4 何をどうクラスとして定義すべきか

4–1 クラス設計の基本方針

何をクラスとして定義すべきか：

まず注意すべきことは、

- オブジェクトはC言語構造体に操作方法も加えたもの。
- オブジェクトの中心にあるのは、
  (何らかのモノを表すために) 集められたデータ群

⇩

プログラム作りの課題があった時、

一般的には、まず、

. . . . . .

① 課題文中の名詞 もしくは
関連した物／エージェント
} の中から 適切なもの を選び、

補足： ここでの「 適切なもの(名詞) 」としては、
◇ 汎用性のあるオブジェクトに繋がりそうなもの、
◇ オブジェクト化するとプログラム全体が互いに独立な部分に綺麗に分割されそうなもの、など

② その名詞 もしくは
関連物／エージェント
} に相当するモノ1個を表すデータ群を定める。

そして、

③ 前ステップ②で定めたデータ群を操作するための関数群を定め、

④ 前ステップ②~③で定めたデータや関数のそれぞれについて
公開／非公開を定める、

というクラス定義の作業を繰り返し て、
問題領域に固有の抽象データ型をクラス定義の形で必要なだけ実現

クラスを設計／定義する際の注意：

- 適切なクラス名、メンバ名を付ける。
  - ◇ クラス名としては
    前段落ステップ① で選んだ「名詞やその関連物／エージェントに相当するモノ」を意味する英単語(列,名詞句)を採用すべき。
  - ◇ データメンバの名前としては その要素の役割に応じた名詞(句)
  - ◇ 関数の名前としては 行う操作を意味する動詞(句)
- クラス内に用意したメンバは可能な限り非公開とする。

  その恩恵：
  非公開メンバについては、
  内部の実装方法を自由に変更できる。
- 将来の利用を想定して、有用そうな操作は関数として用意しておく。

(補足) 名前の構成に関して、

この講義ノート

... Java言語の習慣に倣って、
2つ目以降の単語の頭文字を大文字にして複数の単語を並べる。

例えば、`StackChar`, `numOfInstances`, `showContents`, ...

C++言語の通常

... Stroustrupの書籍に倣って、
単語と単語の間に”_”記号を入れて複数の単語を並べる。

例えば、`Stack_char`, `num_of_instances`, `show_contents`, ...

4–2 クラス設計の例

4–2–1 平面上の点のクラス

例4. 1 (平面上の点のクラス)

平面上の点を扱う場合 は、

「平面上の点」を抽象的な1つのデータとして扱う

ことにより見通しの良いプログラムができると期待できる。

⇨ 平面上の点のクラス Point2D を定義

```
[motoki@x205a]$  cat -n Point2D.h
     1 /* 「平面上の点」オブジェクトのクラス Point2D (仕様部) */
     2
     3 #ifndef ___Class_Point2D
     4 #define ___Class_Point2D
     5
     6 #include <string>
```

```
 7
 8 class Point2D {
 9   double x;
10   double y;
11 public:
12   Point2D(double x = 0.0, double y = 0.0)
                                        : x(x), y(y) {}
13   //Point2D(const Point2D& pt)  //コピーコンストラクタ
14   //  : x(pt.x), y(pt.y) {}     // (heap領域からの
                                   // 領域確保がないので、
15   //                            // 同等のものがデフォ
                                   // ルトで用意される。)
16   double getX() const { return x; }
17   double getY() const { return y; }
18   std::string toString() const; //内部保持の点座標を
                                   // stringデータとして返す
19   void setXY(double x = 0.0, double y = 0.0);
```

```
                                                    //点の座標を設定
    20    void plus(const Point2D& pt);             //点の座標を移動
    21    void plus(const double offsetX = 0.0,
                                    const double offsetY = 0.0);
    22 };
    23
    24 #endif
[motoki@x205a]$  cat -n Point2D.cpp
     1 /* 「平面上の点」オブジェクトのクラス Point2D (実装部) */
     2
     3 #include <sstream>
     4 #include <string>
     5 #include "Point2D.h"
     6 using namespace std;
     7
     8 // Point2D型オブジェクトを操作するための関数群 ----------
```

```
 9 // オブジェクト内部に保持している点の座標を
                                    // string型データとして返す
10 string Point2D::toString() const
11 {
12   ostringstream os;
13   os << "(" << x << "," << y << ")";
14   return os.str();
15 }
16
17 // オブジェクト内部に保持している点の座標を設定
18 void Point2D::setXY(double x, double y)
19 {
20   this->x = x;
21   this->y = y;
22 }
23
```

出力用の文字列ストリームのクラス

os内の文字列をstringとして取り出し

```
24 // オブジェクト内部に保持している点の座標を移動
25 void Point2D::plus(const Point2D& pt)
26 {
27   x += pt.x;
28   y += pt.y;
29 }
30
31 void Point2D::plus(const double offsetX,
                       const double offsetY)
32 {
33   x += offsetX;
34   y += offsetY;
35 }
[motoki@x205a]$
```

これに関して、

- 上のクラス Point2D が定義されていれば 、
  次の様なプログラムを書くこともできる。

```
[motoki@x205a]$  cat -n usePoint2D.cpp
     1 /* Point2D.h, Point2D.cpp の利用例 */
     2
     3 #include <iostream>
     4 #include <string>
     5 #include "Point2D.h"
     6 using namespace std;
     7
     8 inline double square(const double x) { return x*x; }
     9
    10 int main()
    11 {
    12   Point2D p1(0.5), p2(p1), q[11];
    13
    14   p2.plus(1,2);
    15   for (int i=0; i<=10; ++i) {
    16     double x = static_cast<double>(i) / 10.0;
```

コピーコンストラクタ呼出し

↓

↑

デフォルトコンストラクタ呼出し

```
    17      q[i].setXY(x, square(x));
    18    }
    19
    20    cout << "p1=" << p1.toString() << endl;
    21    cout << "p2=" << p2.toString() << endl;
    22    for (int i=0; i<=10; ++i) {
    23      cout << "q[" << i << "]=" << q[i].toString()
                                                << endl;
    24    }
    25 }
[motoki@x205a]$  g++ usePoint2D.cpp Point2D.cpp
[motoki@x205a]$  ./a.out
p1=(0.5,0)
p2=(1.5,2)
q[0]=(0,0)
q[1]=(0.1,0.01)
q[2]=(0.2,0.04)
```

```
q[3]=(0.3,0.09)
q[4]=(0.4,0.16)
q[5]=(0.5,0.25)
q[6]=(0.6,0.36)
q[7]=(0.7,0.49)
q[8]=(0.8,0.64)
q[9]=(0.9,0.81)
q[10]=(1,1)
[motoki@x205a]$
```

4–2–2 長方形のクラス

例題4. 2 (長方形のクラス)　標準入力から3つの長方形のデータ (横幅と高さ) を読み取り、その中から最大面積の長方形の情報を出力するC++プログラムを作成せよ。

(考え方)

> 1個の長方形の横幅と高さを1箇所に集めて
> 操作方法と共に抽象的な1つのデータとして扱う

ことにより、見通しが良く間違いが少ないプログラムを期待できる。

⇨ 1個の長方形を表すオブジェクトの型枠となるクラスRectangleを定義。

⇨ 1個の長方形を表すオブジェクトの型枠となるクラスRectangle
を定義。その際、

- ソフトウェア部品としては汎用性のあるものが望ましい
  ので、

  ◇ 個々のRectangleインスタンスには固有のid番号を保持させる。

  ◇ id番号の一意性を保証するために最新のid番号
  (＝過去に生成したRectangleインスタンスの個数)
  を常に保持する領域をRectangleメンバとして用意。

- 十分に情報隠蔽も考慮すべきである
  ので、

  ◇ 個々のデータメンバは外部から直接アクセスできない様にする。

  ◇ 外部からの正当な利用ができる様に
  参照や値変更の関数メンバを用意。

(Rectangleクラスの定義)　Rectangleクラスの定義例

```
[motoki@x205a]$  cat -n Rectangle.h
     1 /* 長方形オブジェクトのクラス Rectangle (仕様部) */
     2
     3 #ifndef ___Class_Rectangle
     4 #define ___Class_Rectangle
     5
     6 #include <string>
     7
     8 class Rectangle {
     9   static int numOfInstances;
                          // これまでに生成したインスタンスの個数
    10   const int id;      // 長方形インスタンスに付けるid番号
    11   double width;      // 長方形の横幅
    12   double height;     // 長方形の高さ
```

```
13 public:
14   Rectangle(double width = 0.0, double height = 0.0)
15     : id(numOfInstances++), width(width),
                                        height(height) {}
16   Rectangle(const Rectangle& rectangle);
                                        //コピーコンストラクタ
17   //~Rectangle() {}
18   // オブジェクト(もしくはRectangleクラス全体)の情報を
                                        // 提供するための関数群
19   static int getNumOfInstances()
                                    { return numOfInstances; }
20                                  //これまでに生成したRectangle
                                        //インスタンスの個数を返す
21   double getId() const { return id; }
22   double getWidth() const { return width; }
23   double getHeight() const { return height; }
```

```
    24    void setWidth(double width) { this->width = width; }
    25    void setHeight(double height)
                                        { this->height = height; }
    26    double getArea() const { return width*height; }
                                                  //長方形の面積を返す
    27    std::string toString() const;
                          //内部の長方形情報をstringデータとして返す
    28 };
    29
    30 #endif
[motoki@x205a]$  cat -n Rectangle.cpp
     1 /* 長方形オブジェクトのクラス Rectangle (実装部) */
     2
     3 #include <sstream>
     4 #include <string>
     5 #include "Rectangle.h"
     6 using namespace std;
```

```
 7
 8 // static変数の初期化 ----------------------------------
 9 int Rectangle::numOfInstances = 0;
10
11 // 各種コンストラクタ ------------------------------------
12 Rectangle::Rectangle(const Rectangle& rectangle)
                                          // コピーコンストラクタ
13           : id(numOfInstances++),
14             width(rectangle.width),
                                  height(rectangle.height) {}
15
16 // Rectangle型オブジェクトを操作するための関数群 ---------
17 // オブジェクト内部に保持している長方形情報を
                                        // string型データとして返す
18 string Rectangle::toString() const
19 {
20   ostringstream os;
```

```
    21    os << "rectangle(id=" << id
    22       << ",width=" << width << ",height=" << height
                                                  << ")";
    23    return os.str();
    24 }
[motoki@x205a]$
```

(Rectangleクラスの利用)

Rectangleクラスが上の様に定義されていれば、
それを利用して例題4.2で要求されている作業を例えば次の様に...

```
[motoki@x205a]$  cat -n findMaxAmong3Rectangles.cpp
     1 /* Rectangle.h, Rectangle.cpp の利用例
     2 /* 次の作業を順に行うC++プログラム
     3 /* (1)標準入力から得られたデータを基に
                                  Rectangleインスタンスを3つ生成 */
     4 /* (2)面積最大のインスタンスを見つけてその情報を出力
     5
```

```
 6 #include <iostream>
 7 #include <string>
 8 #include "Rectangle.h"
 9 using namespace std;
10
11 int main()
12 {
13   Rectangle* rectangle[3];
14
15   //標準入力からデータを入力、
                          //それを基に長方形インスタンスを生成
16   for (int i=0; i<3; ++i) {
17     double width, height;
18     cout << "長方形の幅と高さを入力：";
19     cin >> width >>height;
20     rectangle[i] = new Rectangle(width, height);
21   }
```

オブジェクトをheap領域上に構成するので

↓

Rectangleオブジェクトをheap領域上に構成

```
22
23    //生成された長方形インスタンスの内部情報を出力
24    cout << "生成された長方形インスタンス：" << endl;
25    for (int i=0; i<3; ++i) {
26      cout << "*rectangle[" << i << "]="
                    << rectangle[i]->toString() << endl;
27    }
28
29    //面積最大の長方形インスタンスを見つけてその情報を出力
30    int indexOfMaxArea = 0;
31    double maxArea     = rectangle[0]->getArea();
32    for (int i=1; i<3; ++i) {
33      double area = rectangle[i]->getArea();
34      if (area > maxArea) {
35        indexOfMaxArea = i;
36        maxArea        = area;
37      }
```

```
38    }
39    cout << "==>面積最大のものは"
40         << rectangle[indexOfMaxArea]->toString()
                                              << endl;
41
42    //heap領域から確保したメモリを開放
43    for (int i=0; i<3; ++i)
44      delete rectangle[i];
45 }
[motoki@x205a]$  g++ findMaxAmong3Rectangles.cpp
                                              Rectangle.cpp
[motoki@x205a]$  ./a.out
長方形の幅と高さを入力： 3  17
長方形の幅と高さを入力： 10 10
長方形の幅と高さを入力： 15  5
生成された長方形インスタンス：
*rectangle[0]=rectangle(id=0,width=3,height=17)
```

```
*rectangle[1]=rectangle(id=1,width=10,height=10)
*rectangle[2]=rectangle(id=2,width=15,height=5)
==>面積最大のものはrectangle(id=1,width=10,height=10)
[motoki@x205a]$
```

4–2–3 文字列のクラス

例4. 3 (文字列のクラス)

C++言語 ... 文字列を扱うためにstringクラス。

ここでは 、このstringクラスの様に、

> 内部の配列容量を意識することなく
> 任意長の文字列を表すための枠組み

を自前で定義してみる。例えば次の通り。(クラス名String)

```
[motoki@x205a]$  cat -n String.h
     1 /* 文字列オブジェクトのクラス String (仕様部) */
     2
     3 #ifndef ___Class_String
     4 #define ___Class_String
     5
     6 #include <iostream>
     7
```

```
 8 class String {
 9   char* s;
10   int length;
11 public:
12   String();
13   String(const char* stringWithNullTerminal);
                                          //変換コンストラクタ
14   String(const String& string);  //コピーコンストラクタ

15   ~String() { delete[] s; }
16   // 文字列操作の関数群
17   char charAt(int index) const;
                                  //指定された添字番号の文字要素
18   char& operator[](int index) const;   演算子[ ]を用意
       ↑                          str[index]という書き方を可能に
     代入文の左辺にも置ける様に
```

```
19    void print() const { std::cout << "\"" << s << "\"";
                                                  //内部の文字列
20                                                //  情報を出力
21    void println() const { std::cout << "\"" << s
                                        << "\"" << std::endl; }
22    void assign(const char* stringWithNullTerminal);
                                                        //代入
23    void assign(const String& string);
24    String& append(const char* stringWithNullTerminal);
                                                  //最後尾に引数
25                                               //  文字列を連接
26    String& append(const String& string);
27    String& shrinkToSubstring(int from, int to);
                                                //部分文字列に縮小
28        //(引数のプラス値はindex,マイナス値は削減幅と解釈)
29 };
```

```
    30
    31 #endif
[motoki@x205a]$  cat -n String.cpp
     1 /* 文字列オブジェクトのクラス String (実装部) */
     2
     3 #include <iostream>
     4 #include <cstring>
     5 #include <cassert>
     6 #include "String.h"
     7 using namespace std;
     8
     9 // 各種コンストラクタ ---------------------------------
    10 String::String(): length(0)
    11 {
    12   s = new char[1];
    13   assert(s != 0);
    14   s[0] = '\0';
```

指定条件を満たさないと強制終了

```
15 }
16
17 String::String(const char* stringWithNullTerminal)
18 {
19   length = strlen(stringWithNullTerminal);
20   s = new char[length+1];
21   assert(s != 0);
22   strcpy(s, stringWithNullTerminal);
23 }
24
25 String::String(const String& string)
                                    : length(string.length)
26 {
27   s = new char[length+1];
28   assert(s != 0);
29   strcpy(s, string.s);
30 }
```

```
31
32 // 文字列操作の関数群 ----------------------------------
33 // 指定された添字番号の文字
34 char String::charAt(int index) const
35 {                                  指定された添字番号の妥当性をチェック
36   assert(0 <= index && index < length);
37   return s[index];
38 }
39       演算子[ ]を用意してstr[index]という書き方を可能に
40 char& String::operator[](int index) const
41 {   ↑                                代入文の左辺にも置ける様に
42   assert(0 <= index && index < length);
43   return s[index];
44 }
45
```

```
46 // 代入操作
47 void String::assign(
                          const char* stringWithNullTerminal)
48 {
49   delete[] s;
50   length = strlen(stringWithNullTerminal);
51   s = new char[length+1];
52   assert(s != 0);
53   strcpy(s, stringWithNullTerminal);
54 }
55
56 void String::assign(const String& string)
57 {
58   if (this == &string)  // assignment of the form a=a;
59     return;
60   delete[] s;
```

```
61    length = string.length;
62    s = new char[length+1];
63    assert(s != 0);
64    strcpy(s, string.s);
65 }
66
67 // 最後尾に引数文字列を連接
68 String& String::append(
                          const char* stringWithNullTerminal)
69 {
70    int initLength = length;
71    length += strlen(stringWithNullTerminal);
72    char* temp = new char[length+1];
73    assert(temp != 0);
74    strcpy(temp, s);
75    strcpy(temp+initLength, stringWithNullTerminal);
76    delete[] s;
```

```
77    s = temp;
78    return *this;
79 }
80
81 String& String::append(const String& string)
82 {
83    int initLength = length;
84    length += string.length;
85    char* temp = new char[length+1];
86    assert(temp != 0);
87    strcpy(temp, s);
88    strcpy(temp+initLength, string.s);
89    delete[] s;
90    s = temp;
91    return *this;
92 }
93
```

```
 94 // 部分文字列に縮小
        // (引数のプラス値はindex,マイナス値は削減幅と解釈)
 95 String& String::shrinkToSubstring(int from, int to)
 96 {
 97   assert(from<length && to <length);
 98   if (from < 0)   //マイナス値-->先頭からの削減幅と解釈
 99     from = - from;
100   if (to < 0)   //マイナス値-->最後尾からの削減幅と解釈
101     to = length -1 + to;
102   if (from <= to)
103     length = to - from + 1;
104   else
105     length = 0;
106   for (int i=0; i<length; ++i) //
107     s[i] = s[from+i];
108   s[length] = '\0';
109   return *this;
```

```
   110 }
[motoki@x205a]$
```

これに関して、

- 上のクラスString が定義されていれば、
  それを使って次の様なプログラムを書くこともできる。

```
[motoki@x205a]$  cat -n useString.cpp
     1 /* String.h, String.cpp の利用例 */
     2
     3 #include <iostream>
     4 #include "String.h"
     5 using namespace std;
     6
     7 int main()
     8 {          ↓     C言語文字列を引数とするコンストラクタで
     9   String x("abc"), y, z;        スタック領域上に変数確保
    10                     ↑  ↑     デフォルトコンストラクタで
    11   y.assign("123_");
```

```
12    x[0] = 'A';
13    for (int i=0; i<5; ++i)
14      z.append(x).append(y);
15    cout << "z=";
16    z.println();
17    z.shrinkToSubstring(2, -5);
18    cout << "z.shrinkToSubstring(2, -5); " << endl
                                              << "==> z=";
19    z.println();
20  }
[motoki@x205a]$  g++ useString.cpp String.cpp
[motoki@x205a]$  ./a.out
z="Abc123_Abc123_Abc123_Abc123_Abc123_"
z.shrinkToSubstring(2, -5);
==> z="c123_Abc123_Abc123_Abc123_Ab"
[motoki@x205a]$
```

4–2–4 複素数のクラス

例題4. 4 (複素数のクラス)　非負整数データ $\mathbf{n}$ と複素数データ $\mathbf{C_n}$, $\mathbf{C_{n-1}}$, ..., $\mathbf{C_0}$ を読み込み、これらの定数値の下で複素多項式

$$\mathbf{C_n z^n + C_{n-1} z^{n-1} + C_{n-2} z^{n-2} + \cdots + C_1 z + C_0}$$

の値が

$$\mathbf{z = e^{i\pi k/5} = \cos\frac{\pi k}{5} + i\ \sin\frac{\pi k}{5}} \qquad \mathbf{(k=0,1,2,3,\ \ldots,\ 9)}$$

のそれぞれの値に対してどの様に変化するかを、表の形に見易く出力するC++プログラムを作成せよ。

(考え方)

プログラム内で複素数を double 型データと同じ様に扱う

ことが出来れば、

通常の実数値多項式を計算するコードとほぼ同じで済む予想。

⇨ 複素数を抽象的な 1 つのデータとして扱うためのクラス ComplexNum を定義。その際、

- 将来の利用のために、
  - ◇ 複素数間の四則演算子 +, -, *, / だけでなく
  - ◇ 等価判定演算子 ==, != や
  - ◇ 複合代入演算子 +=, -=, *=, /=

  も使える様にする。

- 複素数とdouble型の間の演算も可能にしたい。
  演算子の左側の要素がdouble型の場合の対応
  $\longrightarrow$ ComplexNumクラスの外で行わざるを得ない。

  ⇨ 一貫性を保つため、
  これらの(多重)定義は全てComplexNumクラスの定義の外で

(ComplexNumクラスの定義)　定義例を示す。

```
[motoki@x205a]$  cat -n ComplexNum.h
     1 /* 複素数のクラス ComplexNum, ver.4 (仕様部) */
     2
     3 #ifndef ___Class_ComplexNum
```

```
 4 #define ___Class_ComplexNum
 5
 6 #include <iostream>
 7 #include <string>
 8
 9 class ComplexNum {
10   double re;
11   double im;
12 public:
13   ComplexNum(double re = 0.0, double im = 0.0)
                                        : re(re), im(im) {}
14   //ComplexNum(const ComplexNum& x) //コピーコンスト
                                       //ラクタ (heap領域から
15   //  : re(x.re), im(x.im) {} // の領域確保がないので、
16   //                          // 同等のものがデフォルト
                                 // で用意される。)
17   double getRe() const { return re; }
```

```
18    double getIm() const { return im; }
19    void setRe(double re) { this->re = re; }
20    void setIm(double im) { this->im = im; }
21    std::string toString() const;
                          // 保持している複素数をstring型で返す
22    std::string toString(int precision) const;
23                                    // 複合代入演算(下で定義)
                    演算子を定義する関数, 演算子関数
24    ComplexNum& operator+=(const ComplexNum& y);
25    ComplexNum& operator-=(const ComplexNum& y);
26    ComplexNum& operator*=(const ComplexNum& y);
27    ComplexNum& operator/=(const ComplexNum& y);
28    ComplexNum& operator+=(double y);          ↑
29    ComplexNum& operator-=(double y);          ↑
30    ComplexNum& operator*=(double y);          ↑
31    ComplexNum& operator/=(double y);          ↑
                                    ↑     演算子の右側要素
```

```
32                                          // 四則演算(メンバ関数外)
```

右の関数に要素への参照を例外的に許可することを宣言

↓

フレンド関数

```
33    //friend ComplexNum operator+(ComplexNum x,
```

↑ ↑ ↑ ↑ ↑

```
                                          // const ComplexNum& y);
```

↑ ↑ ↑

演算子の左側要素

演算子の右側要素

右の関数の定義に内部メンバ参照は不要なので、コメントアウト

```
34    //friend ComplexNum operator-(ComplexNum x,
                                          // const ComplexNum& y);
35    //friend ComplexNum operator*(ComplexNum x,
                                          // const ComplexNum& y);
36    //friend ComplexNum operator/(ComplexNum x,
                                          // const ComplexNum& y);
37    //friend ComplexNum operator+(ComplexNum x,
                                                 // double y);
```

```
38    //friend ComplexNum operator-(ComplexNum x,
                                                  // double y);
39    //friend ComplexNum operator*(ComplexNum x,
                                                  // double y);
40    //friend ComplexNum operator/(ComplexNum x,
                                                  // double y);
41    friend ComplexNum operator+(double x,
                                  const ComplexNum& y);
42    friend ComplexNum operator-(double x,
                                  const ComplexNum& y);
43    friend ComplexNum operator*(double x,
                                  const ComplexNum& y);
44    friend ComplexNum operator/(double x,
                                  const ComplexNum& y);
45    friend ComplexNum operator-(const ComplexNum& y);
```

↑

右の関数に要素への参照を例外的に許可することを宣言　フレンド関数

```
46                                  // 等価判定演算(メンバ関数外)
47    friend bool operator==(const ComplexNum& x,
                             const ComplexNum& y);
48    friend bool operator!=(const ComplexNum& x,
                             const ComplexNum& y);
49    friend bool operator==(const ComplexNum& x,
                                       double y);
50    friend bool operator!=(const ComplexNum& x,
                                       double y);
51    friend bool operator==(double x,
                             const ComplexNum& y);
52    friend bool operator!=(double x,
                             const ComplexNum& y);
```

```
53                                  // 入出力ストリームとの演算 <<, >>
54    friend std::ostream& operator<<(std::ostream& out,
                                      const ComplexNum& y);
55    friend std::istream& operator>>(std::istream& in,
                                      ComplexNum& y);
56 };
57
58 // inline宣言する演算子の定義 ---------------------------
59 // (inline宣言するメンバ関数については
                                  // クラス定義と同じファイル内に入れる)
60
61 // 複合代入演算子 (ComplexNum += ComplexNum 等)
62 inline ComplexNum& ComplexNum::operator+=(
                                      const ComplexNum& y) {
63    re += y.re;
64    im += y.im;
```

```
65   return *this;
66 }
67
68 inline ComplexNum& ComplexNum::operator-=(
                                    const ComplexNum& y) {
69   re -= y.re;
70   im -= y.im;
71   return *this;
72 }
73
74 inline ComplexNum& ComplexNum::operator*=(
                                    const ComplexNum& y) {
75   double next_im = re*y.im+im*y.re;
76   re = re*y.re-im*y.im;
77   im = next_im;
78   return *this;
79 }
```

```
80
81 inline ComplexNum& ComplexNum::operator/=(
                                        const ComplexNum& y) {
82   double next_im = (im*y.re-re*y.im)
                                      /(y.re*y.re+y.im*y.im);
83   re = (re*y.re+im*y.im)/(y.re*y.re+y.im*y.im);
84   im = next_im;
85   return *this;
86 }
87
88 // 複合代入演算子 (ComplexNum += double 等)
89 inline ComplexNum& ComplexNum::operator+=(double y) {
90   re += y;
91   return *this;
92 }
93
94 inline ComplexNum& ComplexNum::operator-=(double  y) {
```

```
 95   re -= y;
 96   return *this;
 97 }
 98
 99 inline ComplexNum& ComplexNum::operator*=(double  y) {
100   re *= y;
101   im *= y;
102   return *this;
103 }
104
105 inline ComplexNum& ComplexNum::operator/=(double  y) {
106   re /= y;
107   im /= y;
108   return *this;
109 }
110
```

```
111 // 四則演算子 (メンバ関数外, ComplexNum + ComplexNum 等)
112 inline ComplexNum operator+(ComplexNum x,
                    const ComplexNum& y) { return x+=y; }
113 inline ComplexNum operator-(ComplexNum x,
                    const ComplexNum& y) { return x-=y; }
114 inline ComplexNum operator*(ComplexNum x,
                    const ComplexNum& y) { return x*=y; }
115 inline ComplexNum operator/(ComplexNum x,
                    const ComplexNum& y) { return x/=y; }
116
117 // 四則演算子 (メンバ関数外, ComplexNum + double 等)
118 inline ComplexNum operator+(ComplexNum x, double y) {
                                          return x+=y; }
119 inline ComplexNum operator-(ComplexNum x, double y) {
                                          return x-=y; }
120 inline ComplexNum operator*(ComplexNum x, double y) {
```

```
                                                    return x*=y; }
121 inline ComplexNum operator/(ComplexNum x, double y) {
                                                    return x/=y; }
122
123 // 四則演算子 (メンバ関数外, double + ComplexNum 等)
124 inline ComplexNum operator+(double x,
                                          const ComplexNum& y)
125                     { return ComplexNum(x+y.re, y.im); }
126 inline ComplexNum operator-(double x,
                                          const ComplexNum& y)
127                     { return ComplexNum(x-y.re, -y.im); }
128 inline ComplexNum operator*(double x,
                                          const ComplexNum& y)
129                     { return ComplexNum(x*y.re, x*y.im); }
130 inline ComplexNum operator/(double x,
                                          const ComplexNum& y)
131     { return ComplexNum(x*y.re/(y.re*y.re+y.im*y.im),
```

```
132                         -x*y.im/(y.re*y.re+y.im*y.im)); }
133
134 // 単項マイナス演算子 (メンバ関数外, - ComplexNum)
135 inline ComplexNum operator-(const ComplexNum& y)
136                     { return ComplexNum(-y.re, -y.im); }
137
138 // 等価演算子 (メンバ関数外, ComplexNum == ComplexNum 等)
139 inline bool operator==(const ComplexNum& x,
                           const ComplexNum& y)
140                   { return (x.re==y.re && x.im==y.im); }
141 inline bool operator!=(const ComplexNum& x,
                           const ComplexNum& y)
142                   { return (x.re!=y.re || x.im!=y.im); }
143
144 // 等価演算子 (メンバ関数外, ComplexNum == double 等)
145 inline bool operator==(const ComplexNum& x, double y)
146                     { return (x.re==y && x.im==0.0); }
```

```
147 inline bool operator!=(const ComplexNum& x, double y)
148                         { return (x.re!=y || x.im!=0.0); }
149
150 // 等価演算子 (メンバ関数外, double == ComplexNum 等)
151 inline bool operator==(double x, const ComplexNum& y)
152                         { return (x==y.re && 0.0==y.im); }
153 inline bool operator!=(double x, const ComplexNum& y)
154                         { return (x!=y.re || 0.0!=y.im); }
155
156 #endif
[motoki@x205a]$  cat -n ComplexNum.cpp
  1 /* 複素数のクラス ComplexNum, ver.4 (実装部) */
  2
  3 #include <sstream>
  4 #include <iostream>
  5 #include <iomanip>
  6 #include <string>
```

```
 7 #include "ComplexNum.h"
 8 using namespace std;
 9
10 // 各種コンストラクタ ---------------------------------
11
12 // 複素数オブジェクト操作の関数群 ----------------------
13 // 保持している複素数string型データとして返す
14 string ComplexNum::toString() const {
15   ostringstream ostr;
16   ostr << "(" << re << ")+(" << im << ")i";
17   return ostr.str();
18 }
19
20 string ComplexNum::toString(int precision) const {
21   ostringstream ostr;
22   ostr << scientific << setprecision(precision)
23        << "(" << setw(precision+7) << re << ")+("
```

```
24          << setw(precision+7) << im << ")i";
25    return ostr.str();
26 }
27
28 // 入出力ストリームとの演算 ----------------------------
29 // 出力ストリームへComplexNum型データを挿入する演算子
                                              // << の多重定義
30 ostream& operator<<(ostream& out,
                                    const ComplexNum& y) {
31    return (out << y.toString());
32 }
33
34 // 入力ストリームからComplexNum型データを抽出する演算子
                                              // >> の多重定義
35 istream& operator>>(istream& in, ComplexNum& y) {
36    return (in >> y.re >> y.im);
37 }
```

```
[motoki@x205a]$
```

(ComplexNumクラスの利用)

ComplexNumクラスが上の様に定義されていれば、
それを利用して次の様なプログラムを書くこともできる。

```
[motoki@x205a]$  cat -n useComplexNum.cpp
     1 /* ComplexNum.h, ComplexNum.cpp の利用例 */
     2
     3 #include <iostream>
     4 #include <string>
     5 #include "ComplexNum.h"
     6 using namespace std;
     7
     8 int main()
     9 {
    10   ComplexNum x, y, z;
    11
```

```
12    cin >> x >> y;
13    cout << "x=" << x << ", y=" << y << endl;
14    z = (x+y+0.5)/5;
        ↑                          コピーコンストラクタが呼び出される
15    cout << "==> z = (x+y+0.5)/5 = " << z << endl;
16 }
[motoki@x205a]$  g++ useComplexNum.cpp ComplexNum.cpp
[motoki@x205a]$  ./a.out
1 2 3 4
x=(1)+(2)i, y=(3)+(4)i
==> z = (x+y+0.5)/5 = (0.9)+(1.2)i
[motoki@x205a]$
```

例題4.4で要求されている作業に関しては、
例えば次のC++プログラムの様に表すことができる。

注目：
実数値多項式を計算するコードとほぼ同じ

```
[motoki@x205a]$  cat -n calcComplexPolynomials.cpp
     1 /* 非負整数データ n と複素数データ C_n, C_n-1, ..., C_0 *
     2 /* を読み込み、これらの定数値の下で複素多項式              *
     3 /*        C_n*z^n + C_n-1*z^n-1 + ... + C_1*z + C_0     *
     4 /* の値が                                               *
     5 /*        z = exp(i π k/5)   (k=0,1,2, ..., 9)          *
     6 /* のそれぞれの値に対してどの様に変化するかを、          *
     7 /* 表の形に見易く出力するC++プログラム                   *
     8
     9 #include <iostream>
    10 #include <iomanip>
    11 #include <cmath>
```

```
12 #include "ComplexNum.h"
13 using namespace std;
14
15 const int MAX_DEGREE = 100;
16 const double PI = 3.1415926535897932;   // 円周率
17
18 int main()
19 {
20   int degree;
21   ComplexNum coeff[MAX_DEGREE+1];
22
23   //多項式の次数と係数を標準入力から読み込む
24   cout << "複素多項式の次数(100以下):";
25   cin >> degree;
26   for (int i=degree; i>=0; --i) {
27     cout << i << "次係数の実部と虚部:";
28     double re, im;
```

```
29      cin >> re >> im;
30      coeff[i].setRe(re);
31      coeff[i].setIm(im);
32    }
33
34    //入力された値の確認
35    cout << "degree = " << degree << endl;
36    for (int i=degree; i>=0; --i) {
37      cout << "coeff[" << i << "] = " << coeff[i].toStrin
38    }
39
40    //多項式の値を計算して出力
41    cout << " k                          z                    "
42         << "  c[d]*z^d+c[d-1]*z^(d-1)+ ... +c[1]*z+c[0]'
43         << "--  ------------------------------"
44         << "  ------------------------------------'
45    for (int k=0; k<10; ++k) {
```

```
46      ComplexNum z(cos(PI*k/5.0), sin(PI*k/5.0));
47      ComplexNum result= coeff[degree];
48      for (int i=degree-1; i>=0; --i)
49        result = result*z + coeff[i];
50      cout << setw(2) << k << "   "
51           << z.toString(6) << "   "
52           << result.toString(6) << endl;
53    }
54 }
```

```
[motoki@x205a]$  g++ calcComplexPolynomials.cpp
                                              ComplexNum.cpp
[motoki@x205a]$  ./a.out
複素多項式の次数(100以下)： 3
3次係数の実部と虚部： 1.0 -2.0
2次係数の実部と虚部： -3.0 4.0
1次係数の実部と虚部： 5.0 -6.0
0次係数の実部と虚部： -7.0 8.0
```

```
degree = 3
coeff[3] = (1)+(-2)i
coeff[2] = (-3)+(4)i
coeff[1] = (5)+(-6)i
coeff[0] = (-7)+(8)i
 k                    z                          c[d]*z^d+c[d-1]*z^(d-1)+
--  ---------------------------------  ---------------------------------
 0  ( 1.000000e+00)+( 0.000000e+00)i  (-4.000000e+00)+( 4.0000
 1  ( 8.090170e-01)+( 5.877853e-01)i  (-2.566385e+00)+( 6.0368
 2  ( 3.090170e-01)+( 9.510565e-01)i  (-1.657253e+00)+( 6.9320
 3  (-3.090170e-01)+( 9.510565e-01)i  ( 1.572893e+00)+( 1.0930
 4  (-8.090170e-01)+( 5.877853e-01)i  (-2.430068e+00)+( 2.0215
 5  (-1.000000e+00)+( 1.224647e-16)i  (-1.600000e+01)+( 2.0000
 6  (-8.090170e-01)+(-5.877853e-01)i  (-2.089617e+01)+( 6.7289
 7  (-3.090170e-01)+(-9.510565e-01)i  (-1.219093e+01)+(-9.3085
 8  ( 3.090170e-01)+(-9.510565e-01)i  (-6.016509e+00)+( 2.1237
 9  ( 8.090170e-01)+(-5.877853e-01)i  (-5.815581e+00)+( 3.9631
```

```
[motoki@x205a]$
```

4–2–5 線形連結リストのクラス

例題4. 5 (線形連結リストのクラス)　前ターム「プログラミングAI」の例題12.3では、

① 入力ストリームに現れる 識別子 ＿＿＿ 整数 という形のデータを読み込んでは、
（＿＿＿：1個以上の空白）

② それを 識別子 に関して辞書順になる様に線形リストに登録する、

という作業を繰り返し、最後に線形リストに登録されたものを順に出力するCプログラムを作成した。これに対して、ここでは(ほぼ)同等の処理を行うプログラムをC++で構成してみよ。

(考え方)

識別子と整数を要素にもつ多数のNodeを
識別子の辞書順に連結して保持する

エージェントが居れば、
全体の作業を見通し良く記述することができる。

⇨ この種のエージェントをインスタンスとするクラス
SortedLinkedListOfIdInt を定義。その際、

- 次の理由で、**Node**は
  公開のデータ要素とコンストラクタだけから成る構造体
  として表す。

  ◇ **Node**内のデータ要素を非公開にする。
  ⟶ SortedLinkedListOfIdIntエージェント内での作業に
  余分な手間がかかる見込み。

  ◇ **Node**内に関数メンバを用意して協調動作させる。
  ⟶ コードが分散されてしまう。

- 上の例題**4.5**で必要な関数だけでなく、将来のため、
  あったら便利そうな関数をメンバ関数として用意する。

(SortedLinkedListOfIdIntクラスの定義)　定義例を示す。

```
[motoki@x205a]$  cat -n SortedLinkedListOfIdInt.h
     1  /* 識別子と整数を要素にもつNode群を識別子の辞書順に*/
```

```
 2 /* 連結した、連結リストのクラス (仕様部)                    */
 3
 4 #ifndef ___Class_SortedLinkedListOfIdInt
 5 #define ___Class_SortedLinkedListOfIdInt
 6
 7 #include <cstddef>   // for NULL
 8 #include <string>
 9
10 struct NodeStringInt {
11   const std::string id;
12   const int num;
13   NodeStringInt* next;
14   NodeStringInt(std::string id="", int num=0,
                                    NodeStringInt* next=NULL)
15     : id(id), num(num), next(next) {}
16 };
17
```

```
18 class SortedLinkedListOfIdInt {
19   NodeStringInt* head;
20 public:
21   SortedLinkedListOfIdInt(): head(NULL) {}
22   ~SortedLinkedListOfIdInt() { removeAllNodes(); }
23   NodeStringInt getHeadNodeInfo() const;
                                    //先頭Node内の情報を返す
24   NodeStringInt get1stNodeInfoOf(const std::string&
                                          id) const;
                                //引数指定の最初のNode情報を返す
25   int getNumOfNodesOf(const std::string& id) const;
                                      //引数指定のNode数返却
26   void printAllNodeInfo() const;
                                    //全てのNode内の情報を出力
27   void insertNodeOf(const std::string& id,
                       const int& num);    //新データ挿入
28   int removeHeadNode();                 //先頭Nodeの削除
```

```
29    int remove1stNodeOf(const std::string& id);
                                    //引数指定の最初のNodeを削除
30    int removeAllNodesOf(const std::string& id);
                                    //引数指定の全Nodeを削除
31    void removeAllNodes();                    //全Nodeを削除
32 };
33
34 #endif
[motoki@x205a]$  cat -n SortedLinkedListOfIdInt.cpp
 1 /* 識別子と整数を要素にもつNode群を識別子の辞書順に*/
 2 /* 連結した、連結リストのクラス (実装部)              */
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 #include <cstddef>   // for NULL
 7 #include <string>
 8 #include "SortedLinkedListOfIdInt.h"
```

```
 9 using namespace std;
10
11 // 連結リストの先頭にあるNodeのコピーを返す
12 NodeStringInt SortedLinkedListOfIdInt::
                                    getHeadNodeInfo() const {
13   return NodeStringInt(head->id, head->num, NULL);
14 }
15
16 // 連結リスト中でid要素=引数の最初のNodeのコピーを返す
17 NodeStringInt SortedLinkedListOfIdInt::
18             get1stNodeInfoOf(const string& id) const {
19   NodeStringInt* ptr = head;
20   while (ptr != NULL && ptr->id <= id) {
21     if (ptr->id == id) {
22       return NodeStringInt(ptr->id, ptr->num, NULL);
23     }
24     ptr = ptr->next;
```

```
25   }
26   return NodeStringInt("", 0, NULL);
27 }
28
29 // 連結リスト中で引数をid要素とするNodeの個数を返す
30 int SortedLinkedListOfIdInt::getNumOfNodesOf(
                                  const string& id) const {
31   NodeStringInt* ptr = head;
32   while (ptr != NULL && ptr->id < id)
33     ptr = ptr->next;
34   int count;
35   for (count=0; ptr != NULL && ptr->id == id; ++count)
36     ptr = ptr->next;
37   return count;
38 }
39
40 // 連結リスト内に保持されている内容を表の形に出力
```

```
41 void SortedLinkedListOfIdInt::printAllNodeInfo()
                                                    const {
42   cout << "番号       num          id" << endl
43        << "----  ----------  ----------" << endl;
44   int count = 0;
45   for (NodeStringInt* ptr=head; ptr != NULL;
                                          ptr = ptr->next) {
46     cout << setw(4) << ++count
47          << setw(12) << ptr->num << "  "
                                       << ptr->id << endl;
48   }
49 }
50
51 // 引数要素をもつNodeを連結リスト内の
                                  // idの辞書順を保つ位置に挿入
52 void SortedLinkedListOfIdInt::insertNodeOf(
                        const string& id, const int& num) {
```

```
53   NodeStringInt** ptrptr = &head;
54   while ((*ptrptr)!=NULL && (*ptrptr)->id <= id)
55     ptrptr = &((*ptrptr)->next);
56   *ptrptr = new NodeStringInt(id, num, *ptrptr);
57 }
58
59 // 連結リスト中の先頭Nodeを削除し、削除したNode数を返す
60 int SortedLinkedListOfIdInt::removeHeadNode() {
61   if (head == NULL)
62     return 0;
63   NodeStringInt* tmp = head;
64   head = head->next;
65   delete tmp;
66   return 1;
67 }
68
```

```
69 // 連結リスト中で引数をid要素とする最初のNode削除し、
                                    // 削除したNode数を返す
70 int SortedLinkedListOfIdInt::remove1stNodeOf(
                                          const string& id) {
71   NodeStringInt** ptrptr = &head;
72   while (*ptrptr != NULL && (*ptrptr)->id < id)
73     ptrptr = &((*ptrptr)->next);
74   if (*ptrptr != NULL && (*ptrptr)->id == id) {
75     NodeStringInt* tmp = *ptrptr;
76     *ptrptr = (*ptrptr)->next;
77     delete tmp;
78     return 1;
79   }else {
80     return 0;
81   }
82 }
```

```
83
84 // 連結リスト中で引数をid要素とするNodeを全て削除し、
                                        削除したNode数を返す
85 int SortedLinkedListOfIdInt::removeAllNodesOf(
                                              const string& id) {
86   NodeStringInt** ptrptr = &head;
87   while (*ptrptr != NULL && (*ptrptr)->id < id)
88     ptrptr = &((*ptrptr)->next);
89   int count = 0;
90   for (count=0; *ptrptr != NULL
                        && (*ptrptr)->id == id; ++count) {
91     NodeStringInt* tmp = *ptrptr;
92     *ptrptr = (*ptrptr)->next;
93     delete tmp;
94   }
95   return count;
96 }
```

```
 97
 98 // 連結リスト中のNodeを全て削除
 99 void SortedLinkedListOfIdInt::removeAllNodes() {
100   while (head != NULL) {
101     NodeStringInt* tmp = head;
102     head = head->next;
103     delete tmp;
104   }
105 }
[motoki@x205a]$
```

(SortedLinkedListOfIdIntクラスの利用)

SortedLinkedListOfIdIntクラスが上の様に定義されていれば、
それを利用して例題4.5で要求されている作業を次の様に...

```
[motoki@x205a]$  cat -n
                        readIdNumSeqAndPrintInLexicalOrder.cpp
 1 /* 線形連結リストを用いて
```

```
                                識別子(と整数)を辞書順に登録・出力 */
 2
 3 #include <iostream>
 4 #include <cstdlib>
 5 #include "SortedLinkedListOfIdInt.h"
 6 using namespace std;
 7
 8 int main()
 9 {
10   SortedLinkedListOfIdInt list;
11   string id;
12   int num;
13
14   cin >> id >> num;
15   while (cin.good()) {   入力時の問題なし→非0、あり→0
16     list.insertNodeOf(id, num);
17     cin >> id >> num;
```

```
18   }
19
20   if (! cin.eof()) {                    入力終端→非0、まだ→0
21     cout << "Input Error!" << endl;
22     exit(EXIT_FAILURE);
23   }
24
25   list.printAllNodeInfo();
26 }
[motoki@x205a]$  g++ readIdNumSeqAndPrintInLexicalOrder.cpp
                                  SortedLinkedListOfIdInt.cpp
[motoki@x205a]$  ./a.out < dynamic-llist.data
番号       num          id
----   ----------   ----------
   1          123   abc
   2        55555   asdfghjk
   3           77   efghi
```

```
   4          456  kkk
   5         2345  ppp_qqq
   6       222222  qwertyui
   7          987  zxcv
[motoki@x205a]$
```

4–2–6 トランプ札の配り手のクラス

例題4. 6 (トランプ札の配り手のクラス)　トランプの(Jokerを除く)カード5枚がランダムに配られた時、それがPokerのストレートフラッシュになる確率、4カードになる確率、フルハウスになる確率、フラッシュになる確率、ストレートになる確率、3カードになる確率、2ペアになる確率、1ペアになる確率、ノーペアになる確率をMonteCarlo法により見積もるC++プログラムを作成せよ。
すなわち、コンピュータ上で①ランダムに5枚のカードを配り②その役を調べる、という作業を繰り返すシミュレーションを行い、それぞれの役が出る割合を求めるC++プログラムを作成せよ。

(考え方)

トランプの(Jokerを除く)カードを公正にシャッフルして配る

エージェントが居れば、
あとは配られた手配の役を見極めることだけが主な作業。

⇨ この種のエージェントをインスタンスとするクラス
CardDealer を定義。その際、

- 次の理由で、個々のカードを
  スーツ と ランク を公開メンバとする構造体Card
  (e.g. スペード) (2~10,J,Q,K,A)
  として表す。

  ◇ 個々のカードの内容はトランプゲームを行う際の基本データ。
  ⟶ 非公開にすると 煩わしいことも多い。

- スーツ(e.g. スペード)とランク(2~10,J,Q,K,A)は、どちらも、数値データとは言い難い。⟶列挙型定数として表す。

- 上の例題4.6で必要な関数だけでなく、将来のため、
  あったら便利そうな関数をメンバ関数として用意する。

(CardDealerクラスの定義) 定義例を示す。

[motoki@x205a]$ cat -n CardDealer.h

```
 1 /* トランプのディーラーのクラス CardDealer (仕様部) */
 2
 3 #ifndef ___Class_CardDealer
 4 #define ___Class_CardDealer
 5
 6 #include <string>
 7
 8 enum Suit { CLUB, DIAMOND, HEART, SPADE };
 9 enum Rank { ILLEGAL, RANK2=2,
                              RANK3, RANK4, RANK5, RANK6,
10              RANK7, RANK8, RANK9, RANK10,
                                 JACK, QUEEN, KING, ACE };
11
12 struct Card {
13   Suit suit;                                  列挙タグもデータ型名
14   Rank rank;
15   std::string toString() const {    Card型→文字列表記
```

```
16      const std::string SymbolOfSuit[4] = {"CLB",
                                             "DIA","HRT","SPD"};
17      const std::string SymbolOfRank[15] = {"",
                           "","_2","_3","_4","_5","_6","_7",
18                       "_8","_9","10","_J","_Q","_K","_A"};
19      return SymbolOfSuit[suit]+SymbolOfRank[rank];
20    }
21 };
22
23 class CardDealer {
24    Card card[52];     //Jokerを除く52枚のカードを保持
25    int numOfDealedCards;
                              //配付済みで手元にないカードの枚数
26 public:
27    CardDealer();
28    void renewCardDeck();
29    void shuffle();
```

```
    30    Card deal1Card();                    //カードを1枚配る
                                               //(戻り値が次のカード)
    31    void dealCards(const int numOfCards, Card* hand);
    32                          //引数で指定された枚数だけ、
                                  //カードを引数指定の配列上に配る
    33 };
    34
    35 #endif
[motoki@x205a]$  cat -n CardDealer.cpp
     1 /* トランプのディーラーのクラス CardDealer (実装部) */
     2
     3 #include <iostream>
     4 #include <string>
     5 #include <cstdlib>
     6 #include <ctime>
     7 #include "CardDealer.h"
     8 using namespace std;
```

```
 9                                              int型→Suit型の変換のため
10 Suit toSuit[4] = {CLUB, DIAMOND, HEART, SPADE};
11 Rank toRank[15] = {ILLEGAL, ILLEGAL, RANK2,
                                    RANK3, RANK4, RANK5, RANK6,
12                            RANK7, RANK8, RANK9, RANK10,
                                      JACK, QUEEN, KING, ACE};
13                                            int型→Rank型の変換のため
14 // コンストラクタ ------------------------------------
15 CardDealer::CardDealer()
16 {
17   renewCardDeck();
18   srand(time(NULL));
19   shuffle();
20 }
21
22 // カードデッキ操作の関数群 ----------------------------
23 // カードの束を新しいものと取り替える
```

```
24 void CardDealer::renewCardDeck()
25 {
26   int k = 0;                                  Suit型→int型の自動変換
27   for (int suit=CLUB; suit<=SPADE; ++suit) {
28     card[k].suit = toSuit[suit];              int型→Suit型
29     card[k].rank = ACE;
30     ++k;
31     for (int rank=RANK2; rank<=KING; ++rank) {
32       card[k].suit = toSuit[suit];            int型→Suit型
33       card[k].rank = toRank[rank];            int型→Rank型
34       ++k;
35     }
36   }
37   numOfDealedCards = 0;
38 }
39
40 // 保持しているカードの束を混ぜる
```

```
41 void CardDealer::shuffle()
42 {
43   for (int k=numOfDealedCards; k<52; ++k) {
44     int i = k + (rand() % (52-k));
45     Card tmp = card[k];  //swap cards
46     card[k] = card[i];
47     card[i] = tmp;
48   }
49 }
50
51 // 次の1枚を配る
52 Card CardDealer::deal1Card()
53 {
54   if (numOfDealedCards >= 52) {
55     numOfDealedCards = 0;   //新しいカードを一組用意
                                //(renewCardDeck()省略)
56     shuffle();
```

```
57    }
58    return card[numOfDealedCards++];
59  }
60
61  // 引数指定の枚数を引数の配列上に配る
62  void CardDealer::dealCards(const int numOfCards,
                                                    Card* hand)
63  {
64    if (numOfCards < 1 || 52 < numOfCards) {
65      cout << "Invalid number of cards is specified "
                                              "in dealCards()."
66           << endl;
67      exit(EXIT_FAILURE);
68    }else if (numOfCards > 52-numOfDealedCards) {
69      numOfDealedCards = 0;   //新しいカードを一組用意
                                        //(renewCardDeck()省略)
70      shuffle();
```

```
    71    }
    72    for (int k=0; k<numOfCards; ++k) {
    73      hand[k] = card[numOfDealedCards+k];
    74    }
    75    numOfDealedCards += numOfCards;
    76  }
[motoki@x205a]$
```

(CardDealerクラスの利用)

配られた5枚のカードに関して、

◇ ストレートフラッシュ

⇔ (1種類のsuit, 5種類のrank,
最大rank~(最大rank−4)のカードが1枚ずつ),
または
(1種類のsuit, 5種類のrank, Aと2~5のカードが1枚ずつ)

◇ 4カード⇔ 4枚揃ったrankが1つある

◇ フルハウス $\Leftrightarrow$ 上記以外, 2種類のrank

◇ フラッシュ $\Leftrightarrow$ 上記以外, 1種類のsuit

◇ ストレート

$\Leftrightarrow$ (上記以外, 5種類のrank,
最大rank~(最大rank−4)のカードが1枚ずつ),
または
(上記以外, 5種類のrank, Aと2~5のカードが1枚ずつ)

◇ 3カード $\Leftrightarrow$ 上記以外, 3枚揃ったrankが1つある

◇ 2ペア $\Leftrightarrow$ 上記以外, 2枚揃ったrankが2つある

◇ 1ペア $\Leftrightarrow$ 上記以外, 2枚揃ったrankが1つある

◇ ノーペア $\Leftrightarrow$ 上記以外

⇨ `CardDealer`クラスが上の様に定義されていれば 、
それを利用して例題4.6で要求されている作業を次の様に...

```
[motoki@x205a]$ cat -n estimateProbOfDrawingPokerHand.cpp
     1 /* Pokerのそれぞれの役が出る確率を
                                        MonteCarlo手法で見積もる */
     2
     3 #include <iostream>
     4 #include <cstdlib>
     5 #include <ctime>
     6 #include "CardDealer.h"
     7 using namespace std;
     8
     9 int main()
    10 {
    11   CardDealer dealer;  初期値
    12   Card cardInHand[5];    ↓
    13   int freqStraightFlush(0), freq4OfAKind(0),
           ↑StraightFlushになった回数    freqFullHouse(0),
    14     freqFlush(0), freqStraight(0), freq3OfAKind(0),
```

```
15      freq2Pair(0), freq1Pair(0), freqNoPair(0);
                ↓                                  各々のスーツの枚数
16    int freqSuit[4], numOfSuitTypes;  ←スーツの種類数
                ↓                                  各々のランクの枚数
17    int freqRank[15], numOfRankTypes,  ←ランクの種類数
                          freqOfFreqRankValue[5], highestRank;
添字数の枚数分揃ったランクの種類数 ↑                          ↑
18              10億回試行 ↓                       最大ランク値
19    for (int i=1; i<=1000000000; ++i) {
20      dealer.dealCards(5, cardInHand);
21      //手配カードのsuitやrankの分布を調べる
22      for (int k=CLUB; k<=SPADE; ++k)
23        freqSuit[k] = 0;
24      for (int k=RANK2; k<=ACE; ++k)
25        freqRank[k] = 0;
26      for (int k=0; k<5; ++k) {
27        ++freqSuit[cardInHand[k].suit];
```

```
28        ++freqRank[cardInHand[k].rank];
29      }
30      numOfSuitTypes = 0;
31      for (int k=CLUB; k<=SPADE; ++k) {
32        if (freqSuit[k] > 0)
33          ++numOfSuitTypes;
34      }
35      numOfRankTypes = 0;
36      highestRank = RANK2;
37      for (int k=0; k<5; ++k)
38        freqOfFreqRankValue[k] = 0;
39      for (int k=RANK2; k<=ACE; ++k) {
40        if (freqRank[k] > 0) {
41          ++numOfRankTypes;
42          highestRank = k;
43        }
44        ++freqOfFreqRankValue[freqRank[k]];
```

```
45      }
46      //手配カードの役を調べる
47      if (numOfSuitTypes==1 && numOfRankTypes==5
48          && freqRank[highestRank]==1
                                && freqRank[highestRank-1]==1
49          && freqRank[highestRank-2]==1
                                && freqRank[highestRank-3]==1
50          && freqRank[highestRank-4]==1)
51        ++freqStraightFlush;
52      else if (numOfSuitTypes==1 && numOfRankTypes==5
53                && freqRank[ACE]==1 && freqRank[2]==1
                                        && freqRank[3]==1
54                && freqRank[4]==1 && freqRank[5]==1)
55        ++freqStraightFlush;
56      else if (freqOfFreqRankValue[4] == 1)
57        ++freq4OfAKind;
58      else if (numOfRankTypes == 2)
```

```
59        ++freqFullHouse;
60      else if (numOfSuitTypes==1)
61        ++freqFlush;
62      else if (numOfRankTypes==5
63               && freqRank[highestRank]==1
                          && freqRank[highestRank-1]==1
64               && freqRank[highestRank-2]==1
                          && freqRank[highestRank-3]==1
65               && freqRank[highestRank-4]==1)
66        ++freqStraight;
67      else if (numOfRankTypes==5
68               && freqRank[ACE]==1 && freqRank[2]==1
                                     && freqRank[3]==1
69               && freqRank[4]==1 && freqRank[5]==1)
70        ++freqStraight;
71      else if (freqOfFreqRankValue[3] == 1)
72        ++freq3OfAKind;
```

```
73      else if (freqOfFreqRankValue[2] == 2)
74        ++freq2Pair;
75      else if (freqOfFreqRankValue[2] == 1)
76        ++freq1Pair;
77      else
78        ++freqNoPair;
79      //時々、途中経過を画面に表示実行終了の不安を和らげる
80      if (i%100000000 == 0) {
81        time_t currentTime = time(NULL);
82        cout << i                    1970年1月1日からの経過秒数
                << "-th hand's rank has been determined: "
83              << ctime(&currentTime);
84      }              ↑       ローカルな時刻を表す文字列に変換
85    }
86    //結果を表示
87    cout << "ストレートフラッシュの確率の見積り："
88         << freqStraightFlush/1e9 << endl
```

```
89          << "4カード　の確率の見積り："
                                    << freq4OfAKind/1e9 << endl
90          << "フルハウスの確率の見積り："
                                  << freqFullHouse/1e9 << endl
91          << "フラッシュの確率の見積り："
                                      << freqFlush/1e9 << endl
92          << "ストレートの確率の見積り："
                                    << freqStraight/1e9 << endl
93          << "3カード　の確率の見積り："
                                    << freq3OfAKind/1e9 << endl
94          << "2ペア　　の確率の見積り："
                                      << freq2Pair/1e9 << endl
95          << "1ペア　　の確率の見積り："
                                      << freq1Pair/1e9 << endl
96          << "ノーペア　の確率の見積り："
                                     << freqNoPair/1e9 << endl;
97 }
```

```
[motoki@x205a]$ g++ estimateProbOfDrawingPokerHand.cpp
                                                CardDealer.cpp
[motoki@x205a]$ ./a.out
100000000-th hand's rank has been determined: Thu Oct 19 09:34
200000000-th hand's rank has been determined: Thu Oct 19 09:35
300000000-th hand's rank has been determined: Thu Oct 19 09:35
400000000-th hand's rank has been determined: Thu Oct 19 09:35
500000000-th hand's rank has been determined: Thu Oct 19 09:36
600000000-th hand's rank has been determined: Thu Oct 19 09:36
700000000-th hand's rank has been determined: Thu Oct 19 09:37
800000000-th hand's rank has been determined: Thu Oct 19 09:37
900000000-th hand's rank has been determined: Thu Oct 19 09:37
1000000000-th hand's rank has been determined: Thu Oct 19 09:3
ストレートフラッシュの確率の見積り：1.5332e-05
4カード　の確率の見積り：0.000239412
フルハウスの確率の見積り：0.00144001
フラッシュの確率の見積り：0.00196517
```

ストレートの確率の見積り：0.00392315
3カード　の確率の見積り：0.0211279
2ペア　　の確率の見積り：0.0475396
1ペア　　の確率の見積り：0.422551
ノーペア　の確率の見積り：0.501198
[motoki@x205a]$