# プログラム演習 (8)C++の「クラス」を学ぶ

C++のようなオブジェクト指向言語では、**クラス**というものを定義して、その実体である**インスタンス**を用いて種々の処理を行います。クラスは、データ（**メンバ変数**）と（データを用いた）計算処理（**メソッド**）で構成されたものであり、抽象的な概念に対応します。また、クラスに実体を持たせたものがインスタンスに対応します。「男の子」がクラスだとしたら、「木田拓実」「柳原健人」がそれぞれ「男の子クラス」のインスタンスになるわけです。クラスは、例えば、プログラマ自身が定義できる、int とか float といったデータ型式の一種と考えてもよいかもしれません。

それでは早速簡単なプログラムを書いてみましょう。とりあえず以下のプログラムを打ち込み、コンパイルして実行してみて下さい。きっと何も起こりません。

Listing 1: プログラム例 1

```
1  #include<string>
2  #include<vector>
3
4  using namespace std;
5
6  class human{
7   private:
8   public:
9    human(){};
10   ~human(){};
11 };
12
13 int main()
14 {
15   human boku;
16 };
```

6 行目から 11 行目において、クラス human を定義しています。通常、メンバ変数とメソッドがクラス内で定義されますが、ここでは何も定義していません。private、public とありますが、これらは**アクセス属性**と呼ばれ、データやメソッドへのアクセス権限を指定するためのものです。private 以下で定義されるのは外部からアクセスが出来ないメンバ変数、メソッドであり、public 以下で定義されるのは外部からアクセス可能なメンバ変数、メソッドになります。9 行目は**コンストラクタ**と呼ばれるものであり、human クラスのインスタンスが生成されたとき、自動的にここで定義された処理が行われます。ただし、このプログラムでは処理内容が空（から）{} になっているので、human クラスのインスタンスが生成されても何の処理も行われません。一方、10 行目は**デストラクタ**と呼ばれるものであり、このクラスのインスタンスがメモリから解放されるとき（使用済みになったとき）に、自動的にここで定義された処理が行われます。

13 行目から 16 行目からが実際の処理に該当します（メインプログラム）。ここでは human クラスのインスタンス boku を生成して処理が終わっています。インスタンスの生成のみを行っているので、画面上には何も表示されないわけです。

それでは次に、human クラスに年齢のデータを実装してみましょう。プログラムは以下のように変更されるでしょう。

Listing 2: プログラム例 2

```
1  #include<string>
2  #include<vector>
3
4  using namespace std;
5
6  class human{
7   private:
8    int age;
9   public:
10   human(){age=0;};
11   ~human(){};
12 };
13
14 int main()
15 {
16   human boku;
17 };
```

human クラスの private にメンバ変数 age が追加されています。これは整数型の変数なので、それを示す int が変数名の頭に付されています。また、コンストラクタに age=0 という処理が加えられています。これは、human クラスのインスタンスが生成された場合、そのインスタンスのメンバ変数 age はゼロに設定されることを意味します。ですので、今回のメインプログラムでは、生成されたインスタンス boku の年齢はゼロとなっています。

それでは、boku の年齢を出力させてみましょう。

Listing 3: プログラム例 3

```
1  #include<string>
2  #include<vector>
3  #include<iostream>
4
5  using namespace std;
6
7  class human{
8   private:
9    int age;
10  public:
11   human(){age=0;};
12   ~human(){};
13   int GetAge(){return age;};
14 };
15
16 int main()
17 {
18   human boku;
19   cout<<"# Age of boku : "<<boku.GetAge()<<"\n";
20 };
```

3 行目が追加されていますが、これは画面上に出力するための命令 cout を使うためのものです。C++では一般にメンバ変数は private として定義されます。これは、クラスの外側からのメンバ変数へのアクセスを制限するためです（外部から自分の「内部情報」をいろいろいじられるのは誰でも嫌なものでしょう）。そのかわりに**アクセス関数**なるものを用意してあげます。それが 13 行目の GetAge() に対応します。このメソッドを使うことにより、int 型変数のデータとして age の値を返してあげるわけです[1]。19 行目では実際に、boku の age をアクセス関数を介して入手し、それを画面上に出力させています。cout は画面上に出力させるための命令であり、使い方はこの例から類推して下さい。

human クラスのインスタンスが必ず 0 歳であるといろいろと不都合なので、年齢を変更してあげられるようにしましょう。

Listing 4: プログラム例 4

```
1  #include<string>
2  #include<vector>
3  #include<iostream>
4
5  using namespace std;
6
7  class human{
8   private:
9    int age;
10  public:
11   human(){age=0;};
12   ~human(){};
13   int GetAge(){return age;};
14   void PutAge(int inp){age=inp;};
15 };
16
17 int main()
18 {
19   human boku;
20   boku.PutAge(20);
21   cout<<"# Age of boku : "<<boku.GetAge()<<"\n";
22 };
```

この例では 14 行目が追加されており、このアクセス関数により、メンバ変数 age に具体的な値を代入することができます。なお、PutAge(int inp) というように、括弧内に変数の型と変数名が示されていますが、これを**引数**とよびます。引数は、そのメソッドを呼び出すときに、メソッド（そのメソッドを有するクラスのインスタンス、と言ったほうが適切かもしれません）の「外部から」引き渡されるデータです。また、void というのはこのメソッドでは何のデータも返さないことを意味します[2]。20 行目で boku を 20 歳にしているので、画面上には成人となった boku の年齢が表示されるでしょう。

さらに、「誕生日を迎える」というメソッドを human クラスに追加してあげましょう。

Listing 5: プログラム例 5

```
1  #include<string>
2  #include<vector>
3  #include<iostream>
4
5  using namespace std;
```

[1] int 型変数を返すことは、13 行目において GetAge() の前に int が付されていることから分かります。なお、返される値は return で指定してあげます。

[2] 英語の「void」は、「真空」「空（から）」という意味を持ちます。

```
6
7  class human{
8   private:
9    int age;
10  public:
11   human(){age=0;};
12   ~human(){};
13   int GetAge(){return age;};
14   void PutAge(int inp){age=inp;};
15   void BirthDay(){age++;};
16 };
17
18 int main()
19 {
20   human boku;
21   boku.PutAge(20);
22   cout<<"# Age of boku : "<<boku.GetAge()<<"\n";
23   boku.BirthDay();
24   cout<<"# Age of boku : "<<boku.GetAge()<<"\n";
25 };
```

15 行目で定義されている BirthDay メソッドがそれに対応します。age++は、整数型の変数 age の値をひとつ増やす操作に対応しています。ひとつ減らす場合には age−− と書けばよいですし、同じ増やすにしても、age=age+1 とか age+=1 などとも書くことができます。メインプログラムも変更していますが、まあ、解説は不要でしょう。

今度は human クラスに「性別」を区別するようにしてあげます。

Listing 6: プログラム例 6

```
1  #include<string>
2  #include<vector>
3  #include<iostream>
4
5  using namespace std;
6
7  class human{
8   private:
9    int age;
10   bool man;
11  public:
12   human(){age=0; man=true;};
13   ~human(){};
14   int GetAge(){return age;};
15   void PutAge(int inp){age=inp;};
16   void BirthDay(){age++;};
17   void MakeWoman(){man=false;};
18   void MakeMan(){man=true;};
19 };
20
21 int main()
22 {
23   human boku;
24   boku.PutAge(20);
25   boku.MakeMan();
26
27   human watashi;
28   watashi.PutAge(20);
29   watashi.MakeWoman();
30
31   cout<<"# Age of boku     : "<<boku.GetAge()<<"\n";
32   cout<<"# Age of watashi : "<<boku.GetAge()<<"\n";
33 };
```

性別は 10 行目の bool man で定義するようにしています。これは**ブーリアン型変数**と呼び、true（真）か false（偽）の値をとります。この例ではコンストラクタで man=true とされていますから、human クラスのインスタンスが生成された場合は自動的に男性になります。ただし、性別を指定し直すメソッドを 17、18 行目に実装していますので、インスタンスを作ったあとにこれらを使って性別を指定し直してあげればよいでしょう。

さらに human クラスに飲酒させるメソッドを追加しましょう。通常は 20 歳未満は飲酒禁止ですが、ここでは飲酒禁止を男性は 19 歳未満、女性は 21 歳未満としましょう。

Listing 7: プログラム例 7

```
1  #include<string>
2  #include<vector>
3  #include<iostream>
4
5  using namespace std;
6
7  class human{
8   private:
9    int age;
10   bool man;
```

```
11  public:
12    human(){age=0; man=true;};
13    ~human(){};
14    int GetAge(){return age;};
15    void PutAge(int inp){age=inp;};
16    void BirthDay(){age++;};
17    void MakeWoman(){man=false;};
18    void MakeMan(){man=true;};
19    void Drinking(){
20      bool forbidden=false;
21      if(man&&age<19)forbidden=true;
22      if(!man&&age<21)forbidden=true;
23      if(forbidden){
24        cout<<"# Drinking is forbidden!\n";
25        cout<<"#   Age       : "<<age<<"\n";
26        cout<<"#   Man/Woman : "<<man<<"\n";
27      };
28    };
29  };
30
31  int main()
32  {
33    human boku;
34    boku.PutAge(20);
35    boku.MakeMan();
36
37    human watashi;
38    watashi.PutAge(20);
39    watashi.MakeWoman();
40
41    cout<<"# Age of boku    : "<<boku.GetAge()<<"\n";
42    cout<<"# Age of watashi : "<<boku.GetAge()<<"\n";
43
44    boku.Drinking();
45    watashi.Drinking();
46  };
```

ここでは飲酒をするためのメソッド Drinking を human クラスに追加しています。Drinking メソッドではまず、飲酒が禁止されるべきかを boolean 型の変数 forbidden で定義します。デフォルトは false、すなわち飲酒は禁止ではないことになります。21 行目では条件分岐の if 構文が使われています。man&&age<19 とありますが、まず&&は「and」に対応しますので、man が true、age<19 も true の場合にそれ以下の処理が行われます[3]。つまりここでは、boolean 型の変数 man が true であり年齢が 19 未満の場合に、forbidden を true にしています。また、22 行目には!man がありますが、これは man が false である場合に true になることに対応します。ですから 22 行目は、boolean 型の変数 man が false であり年齢が 21 未満の場合に、forbidden が true になります。それらの処理の後、forbidden が true の場合には、飲酒が禁止されている旨が画面上に表示されます。ここでは、boolean 型の変数 man の中身が表示されますが、true の場合には 1 が、false の場合には 0 が表示されます。なお、「age が 20 と等しい」というような場合には age==20 となり、「age が 20 と異なる」というような場合には age!=20 というような書き方となります。

さて、あとはお約束でしょうから、以下のようなプログラムを書きました。適当に解読して下さい。なお、4 行目が追加されているので注意して下さい。

Listing 8: プログラム例 8

```
1   #include<string>
2   #include<vector>
3   #include<iostream>
4   #include<cstdlib>
5
6   using namespace std;
7
8   class human{
9    private:
10    int age;
11    bool man;
12   public:
13    human(){age=0; man=true;};
14    ~human(){};
15    int GetAge(){return age;};
16    void PutAge(int inp){age=inp;};
17    void BirthDay(){age++;};
18    void MakeWoman(){man=false;};
19    void MakeMan(){man=true;};
20    bool Man(){return man;};
21    human FallInLove(human &other){
22      if((man&&other.Man())||(!man&&!other.Man())){
23        cout<<"# No baby.\n";
24        exit(0);
25      };
26      human baby;
27      return baby;
28    };
```

[3]ちなみに、「or」に対応するものは ||（縦線（キーボード右上）二本）になります。

```
29 };
30
31 int main()
32 {
33   human boku;
34   boku.PutAge(20);
35   boku.MakeMan();
36
37   human watashi;
38   watashi.PutAge(20);
39   watashi.MakeWoman();
40
41   cout<<"# Age of boku     : "<<boku.GetAge()<<"\n";
42   cout<<"# Age of watashi  : "<<boku.GetAge()<<"\n";
43
44   human baby=boku.FallInLove(watashi);
45   cout<<"# Age of baby     : "<<baby.GetAge()<<"\n";
46   cout<<"# Age of watashi  : "<<baby.GetAge()<<"\n";
47 };
```

21 行目の FallInLove メソッドですが、引数として human &other としています。この&ですが、引数で指定したデータの「実体を」渡していることを意味します。これについて少し簡単な説明を行います。

この&に関して、C++にはいろいろな機能がありますが、そのうちの重要なものの一つが**参照**と呼ばれるものです。次の関数を見て下さい。

Listing 9: C++の参照を理解するための例 1

```
1 void reference()
2 {
3   int x;
4   int &y=x;
5 };
```

この関数では変数 x を生成し、y を x と同一としています。もし y に対して何か操作を行ったとすると、この場合は x もそれにあわせて変化することになります。これは「&」を用いることによって、y が x の「別名」のような扱いとなったからです。従って、この例で示した関数では、実体としてはただひとつの変数 x が定義されていることになります。

これに対して、以下の例を見て下さい。

Listing 10: C++の「参照」を理解するための例 2

```
1 void compare()
2 {
3   int x;
4   int y=x;
5 };
```

この関数では 2 つの変数 x と y が生成されており、int y=x によって x のコピーが y として生成されています。一般的にはこの方法で問題ないですが、仮にコピー元のデータのサイズが極端に大きい場合などは、それと同一のインスタンスが生成されることになるため、不要な場合は避けたほうがよいと言えます。

FallInLove メソッドに戻りますが、メインプログラムである 44 行目において、インスタンス watashi が引数として指定されていますが、この場合はインスタンス自身が FallInLove メソッドに渡されることになります。ですから、仮に FallInLove メソッド内で、other に処理が加えられた場合（例えば MakeMan で性別を変えるとか）、FallInLove メソッドにおける other は watashi そのものなので、メソッド終了時には watashi は男性になってしまいます。一方、21 行目の FallInLove メソッドの引数に&が付されていなかった場合には、引数として渡された watashi が、FallInLove メソッド内で other にコピーされます。つまり、watashi と全く同一のデータが other として作成されることを意味します。従って、FallInLove メソッドで other に処理が加えられても、それは watashi のコピーであり watashi 自身ではありませんので、watashi は何も変わりません。メソッドに対してサイズの大きいインスタンスを引数として渡す場合には、&をつけないと、そのコピーがメソッド内で複製されることになるので、メモリがそれだけ必要になることになります[4]。また、データのコピーに時間を要することもあります。

問題：C++を用いて原子炉工学研究室を計算機上で再現して下さい。「学生」クラスを定義し、その複数のインスタンスをメインプログラムで生成し、各々の成長を表現して下さい。研究ゼミで「知力」が上昇すること、飲み会で「体力」が低下すること、教員の協力により「論文」を作成することを、計算機上で再現して下さい。なお、知力、体力は数値化し、論文の作成は、ある条件の下で達成されるようにして下さい。

[4]例えば 1GB のデータが引数となっている場合には、同一容量のデータが複製されるわけなので、2GB の容量が必要となることになります。

以降は必要に迫ったときに実施して下さい。

この演習では、クラス human とメインプログラムをひとつのファイルにまとめて記述していますが、ある程度大掛かりなプログラムを作るときには、種々のクラスを定義するプログラムと、計算するときに実際に走らせるメインプログラムを別々にしておくと便利です。そのためのサンプルとして package.tgz なるものがあります。Dropbox の CBZ_source ディレクトリに置いていますので、それをダウンロードし、解凍して下さい。Package ディレクトリが作成されると思いますので、まずはそのディレクトリに移動して下さい。そこに@ReadMe ファイルがありますので、その指示に従って操作を行うとよいでしょう。

この Package ディレクトリについて簡単に説明します。まずは src ディレクトリですが、ここに human クラスのソースファイルが格納されています。この src ディレクトリにおいて make を行うことにより、human クラスのソースファイルがコンパイルされ、それが定義されたライブラリファイル libRPG.a が生成されます。その後、Package ディレクトリに戻り、main.cxx ファイルを make によりコンパイルします。この make では src ディレクトリにある libRPG.a をインクルードしてコンパイルしますので、main.cxx ファイルでは libRPG.a 内で定義されている human クラスを使用することが出来ます。