し, real part の値と imaginary part の値を setReal()/setImag() 関数でセットしてからノルムを計算していた.

**complex_test.cpp 内部**

```
...
myComplex c; /* c の宣言 */

c.setReal(3); /* real part をセット */
c.setImag(4); /* imaginary part をセット */

cout << "norm of c is " << c.norm() << endl;
...
```

図 1: myComplex クラスの呼び出し.

ここで, setReal()/setImag() 関数による初期化を行わなければ c のデータメンバ x, y の値は不定であるから, c.norm() はどんな値を返すかわからない. このような初期化忘れを防ぐために, **コンストラクタ**という仕組みがある.

前回の myComplex.h, myComplex.cpp, complex_test.cpp を用いて例を続けよう。コンストラクタは特別なメンバ関数である. 通常のメンバ関数のように myComplex.cpp に記述しても良いのだが, 今回は値の代入を行うだけなので, myComplex.h のクラス宣言内部に記述し, インライン関数として実現してみよう. ただし, 2 つの数 xx と yy を引数にとり, コンストラクタ内部でデータメンバ x, y に xx と yy の値を代入することにする.

コンストラクタの定義と利用の例を図 2 のリストに示した. ソースの一部しか記述していないので注意して欲しい.

**(a) myComplex.h 内部**

```
...
public:
  myComplex(double xx, double yy){
    x = xx; y = yy;
  } /* コンストラクタ */
...
```

**(b) complex_test.cpp 内部**

```
...
myComplex c(3,4);
myComplex* cp;

cp = new myComplex(0.5, sqrt(3)/2);
...
```

図 2: myComplex クラスのコンストラクタ (第一版) とその利用.

コンストラクタの宣言とその利用について, ポイントになるのは以下の通り.

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

- コンストラクタは「**戻り値を持たず**」,「**関数名**

**がクラス名と等しい**」特別なメンバ関数である.

- コンストラクタの引数は自由に決めてよいが, ここでは x と y を初期化したいので, 2 つの引数 xx と yy を取ることにした.

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

- 変数 c の宣言の際, c(3,4) と宣言することでコンストラクタが呼ばれ, c の real part は 3, imaginary part は 4 と初期化される.

- ポインタ cp の場合, new で領域を確保する際に new myComplex(···) のように引数を与えることで, cp のデータメンバが初期化される.

このように myComplex 型の変数宣言の際, 必ず値を要求することで初期化忘れを防ぐことができる.

しかし, このままでは図 3 のリストのような変数宣言ができなくなってしまう. 試しに, 図 2 の

**complex_test.cpp 内部**

```
...
myComplex c;
...
```

図 3: myComplex 型の変数 c の宣言.

コンストラクタが定義されている時, 図 3 の宣言を含むソースを BCC でコンパイルすると, 「myComplex::myComplex() に一致するものが見つからない」というエラーがでて, コンパイルに失敗してしまう.

「myComplex c;」という宣言の際, コンストラクタを呼びだそうとしているのだが, 定義されているコンストラクタは引数を 2 つ取るもののみであるので, 「引数をとらないコンストラクタは存在しない」という意味で上記のエラーが出てしまうのである.

引数なしでのコンストラクタ呼び出しを実現するには, 次章の「デフォルトコンストラクタ」の知識が必要になる.

## 3 デフォルトコンストラクタ

前章の最後に触れたように, 「myComplex c;」と宣言した時にも, 「myComplex c(3,4);」と宣言したときにもそれぞれ適切なコンストラクタが呼び出されるようにしたい.

「myComplex c;」の形式の宣言ができるためには, 「引数なしで起動できるコンストラクタ」が必要である. このようなコンストラクタを「**デフォルトコンストラクタ**」という. 本章では, デフォルトコンストラクタの記法を 2 つ学ぶ.

まず, 一つ目の記法を図 4 のリストに記す. この

**myComplex.h 内部**

```
...
public:
  myComplex(void){
    x = 0; y = 0;
  } /* デフォルトコンストラクタ */
  myComplex(double xx, double yy){
    x = xx; y = yy;
  } /* コンストラクタ */
...
```

図 4: myComplex クラスのコンストラクタ (第二版).

リストのポイントは以下の通り.

- myComplex(void) と myComplex(double, double) の二つのコンストラクタが定義されている.

- 「myComplex c;」という宣言では myComplex(void) が, 「myComplex c(3,4);」という宣言では myComplex(double, double) が呼び出される.

- 「myComplex c;」と宣言されたとき, c の real part と imaginary part はともに 0 で初期化される.

このように, C++ では一つの関数名 (ここではコンストラクタ myComplex) に対し, 複数の定義を許し, 関数呼び出し時に適切な関数が呼び出されるような仕組みになっている. これは**多重定義** (**オーバーロード**, overloading) と呼ばれる.

デフォルトコンストラクタの 2 つめの記法を図 5 のリストに示す. このリストのポイントは以下の通り.

- 第一の記法とは異なり, コンストラクタの定義は 1 つのみである.

**myComplex.h 内部**

```
    ...
  public:
    myComplex(double xx=0, double yy=0){
        x = xx; y = yy;
    } /* コンストラクタ */
    ...
```

図 5: myComplex クラスのコンストラクタ (第三版). デフォルト引数を用いている.

- コンストラクタの引数に,「**デフォルト引数**」を与え, 引数が与えられなかったときに関数に渡される値を定義している.
- 「myComplex c;」と宣言した時は「xx=yy=0」としてコンストラクタが呼ばれ,「myComplex c(3,4);」と宣言した時は「xx=3, yy=4」としてコンストラクタが呼ばれる.

デフォルト引数により, 引数が与えられない時にも適切なコンストラクタが呼び出されるようになっている.

さらに,「myComplex c(5);」と宣言したときは「xx=5, yy=0」としてコンストラクタが呼ばれることに注意しておく.

図 5 のコンストラクタを定義した時に可能な myComplex 型の宣言方法を図 6 に示しておく.

**complex_test.cpp 内部**

```
    ...
  myComplex c1;          /* c1=0+0i */
  myComplex c2(5);       /* c2=5+0i */
  myComplex c3(3,4);     /* c3=3+4i */

  myComplex *cp1, *cp3, *cp3;
  cp1 = new myComplex;       /* *cp1=0+0i */
  cp2 = new myComplex(5);    /* *cp2=5+0i */
  cp2 = new myComplex(3,4); /* *cp3=3+4i */
    ...
```

図 6: 可能な myComplex 型の変数の宣言.

なお, デフォルト引数はコンストラクタだけではなく, 通常の関数に対しても使用できることに注意しておく.

最後に, 間違ったコンストラクタの定義の例を一つ示す.

**myComplex.h 内部**

```
    ...
  public:
    myComplex(void){
        x = 0; y = 0;
    }
    myComplex(double xx=0, double yy=0){
        x = xx; y = yy;
    }
    ...
```

図 7: myComplex クラスのコンストラクタ. 間違った定義の例.

図 7 のリストは, 2 つのコンストラクタを定義している. しかし,「myComplex c;」と宣言したとき,「myComplex(void)」と「myComplex(double, double)」のどちらを呼び出せばよいか曖昧であるので, コンパイル時にエラーになる.

本演習では, 図 4 の方法と図 5 の方法を適宜使い分けることにする.

## 4 演算子の多重定義

前章において, コンストラクタの多重定義を取り扱った. それに関連し, 本章では「演算子の多重定義」を考える.

演算子の代表的なものに, 四則演算をあらわす「+」,「−」,「∗」,「/」がある. ここでは + 演算子を考えよう.

いま,「1 + 5」は「(int 型) + (int 型)」の演算であり,「1.0 + 5.0」は「(double 型) + (double 型)」の演算であることに注意しよう [1]. すると, + 演算子は int 型に対する加算機能と double 型に対する加算機能を持っていることになり, これは + 演算子が複数の定義をもつ, すなわち多重定義されていると解釈できることを意味する.

同様に, + 演算子を今まで扱って来た myComplex 型に対しても多重定義することができる. すなわち, $c_1+c_2 = (x_1+y_1i)+(x_2+y_2i) = (x_1+x_2)+(y_1+y_2)i$ なる演算を myComplex 型に対する「+」演算として定義する, ということである.

「+」演算子を定義するには, myComplex クラスの内部で「myComplex operator+(const myComplex& c)」なるメンバ関数を定義すればよい.

これは, myComplex の参照 x を (定数) 引数 としてとり, myComplex 型の戻り型を持つメンバ関数である. 少し難しく感じるかも知れないが, わからなくなったら教科書などで確認するとして, ここではまずソースを提示して動かしてみることを重視しよう.

myComplex クラスに対して + 演算子を多重定義する方法を図 8 のリストに示した. これもインライン関数として myComplex.h の内部で定義した.

**myComplex.h 内部**

```
...
public:
   ...
  myComplex operator+(const myComplex& c){
        return( myComplex(x+c.x, y+c.y) );
  }
```

図 8: myComplex クラスにおける + 演算子の多重定義.

図 8 のリストのポイントは以下の通り.

- myComplex 型の変数 c1, c2, c3 に対し「c3 = c1 + c2」という演算を行うと 「c3 = c1.operator+(c2);」が実行される [2].
- operator+ 関数の内部ではコンストラクタを適切な引数 (加算の答え) を与えて起動し, その戻り値を return する.
- operator+ 関数は myComplex 型のメンバ関数であるから, myComplex 型の private 変数に直接アクセスできる (c.x, c.y のこと).

以上, myComplex 型における「+」演算子の多重定義を見た.

四則演算の演算子以外にも, 多くの演算子を多重定義することができる. さらに, 同じ「+」演算子の多重定義にも幾つかの方法があり, ここで取り扱ったのはそのうちの一つである. 興味のある人は C++ の教本 (文献 [3] など) で調べてみるとよいだろう.

[**課題**]

(1) 第七回資料の図 3, 図 5 のリストから出発しよう. myComplex.h に対し今回の資料の図 5 のコンストラクタを追加せよ. また, complex_test.cpp の main 関数内で図 6 の宣言をいくつか試し, コンストラクタの働きを理解せよ.

(2) さらに, myComplex.h に対して図 8 のメンバ関数を追加せよ. さらに, complex_test.cpp の main 関数に,「+」演算子の働きを確認する命令を追加せよ. (ヒント：c3 = c1 + c2 を実行し, その値を表示させるようにすれば良い)

(3) さらに, myComplex.h 内で「−」(減算演算子),「∗」(乗算演算子) の多重定義を行い, それらの働きを確認する命令を complex_test.cpp の main 関数内部に記述せよ. ただし, 減算と乗算は複素数の演算として矛盾のない演算にすること [4].

## 参考文献


[1] もちろん,「1.0 + 5.0」は「(float 型) + (float 型)」であるかもしれないが, ここでは便宜上 double 型として記述した.

[2] 今まで説明しなかったが, クラスのインスタンス間での代入は常に可能である. すなわち, myComplex 型の変数 c1, c2 があり,「c1 = c2」が実行されると, c2 の全てのデータメンバの値が c1 のデータメンバに代入される. だから,「c3 = c1 + c2」のような代入も可能になる.

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

[4] もちろん「/」(除算演算子) も多重定義できるが, 除算の場合, 0 で割ってしまう時の例外処理が必要なため, やや繁雑になるので今回は課題には含めないことにした.