5 既定義クラスの拡張、多態性の実現

5–1 既定義クラスを基にしたクラス定義

既定義クラスを利用したクラス定義、継承：

既に定義されたクラスを利用して新しいクラスを定義できる。

C++言語では、

- 基底クラス ... 元になったクラス、
- 派生クラス ... 新しく定義したクラス

具体的には、C++言語では次の様に書く。

```
class 派生クラス名 : アクセス指定子,省略可 基底クラス名 {
        新しいメンバの宣言、定義、など
}
```

もしくは、

```
struct 派生クラス名 : アクセス指定子,省略可 基底クラス名 {
    新しいメンバの宣言、定義、など
}
```

ここで、

- これまで通り 、
  - ◇ キーワード `class` ⟶ 新しく宣言されたメンバは暗黙に `private`
  - ◇ キーワード `struct` ⟶ 新しく宣言されたメンバは暗黙に `public`

- 新しいメンバへのアクセス可否を指定するキーワードとして 、
  - ◇ `private`
  - ◇ `public`
  - ◇ `protected` ... 派生クラス(とその派生,...)のオブジェクト
    に対してのみ限定公開

- 基底クラス名の前の アクセス指定子,省略可

... 基底クラス内のメンバの、
派生クラスのメンバとしての公開／非公開を指定
private, protected, public のいずれか (省略時→private)

◇ private と指定された(または無指定の)場合は 、 ...private派生

基底クラス内での指定		派生クラスのメンバとしての扱い	派生クラス内からのアクセス
private	⟶	private	不可
protected	⟶	private	可
public	⟶	private	可

⇨ 既存クラスを内部で利用して全く新しいクラスを構成する場合

◇ protected と指定された場合は 、 ...protected派生

基底クラス内での指定		派生クラスのメンバとしての扱い	派生クラス内からのアクセス
private	⟶	private	不可
protected	⟶	protected	可
public	⟶	protected	可

◇ public と指定された場合は 、 ...public派生

基底クラス内での指定		派生クラスのメンバとしての扱い	派生クラス内からのアクセス
private	⟶	private	不可
protected	⟶	protected	可
public	⟶	public	可

⇨ 既存クラスを拡張して新しいクラスを構成することになる。

基底クラスのインタフェースは派生クラスにそのまま継承

- 基底クラスで定義されたメンバは、
  コンストラクタ,デストラクタ,関数operator=()を除いて、
  派生クラスに暗黙に継承される。
  すなわち 、

  ◇ 非staticなデータメンバ
  ... 派生クラスのインスタンスにおいても領域確保される。

  ◇ 基底クラスで定義された非staticなメンバ関数
  ... 派生クラスの中で再定義／上書きされてなければ、
  (オーバーライド)
  派生クラスのメンバ関数として暗黙に使用できる。

  ⇨ 継承をうまく利用すれば、同じ定義をあちこちのクラス定義の中で繰り返す手間はかなり省ける様になる。

- 基底クラスで定義されたコンストラクタ,デストラクタは
  派生クラスに継承されない。

派生クラスのコンストラクタ： 次の形に書く。

派生クラス名（仮引数列）： 基底クラス名（実引数列），
新データメンバ名（初期値の式），．．．，
新データメンバ名（初期値の式）｛

初期設定の仕上げ(必要な場合)

｝

この中の、

- 「： 基底クラス名（実引数列），
．．．， 新データメンバ名（初期値の式）」の部分(初期化子リスト)
… データメンバの初期化をどう行うかを指示

- 「基底クラス名（実引数列）」の部分(コンストラクタ初期化子)
… 基底クラスのコンストラクタの呼び出しを指示
(省略 $\longrightarrow$ 基底クラスのデフォルトコンストラクタ)

- 実際の初期化は 次の順に

① 基底クラスのコンストラクタの呼び出し
(省略 $\longrightarrow$ デフォルトコンストラクタ)

② データメンバの初期化 (クラス定義内でメンバの並んだ順)

③ 派生クラスのコンストラクタの本体部

⇨ 初期化子リストで省略を行うと $\longrightarrow$①,②で支障が出る可能性

(e.g. コンストラクタ初期化子の記述を省略
$\longrightarrow$ デフォルトコンストラクタが無いとコンパイルエラー)

派生クラス名 (仮引数列) : 基底クラス名 (実引数列),
新データメンバ名 (初期値の式), ..., 新データメンバ名 (初期値の式) {

初期設定の仕上げ (必要な場合)

}

例5. 1 (色付き長方形のクラス,Rectangleクラスの拡張)

例題4.2... 長方形を表すオブジェクトのクラス Rectangle を示した。

ここでは、このクラスを拡張して色付き長方形を表すオブジェクトのクラス ColoredRectangle を定義

```
[motoki@x205a]$  cat -n ColoredRectangle.h
     1 /* 色付き長方形オブジェクトのクラス ColoredRectangle
                                                (仕様部) */
     2 /* (既定義の Rectangleクラス の拡張)                  */
     3
     4 #ifndef ___Class_ColoredRectangle
     5 #define ___Class_ColoredRectangle
     6
     7 #include <string>
     8 #include "Rectangle.h"
     9
    10 class ColoredRectangle : public Rectangle {
```

public派生→クラスの拡張

```
11    std::string color;
12 public:
13    ColoredRectangle(double width = 0.0,
                                        double height = 0.0,
14                          std::string color = "black")
15      : Rectangle(width, height), color(color) {}
```

↑基底クラスのコンストラクタ呼出し　初期化子リスト

```
16    ColoredRectangle(const coloredRectangle& rectangle);
                                                  //コピーコンストラクタ

17    // オブジェクト(もしくはクラス全体)の情報を提供する
                                                  // ための関数群(追加)
18    std::string getColor() const { return color; }
19    void setColor(std::string color) {
                                          this->color = color; }
20    std::string toString() const; //内部の長方形情報を
```

オーバーライド　　//stringデータとして返す

```
21 };
22
23 #endif
[motoki@x205a]$  cat -n ColoredRectangle.cpp
 1 /* 色付き長方形オブジェクトのクラス ColoredRectangle
                                                (実装部) */
 2 /* (既定義の Rectangleクラス の拡張)                  */
 3
 4 #include <sstream>
 5 #include <string>
 6 #include "ColoredRectangle.h"
 7 using namespace std;
 8
 9 // 各種コンストラクタ ---------------------------------
```

```
10 ColoredRectangle::ColoredRectangle(const
       ColoredRectangle& rectangle) //コピーコンストラクタ
11                     : Rectangle(rectangle.getWidth(),
              コンストラクタ初期化子 rectangle.getHeight()),
12                       color(rectangle.color) {}
13                                          初期化子リスト
14 // ColoredRectangle型オブジェクトを操作するための関数群 --
15 // 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
16 string ColoredRectangle::toString() const
17 {                              オーバーライド
18   ostringstream ostr;
19   ostr << "rectangle(id=" << getId()
20        << ",width=" << getWidth()
                             << ",height=" << getHeight()
21        << ",color=\"" << color << "\")";
```

```
    22   return ostr.str();
    23 }
[motoki@x205a]$
```

これに関して、

- 上のクラス ColoredRectangle 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 useColoredRectangle.cpp
     1 /* ColoredRectangle.h, ColoredRectangle.cpp の利用例 ›
     2
     3 #include <iostream>
     4 #include <string>
     5 #include "ColoredRectangle.h"
     6 using namespace std;
     7
     8 int main()
     9 {
    10   ColoredRectangle rect;
```

```
11
12    cout << "(1)" << rect.toString() << endl
13         << "  |rect.getId()=" << rect.getId()
14         <<        ", rect.getWidth()="
                              << rect.getWidth() << endl
15         << "  |rect.getHeight()=" << rect.getHeight()
16         <<        ", rect.getColor()=\""
                       << rect.getColor() << "\"" << endl
17         << "  |rect.getArea()=" << rect.getArea()
                                                 << endl;
18    rect.setWidth(2.0);
19    rect.setHeight(3.0);
20    rect.setColor("blue");
21    cout << "(2)" << rect.toString() << endl
22         << "  |rect.getId()=" << rect.getId()
23         <<        ", rect.getWidth()="
                              << rect.getWidth() << endl
```

継承
継承
継承
継承
継承
継承

```
24          << "  |rect.getHeight()=" << rect.getHeight()
25          <<        ", rect.getColor()=\""                    継承
                        << rect.getColor() << "\"" << endl
26          << "  |rect.getArea()=" << rect.getArea()
                                                  継承 << endl;
27
28    //cout << "(3)" << (new ColoredRectangle(4.0, 5.0,
                                        // "red"))->toString()
      //      << endl;                 // heap領域のメモリが
                                       // 開放されないままになる
29    ColoredRectangle* ptrRect
                  = new ColoredRectangle(4.0, 5.0, "red");
30    cout << "(3)" << ptrRect->toString() << endl;
31    delete ptrRect;
32 }
[motoki@x205a]$  g++ useColoredRectangle.cpp
                       ColoredRectangle.cpp Rectangle.cpp
```

```
[motoki@x205a]$  ./a.out
(1)rectangle(id=0,width=0,height=0,color="black")
  |rect.getId()=0, rect.getWidth()=0
  |rect.getHeight()=0, rect.getColor()="black"
  |rect.getArea()=0
(2)rectangle(id=0,width=2,height=3,color="blue")
  |rect.getId()=0, rect.getWidth()=2
  |rect.getHeight()=3, rect.getColor()="blue"
  |rect.getArea()=6
(3)rectangle(id=1,width=4,height=5,color="red")
[motoki@x205a]$
```

例5. 2 (色付き長方形のクラス,protected指定子を用いてRectangleクラ

例題4.2Rectangleクラスのコード

... 将来の派生を全く考えずに データメンバを全て非公開 (private)

しかし、将来の派生を考慮に入れて
データメンバを限定公開(protected) する選択肢も...

```
[motoki@x205a]$  cat -n Rectangle_verProt.h
     1 /* 長方形オブジェクトのクラス Rectangle (仕様部) */
     2 /* (protectedデータ版)                              */
     3
     4 #ifndef ___Class_Rectangle
     5 #define ___Class_Rectangle
     6
     7 #include <string>
     8
```

```
 9 class Rectangle {
10 protected:                                    ← 違いはここだけ
11   static int numOfInstances;
                        // これまでに生成したインスタンスの個数
12   const int id;      // 長方形インスタンスに付けるid番号
13   double width;      // 長方形の横幅
14   double height;     // 長方形の高さ
15 public:
16   Rectangle(double width = 0.0, double height = 0.0)
17     : id(numOfInstances++), width(width),
                                        height(height) {}
18   Rectangle(const Rectangle& rectangle);
                                        //コピーコンストラクタ
19   //~Rectangle() {}
20   // 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
```

```
21    static int getNumOfInstances() {
                                        return numOfInstances; }
22                              //これまでに生成したRectangle
                                        //インスタンスの個数を返す
23    double getId() const { return id; }
24    double getWidth() const { return width; }
25    double getHeight() const { return height; }
26    void setWidth(double width) { this->width = width; }
27    void setHeight(double height) {
                                        this->height = height; }
28    double getArea() const { return width*height; }
                                            //長方形の面積を返す
29    std::string toString() const;
                        //内部の長方形情報をstringデータとして返す
30 };
31
32 #endif
```

```
[motoki@x205a]$  cat -n Rectangle_verProt.cpp
     1 /* 長方形オブジェクトのクラス Rectangle (実装部) */
     2 /* (protectedデータ版)                          */
     3
     4 #include <sstream>
     5 #include <string>
     6 #include "Rectangle_verProt.h"
     7 using namespace std;
     8
     9 // static変数の初期化 ---------------------------------
    10 int Rectangle::numOfInstances = 0;
    11
    12 // 各種コンストラクタ ---------------------------------
    13 Rectangle::Rectangle(const Rectangle& rectangle)
                                          // コピーコンストラクタ
    14           : id(numOfInstances++),
    15             width(rectangle.width),
```

```
                                        height(rectangle.height) {}
    16
    17 // Rectangle型オブジェクトを操作するための関数群 ---------
    18 // 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
    19 string Rectangle::toString() const
    20 {
    21   ostringstream os;
    22   os << "rectangle(id=" << id
    23      << ",width=" << width << ",height="
                                          << height << ")";
    24   return os.str();
    25 }
[motoki@x205a]$
```

これらの Rectangle_verProt.h, Rectangle_verProt.cpp を利用して
色付き長方形を表すオブジェクトのクラス ColoredRectangle を派生...

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

```
 1 /* 色付き長方形オブジェクトのクラス ColoredRectangle
                                              (仕様部) */
 2 /* (既定義の Rectangleクラス の拡張)
 3
 4 #ifndef ___Class_ColoredRectangle
 5 #define ___Class_ColoredRectangle
 6
 7 #include <string>
 8 #include "Rectangle_verProt.h"
 9
10 class ColoredRectangle : public Rectangle {
11 protected:
12   std::string color;
13 public:
14   ColoredRectangle(double width = 0.0,
                                    double height = 0.0,
```

```
15                          std::string color = "black")
16      : Rectangle(width, height), color(color) {}
17    ColoredRectangle(const ColoredRectangle& rectangle);
                                              //コピーコンストラクタ

18    // オブジェクト(もしくはクラス全体)の情報を提供する
                                              // ための関数群(追加)
19    std::string getColor() const { return color; }
20    void setColor(std::string color) {
                                          this->color = color; }
21    std::string toString() const; //内部の長方形情報を
                  オーバーライド     //stringデータとして返す
22 };
23
24 #endif
[motoki@x205a]$  cat -n ColoredRectangle_verProtBase.cpp
 1 /* 色付き長方形オブジェクトのクラス ColoredRectangle
```

```
                                                    (実装部) */
 2
 3 #include <sstream>
 4 #include <string>
 5 #include "ColoredRectangle_verProtBase.h"
 6 using namespace std;
 7
 8 // 各種コンストラクタ ---------------------------------
 9 ColoredRectangle::ColoredRectangle(const
       ColoredRectangle& rectangle) //コピーコンストラクタ
10                     : Rectangle(rectangle.width,
                  メンバ関数の呼出し不要 rectangle.height),
11                       color(rectangle.color) {}
12
13 // ColoredRectangle型オブジェクトを操作するための関数群 --
14 // 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
```

```
15 string ColoredRectangle::toString() const
16 {                                    オーバーライド
17   ostringstream ostr;
18   ostr << "rectangle(id=" << id  メンバ関数の呼出し不要
19        << ",width=" << width
                                  << ",height=" << height
20        << ",color=\"" << color << "\")";
21   return ostr.str();
22 }
[motoki@x205a]$
```

これに関して、

- クラス ColoredRectangle がここでの例の様に定義されていた場合も
  クラス自体に備わったメンバの仕様は例5.1のものと同じ
  $\Longrightarrow$ 例5.1と同様の使用が可能

```
[motoki@x205a]$ cat -n useColoredRectangle_verProtBase.cpp
     1 /* ColoredRectangle_verProtBase.h,                    */
```

```
 2 /* ColoredRectangle_verProtBase.cpp の利用例 */
 3
 4 #include <iostream>
 5 #include <string>
 6 #include "ColoredRectangle_verProtBase.h"
 7 using namespace std;
 8
 9 int main()
10 {
11   ColoredRectangle rect;
12
13   cout << "(1)" << rect.toString() << endl
14        << "  |rect.getId()=" << rect.getId()
15        <<       ", rect.getWidth()="
                              << rect.getWidth() << endl
16        << "  |rect.getHeight()=" << rect.getHeight()
17        <<       ", rect.getColor()=\""
```

```
                    << rect.getColor() << "\"" << endl
18        << "  |rect.getArea()=" << rect.getArea()
                                                << endl;
19  rect.setWidth(2.0);
20  rect.setHeight(3.0);
21  rect.setColor("blue");
22  cout << "(2)" << rect.toString() << endl
23       << "  |rect.getId()=" << rect.getId()
24       <<        ", rect.getWidth()="
                           << rect.getWidth() << endl
25       << "  |rect.getHeight()=" << rect.getHeight()
26       <<        ", rect.getColor()=\""
                     << rect.getColor() << "\"" << endl
27       << "  |rect.getArea()=" << rect.getArea()
                                                << endl;
28
29  //cout << "(3)" << (new ColoredRectangle(4.0, 5.0,
```

```
                                        // "red"))->toString()
        //      << endl;                // heap領域のメモリが
                                        // 開放されないままになる
    30  ColoredRectangle* ptrRect
                  = new ColoredRectangle(4.0, 5.0, "red");
    31  cout << "(3)" << ptrRect->toString() << endl;
    32  delete ptrRect;
    33 }
[motoki@x205a]$  g++ useColoredRectangle_verProtBase.cpp
   ColoredRectangle_verProtBase.cpp Rectangle_verProt.cpp
[motoki@x205a]$  ./a.out
(1)rectangle(id=0,width=0,height=0,color="black")
  |rect.getId()=0, rect.getWidth()=0
  |rect.getHeight()=0, rect.getColor()="black"
  |rect.getArea()=0
(2)rectangle(id=0,width=2,height=3,color="blue")
  |rect.getId()=0, rect.getWidth()=2
```

```
  |rect.getHeight()=3, rect.getColor()="blue"
  |rect.getArea()=6
(3)rectangle(id=1,width=4,height=5,color="red")
[motoki@x205a]$
```

5–2 既定義クラスの拡張、is-a関係—public派生

public派生の場合のインタフェース継承：public派生の場合、
基底クラスのオブジェクトに備わっていたインタフェースは
全て派生クラスのオブジェクトにもそのまま引き継がれる
⇨ インタフェースの拡張されたオブジェクトのクラスが定義される

データ型としてのクラス：
クラス定義... 然るべきインタフェースを備えたオブジェクト
を作り出すための型枠
クラスをデータ型と見る場合は、
クラス定義... そのデータ型に属するために必要なインタフェース
を規定したもの、と考えることができる。
すなわち、

. . . . . .

データ型としてのクラス：

クラスをデータ型と見る場合は、

クラス定義... そのデータ型に属するために必要なインタフェース
を規定したもの、と考えることができる。

すなわち、

定義されたクラスの型に属するオブジェクトの集合

def (そのクラスのインスタンスに備わったインタフェース
と同等のインタフェースを有するオブジェクトの集合)

と考える。

特にpublic派生の場合は、

派生クラスの型に属するオブジェクトの集合
⊆ 基底クラスの型に属するオブジェクトの集合

⇨ 「派生クラス型のオブジェクトは基底クラス型オブジェクトの一種」
「派生クラス型は基底クラス型のサブタイプ (亜種) である」

この言い方に従った場合 、「intデータの集合⊆ doubleデータの集合」であるので、「int型はdouble型のサブタイプ」

is-a関係, has-a関係：

一般に、オブジェクト指向ではクラス間の次の2つの関係に注目する。

- **is-a関係** ···

  クラス A,B の間に**is-a関係**

  ⇔ クラス A のインスタンスがクラス B のインスタンスの一種

  こういう場合には、C++言語では、
  クラス B を`public`派生してクラス A を定義するのが良い。

- **has-a関係** ···

  クラス A,B の間に**has-a関係**

  ⇔ クラス A のインスタンスが
  クラス B のインスタンスを一部分として持つ

  こういう場合には、
  クラス A のデータメンバとしてクラス B のインスタンスを保持(または参照)する領域を用意するのが妥当である。

クラス型間のサブタイプの関係をC++プログラム上にどう反映させるか

public派生の場合の
「派生クラス型オブジェクトの集合⊆ 基底クラス型オブジェクトの集合」
という関係をプログラム上に反映させるために、C++言語では、

> 基底クラス型へのポインタ型変数に
> public派生クラス型オブジェクトへのポインタを保持できる

様になっている。

但し、その際 、
変数への代入の前に
基底クラス型へのポインタ型に暗黙に型変換される。

例5. 3 (ColoredRectangle*型ポインタ値をRectangle*型変数に保持)

例題4.2, 例5.1で考えた

> 基底クラス Rectangle と
> (public)派生クラス ColoredRectangle

を用いて

> 基底クラス型へのポインタ型変数に
> public派生クラス型オブジェクトへのポインタを保持できる

ことを確認した例を次に示す。

```
[motoki@x205a]$ cat -n testStoringPtrToDerivObjInBaseVar.cpp
     1 /* 基底クラス型変数に派生クラスオブジェクトへのポインタ */
     2 /* を保持した時の動作確認                                 */
     3
     4 #include <iostream>
     5 #include "ColoredRectangle.h"
     6 using namespace std;
     7
```

```
 8 int main()
 9 {
10   ColoredRectangle* ptrToDerivedClassObj
                          = new ColoredRectangle(1.0, 2.0);
11   cout << ptrToDerivedClassObj->toString() <<endl;
12
13   Rectangle* ptrToBaseClassObj(ptrToDerivedClassObj);
14   cout << ptrToBaseClassObj->toString() <<endl;
15 }
```

派生クラスのメンバ関数

型変換

初期設定

基底クラスのメンバ関数

```
[motoki@x205a]$  g++ testStoringPtrToDerivObjInBaseVar.cpp
                        ColoredRectangle.cpp Rectangle.cpp
[motoki@x205a]$  ./a.out
rectangle(id=0,width=1,height=2,color="black")
rectangle(id=0,width=1,height=2)
[motoki@x205a]$
```

例5. 4 (private派生の場合)

private派生の場合に先の例5.3で確認した

> 基底クラス型へのポインタ型変数に
> public派生クラス型オブジェクトへのポインタを保持できる

ということがどうなるかを調べたい。
そのために、
例題4.2で示したRectangleクラスをprivate派生させ、
先の例5.3と同様のことを行なってみた結果を次に示す。

```
[motoki@x205a]$  cat -n ColoredRectangle_verPrivDeriv.h
     1 /* 色付き長方形オブジェクトのクラス ColoredRectangle
                                                (仕様部) */
     2 /* (既定義の Rectangleクラス をprivate派生した版)     */
     3
     4 #ifndef ___Class_ColoredRectangle
     5 #define ___Class_ColoredRectangle
     6
```

```
 7 #include <string>
 8 #include "Rectangle.h"
 9
10 class ColoredRectangle : private Rectangle {
11   std::string color;
12 public:
13   ColoredRectangle(double width = 0.0,
                                    double height = 0.0,
14                    std::string color = "black")
15     : Rectangle(width, height), color(color) {}
16   ColoredRectangle(const ColoredRectangle& rectangle);
                                    //コピーコンストラクタ

17   // オブジェクト(もしくはクラス全体)の情報を提供する
                                    // ための関数群(追加)
18   std::string getColor() const { return color; }
19   void setColor(std::string color) {
```

```
                                        this->color = color; }
    20   std::string toString() const; //内部の長方形情報を
                        オーバーライド    //stringデータとして返す
    21 };
    22
    23 #endif
[motoki@x205a]$  cat -n ColoredRectangle_verPrivDeriv.cpp
     1 /* 色付き長方形オブジェクトのクラス ColoredRectangle
                                                (実装部) */
     2 /* (既定義の Rectangleクラス をprivate派生した版)     */
     3
     4 #include <sstream>
     5 #include <string>
     6 #include "ColoredRectangle_verPrivDeriv.h"
     7 using namespace std;
     8
     9 // 各種コンストラクタ ---------------------------------
```

```
10 ColoredRectangle::ColoredRectangle(const
       ColoredRectangle& rectangle) //コピーコンストラクタ
11                     : Rectangle(rectangle.getWidth(),
                                   rectangle.getHeight()),
12                       color(rectangle.color) {}
13
14 // ColoredRectangle型オブジェクトを操作するための関数群 --
15 // 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
16 string ColoredRectangle::toString() const
17 {                                  オーバーライド
18   ostringstream ostr;
19   ostr << "rectangle(id=" << getId()
20        << ",width=" << getWidth() << ",height=" << getH
21        << ",color=\"" << color << "\")";
22   return ostr.str();
23 }
```

```
[motoki@x205a]$ cat -n testStoringPtrToDerivObjInBaseVar_whenPrivDeriv.cpp
     1 /* private派生した場合、                                          */
     2 /* 基底クラス型変数に派生クラスオブジェクトへのポインタ          */
     3 /* を保持した時の動作確認                                        */
     4
     5 #include <iostream>
     6 #include "ColoredRectangle_verPrivDeriv.h"
     7 using namespace std;
     8
     9 int main()
    10 {
```

```
11   ColoredRectangle* ptrToDerivedClassObj
                          = new ColoredRectangle(1.0, 2.0);
```

Rectangleオブジェクトとしてのインタフェースが無い

```
12   cout << ptrToDerivedClassObj->toString() <<endl;
13
14   Rectangle* ptrToBaseClassObj;
                          // <---この宣言自体はエラーにならない
15   ptrToBaseClassObj = ptrToDerivedClassObj;
16 }
```

型変換不可→コンパイルエラー

```
[motoki@x205a]$  g++
      testStoringPtrToDerivObjInBaseVar_whenPrivDeriv.cpp
          ColoredRectangle_verPrivDeriv.cpp Rectangle.cpp
testStoringPtrToDerivObjInBaseVar_whenPrivDeriv.cpp:
関数 ‘int main()’ 内:
testStoringPtrToDerivObjInBaseVar_whenPrivDeriv.cpp:15:21:
エラー:‘Rectangle’ is an inaccessible base of ‘ColoredRectangle
```

```
    ptrToBaseClassObj = ptrToDerivedClassObj;
                      ^

[motoki@x205a]$
```

5–3 仮想関数を用いた多態性の実現

多態性： 一般のプログラミング言語で 、
1つの変数や関数(または関数呼出し)が実行時の状況により色々に振舞える能力を多態性(多相性, ポリモーフィズム)と呼ぶ。

多態変数 ... 実行の状況に応じて色々なデータ型の値を持てる変数
多態引数 ... 〃 引数
(純)多態関数 ... 多態引数を持つ関数

関数実体(1つ)の内部で、実引数のデータ型に応じて適切な処理内容が選択される。

多重定義 ... 1つの関数名や演算記号に対して
引数のデータ型毎に別々の関数本体,演算処理が定義される

⇨ 多態関数を呼び出す側では
多態引数のデータ型に応じて呼出す関数を切り替える必要が無い。

例5. 5 (C言語における演算記号の多重定義)

+演算子
- 整数 の加算を行う時の処理
- 実数 の加算を行う時の処理

非オブジェクト指向言語における
代表的な多重定義・多態性の例

C++言語における多態性の実現：

- 関数の多重定義 $\cdots$ 既に**2.11**節で説明 $\longrightarrow$ 例**2.3**

- 演算子の多重定義 $\cdots$ 例えば、

  <<演算子
  - ストリーム出力 **if** 左側に`ostream`オブジェクト
  - 左シフト **if** 左側に`int`

  +演算子
  - 文字列の連接 **if** 左側に`string`オブジェクト
  - `int`型加算 **if** 左側に`int`
  - `double`型加算 **if** 左側に`double`

  関数の多重定義の一種として各ユーザが独自に導入できる。

  $\longrightarrow$ 例**4.3**, 例**4.4**

- 多態変数 $\cdots$ 前節5.2で説明された

  > 基底クラス型へのポインタ型変数に
  > public派生クラス型オブジェクトへのポインタを保持できる

  ということから判断して、

  基底クラス型へのポインタ型変数は多態変数

  と見做せないこともない。

  ただ、型変換されて保持するので、
  厳密な意味では多態変数とは言えない。
  (各変数のデータ型は固定 $\longrightarrow$ 多態変数はあり得ない。)

- 関数呼び出しの多態性 ··· やはり、前節5.2で説明された

  > 基底クラス型へのポインタ型変数に
  > public派生クラス型オブジェクトへのポインタを保持できる、

  ということに注目する。

  型変換は行われるが、
  ポインタの先には色々なサブタイプ(派生クラス)のオブジェクトが実体として存在している。

  C++言語では、
  このことを利用して、関数呼び出しの多態性を実現できる。

  具体的には、

  . . . . . .

具体的には、
基底クラスにおいてメンバ関数を定義する際に、例えば

```
virtual データ型 funcName( 仮引数列 );
```

と宣言されていた場合、

```
ptrToBaseClassObj-> funcName( 実引数列 )
```

というコードを書くと、この部分は

> ポインタ ptrToBaseClassObj の指している派生クラス
> に備わったメンバ関数 funcName() の動的な呼び出し

という形で多態的に処理される。

> すなわち、
> (型変換前の) 本来のクラスのメンバ関数 funcName()
> を特定した上で、それを実行してくれる。

virtual 指定された (メンバ) 関数... 仮想関数と呼ぶ。

基底クラスにおける virtual 指定は派生クラスに暗黙に継承
$\longrightarrow$ 派生クラスにおいての重ねての指定は不要

例5. 6 (仮想関数, 多態的なメンバ関数呼び出し)

キーワードvirtualの効果、すなわち

> 基底クラスBaseClass でメンバ関数にvirtual指定すると、
> BaseClass*型変数を通じたそのメンバ関数の呼び出しは、
> オブジェクトの元々のデータ型を認識した上で、
> 派生クラスのメンバ関数の動的な選択・実行をもたらす、

ということを確認したい。

そのために、

Rectangle::toString() にvirtual指定を付けた上で、
例5.3で考えた処理を再度実行。

まず、クラスRectangle, ColoredRectangle の定義は...

```
[motoki@x205a]$  cat -n Rectangle_verVirtualToString.h
     1 /* 長方形オブジェクトのクラス Rectangle (仕様部)    */
     2 /* (toString()をvirtual化して多態的に振舞わせる版) */
     3
     4 #ifndef ___Class_Rectangle
```

```
 5 #define ___Class_Rectangle
 6
 7 #include <string>
 8
 9 class Rectangle {
10   static int numOfInstances;
                        // これまでに生成したインスタンスの個数
11   const int id;      // 長方形インスタンスに付けるid番号
12   double width;      // 長方形の横幅
13   double height;     // 長方形の高さ
14 public:
15   Rectangle(double width = 0.0, double height = 0.0)
16     : id(numOfInstances++), width(width),
                                        height(height) {}
17   Rectangle(const Rectangle& rectangle);
                                      //コピーコンストラクタ
18   //~Rectangle() {}
```

```
19    // 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
20    static int getNumOfInstances() {
                                          return numOfInstances; }
21                                  //これまでに生成したRectangle
                                          //インスタンスの個数を返す
22    double getId() const { return id; }
23    double getWidth() const { return width; }
24    double getHeight() const { return height; }
25    void setWidth(double width) { this->width = width; }
26    void setHeight(double height) {
                                          this->height = height; }
27    double getArea() const { return width*height; }
                                              //長方形の面積を返す
28    virtual std::string toString() const;
        ↑     //内部保持の長方形情報をstringデータとして返す
29 }; 変更箇所はここだけ
```

```
    30
    31 #endif
[motoki@x205a]$  cat -n Rectangle_verVirtualToString.cpp
     1 /* 長方形オブジェクトのクラス Rectangle (実装部)    */
     2 /* (toString()をvirtual化して多態的に振舞わせる版) */
     3
     4 #include <sstream>
     5 #include <string>
     6 #include "Rectangle_verVirtualToString.h"      変更箇所
     7 using namespace std;
     8
     9 // static変数の初期化 ------------------------------------
    10 int Rectangle::numOfInstances = 0;
    11
    12 // 各種コンストラクタ ------------------------------------
    13 Rectangle::Rectangle(const Rectangle& rectangle)
                                          // コピーコンストラクタ
```

```
14              : id(numOfInstances++),
15                width(rectangle.width),
                                  height(rectangle.height) {}
16
17 // Rectangle型オブジェクトを操作するための関数群 ---------
18 // 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
19 string Rectangle::toString() const
20 {
21   ostringstream os;
22   os << "rectangle(id=" << id
23      << ",width=" << width << ",height="
                                      << height << ")";
24   return os.str();
25 }
```

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

```
 1 /* 色付き長方形オブジェクトのクラス ColoredRectangle
```

```
                                                    (仕様部) */
 2 /* (既定義(toString()をvirtual化)の Rectangleクラス
                                                      の拡張    */
 3
 4 #ifndef ___Class_ColoredRectangle
 5 #define ___Class_ColoredRectangle
 6
 7 #include <string>
 8 #include "Rectangle_verVirtualToString.h"
 9
10 class ColoredRectangle : public Rectangle {
11   std::string color;
12 public:
13   ColoredRectangle(double width = 0.0,
                                    double height = 0.0,
14                    std::string color = "black")
15     : Rectangle(width, height), color(color) {}
```

変更箇所

```
    16    ColoredRectangle(const ColoredRectangle& rectangle);
                                                  //コピーコンストラクタ
    17    // オブジェクト(もしくはクラス全体)の情報を提供する
                                                  // ための関数群(追加)
    18    std::string getColor() const { return color; }
    19    void setColor(std::string color) {
                                                this->color = color; }
    20    std::string toString() const; //内部の長方形情報を
                        オーバーライド    //stringデータとして返す
    21 };
    22
    23 #endif
[motoki@x205a]$  cat -n
                        ColoredRectangle_verVirtualToString.cpp
     1 /* 色付き長方形オブジェクトのクラス ColoredRectangle
                                                  (実装部) */
     2 /* (既定義(toString()をvirtual化)の Rectangleクラス
```

```
                                                の拡張     */
 3
 4 #include <sstream>
 5 #include <string>
 6 #include "ColoredRectangle_verVirtualToString.h"
 7 using namespace std;                          ↑ 変更箇所
 8
 9 // 各種コンストラクタ ---------------------------------
10 ColoredRectangle::ColoredRectangle(const
       ColoredRectangle& rectangle) //コピーコンストラクタ
11                     : Rectangle(rectangle.getWidth(),
                                   rectangle.getHeight()),
12                       color(rectangle.color) {}
13
14 // ColoredRectangle型オブジェクトを操作するための関数群 --
15 // 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
```

```
16 string ColoredRectangle::toString() const
17 {                                    オーバーライド
18   ostringstream ostr;
19   ostr << "rectangle(id=" << getId()
20        << ",width=" << getWidth()
                                  << ",height=" << getHeight()
21        << ",color=\"" << color << "\")";
22   return ostr.str();
23 }
[motoki@x205a]$
```

これに関して、

- 上の様に Rectangle, CploredRectangle が定義されていた場合は 、例5.3で考えたmain()の処理は次の様に進む。

```
[motoki@x205a]$  cat -n
                 testStoringPtrToDerivObjInVirtualBaseVar.cpp
   1 /* toString()をvirtual化した上で、                              */
```

```
 2 /* 基底クラス型変数に派生クラスオブジェクトへのポインタ */
 3 /* を保持した時の動作確認                                 */
 4
 5 #include <iostream>
 6 #include "ColoredRectangle_verVirtualToString.h"
 7 using namespace std;
 8
 9 int main()
10 {
11   ColoredRectangle* ptrToDerivedClassObj
                        = new ColoredRectangle(1.0, 2.0);
12   cout << ptrToDerivedClassObj->toString() <<endl;
13
14   Rectangle* ptrToBaseClassObj(ptrToDerivedClassObj);
15   cout << ptrToBaseClassObj->toString() <<endl;
16 }
```

派生クラスのメンバ関数

型変換

初期設定

派生クラスのメンバ関数←仮想関数

```
[motoki@x205a]$ g++
                testStoringPtrToDerivObjInVirtualBaseVar.cpp
                    ColoredRectangle_verVirtualToString.cpp
                        Rectangle_verVirtualToString.cpp
[motoki@x205a]$ ./a.out
rectangle(id=0,width=1,height=2,color="black")
rectangle(id=0,width=1,height=2,color="black")
[motoki@x205a]$
```

5–4 抽象クラス

基底クラス内の仮想(メンバ)関数については 、

```
virtual データ型 funcName( 仮引数列 ) = 0;
```

という形で宣言することによって、
関数本体の処理を未定義のままでクラス定義を終えることができる。

C++言語では、
本体処理が未定義のままの仮想関数... 純粋仮想関数,
純粋仮想(メンバ)関数を1個以上含むクラス... 抽象クラスと呼ぶ。

- 抽象クラスに属するオブジェクトを生成することは出来ない。

- 抽象クラス型へのポインタ変数を用意して、
  派生クラスのインスタンスへのポインタを保持させることはできる。

抽象クラスを使う利点：

- 統一感のあるプログラムの作成に繋がる。
- メンバ関数のオーバーライドのやり忘れをコンパイルの時点で防げる。
- 意図しないインスタンス生成をコンパイルの時点で防げる。
- 互いに類似した複数のクラスがある場合、
  - ◇ これらのクラスに共通する部分の詳細だけを記述し、
  - ◇ 個別対応が必要なメンバ関数については、純粋仮想関数にした

  抽象クラスを用意すれば、
  - ◇ 同じコードをあちこちの場所に書く必要が無くなり、
    コードの冗長さが無くなる。
  - ◇ 各派生クラスには派生クラス特有の処理だけを書けばよく、
    コードの見通しが良くなる。
  - ◇ 別の類似クラスが新たに必要になった場合も、
    そのクラスを簡単に定義できる様になる。

例5. 7 (2次元座標上の図形オブジェクトに共通の枠組みを定める抽象クラ

2次元座標上の円や長方形、三角形といった図形オブジェクトを多数扱う場合、これらに共通の枠組みとして抽象クラスを考え、個々の種類のオブジェクトのクラスをこの抽象クラスの派生クラスとして定義することにすれば、図形オブジェクト全体を統一的に扱うことができる様になる。

具体例として、

Shape2D ... 図形オブジェクト全体の抽象基底クラス,
Circle2D ... 円オブジェクトの派生クラス,
Rectangle2D ... 長方形オブジェクトの派生クラス,
Triangle2D ... 三角形オブジェクトの派生クラス

の定義例を次に示す。

```
[motoki@x205a]$  cat -n Shape2D.h
     1 /* 頂点の座標情報等を保持する2次元図形オブジェクト          *
     2 /* に共通の枠組みを定める抽象基底クラス Shape2D  (仕様部) *
     3
     4 #ifndef ___Class_Shape2D
```

```
 5 #define ___Class_Shape2D
 6
 7 #include <string>
 8
 9 class Shape2D {
10   static int numOfInstances;
                    // これまでに生成したインスタンスの個数
11 protected:
12   const int id;     // 図形インスタンスに付けるid番号
13   Shape2D(): id(numOfInstances++) {}
14 public:          ↑                          id番号の割り振り
15   int getId() const { return id; }純粋仮想(メンバ)関数
16   virtual std::string toString() const = 0; ←┘
              //内部保持の図形情報をstringデータとして返す
17   virtual double getArea() const = 0;
18 };                    ↑インスタンスの備えるべき関数名を規定
19          純粋仮想(メンバ)関数
```

```
    20 #endif
[motoki@x205a]$  cat -n Shape2D.cpp
     1 /* 頂点の座標情報等を保持する2次元図形オブジェクト          *
     2 /* に共通の枠組みを定める抽象基底クラス Shape2D  (実装部) *
     3
     4 #include "Shape2D.h"
     5
     6 // static変数の初期化 ------------------------------
     7 int Shape2D::numOfInstances = 0;
[motoki@x205a]$  cat -n Circle2D.h
     1 /* 中心座標と半径の情報を保持する2次元円オブジェクト */
     2 /* のクラス Circle2D                    (仕様部) */
     3
     4 #ifndef ___Class_Circle2D
     5 #define ___Class_Circle2D
     6
     7 #include <string>
```

```
 8 #include "Shape2D.h"
 9
10 const double PI = 3.1415926535897932;  //円周率
11
12 class Circle2D : public Shape2D {
13   double x;        //円の中心のx座標
14   double y;        //円の中心のy座標
15   double radius;   //円の半径
16 public:
17   Circle2D(double x=0.0, double y=0.0,
                                    double radius=1.0)
18     : Shape2D(), x(x), y(y), radius(radius) {}
19   std::string toString() const;
20   double getArea() const { return PI*radius*radius; }
21 };
22
23 #endif
```

```
[motoki@x205a]$  cat -n Circle2D.cpp
     1 /* 中心座標と半径の情報を保持する2次元円オブジェクト */
     2 /* のクラス Circle2D                              (実装部) */
     3
     4 #include <sstream>
     5 #include <string>
     6 #include "Circle2D.h"
     7 using namespace std;
     8
     9 // Circle2D型オブジェクトを操作するための関数群 ----------
    10 // オブジェクト内部に保持している円情報を
                                        // string型データとして返す
    11 string Circle2D::toString() const
    12 {
    13   ostringstream ostr;
    14   ostr << "circle[id=" << id
    15        << "] of center (" << x << "," << y
```

```
                                    << ") and radius " << radius;
    16   return ostr.str();
    17 }
[motoki@x205a]$  cat -n Rectangle2D.h
     1 /* 頂点の座標情報を保持する2次元長方形オブジェクト */
     2 /* のクラス Rectangle2D                          (仕様部) */
     3
     4 #ifndef ___Class_Rectangle2D
     5 #define ___Class_Rectangle2D
     6
     7 #include <string>
     8 #include <cmath>
     9 #include "Shape2D.h"
    10
    11 class Rectangle2D : public Shape2D {
    12   double x0, y0;      //長方形の1つの頂点のx座標,y座標
    13   double x1, y1;      //(x0,y0)と対角の位置にある頂点
```

```
                                                    //のx座標,y座標
    14 public:
    15   Rectangle2D(double x0=0.0, double y0=0.0,
                                  double x1=1.0, double y1=1.0)
    16     : Shape2D(), x0(x0), y0(y0), x1(x1), y1(y1) {}
    17   std::string toString() const;
    18   double getArea() const {
                                 return fabs((x1-x0)*(y1-y0)); }
    19 };
    20
    21 #endif
[motoki@x205a]$  cat -n Rectangle2D.cpp
     1 /* 頂点の座標情報を保持する2次元長方形オブジェクト */
     2 /* のクラス Rectangle2D                          (実装部) */
     3
     4 #include <sstream>
     5 #include <string>
```

```
 6 #include "Rectangle2D.h"
 7 using namespace std;
 8
 9 // Rectangle2D型オブジェクトを操作するための関数群 -------
10 // 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
11 string Rectangle2D::toString() const
12 {
13   ostringstream ostr;
14   ostr << "rectangle[id=" << id << "] of vertices ("
15        << x0 << "," << y0 << "), ("
16        << x1 << "," << y0 << "), ("
17        << x1 << "," << y1 << "), ("
18        << x0 << "," << y1 << ")";
19   return ostr.str();
20 }
[motoki@x205a]$  cat -n Triangle2D.h
```

```
 1 /* 頂点の座標情報を保持する2次元三角形オブジェクト */
 2 /* のクラス Triangle2D                          (仕様部) */
 3
 4 #ifndef ___Class_Triangle2D
 5 #define ___Class_Triangle2D
 6
 7 #include <string>
 8 #include "Shape2D.h"
 9
10 class Triangle2D : public Shape2D {
11   double x0, y0;     //三角形の1つ目の頂点のx座標,y座標
12   double x1, y1;     //三角形の2つ目の頂点のx座標,y座標
13   double x2, y2;     //三角形の3つ目の頂点のx座標,y座標
14 public:
15   Triangle2D(double x0=0.0, double y0=0.0,
16       double x1=1.0, double y1=0.0,
                              double x2=0.0, double y2=1.0)
```

```
    17      : Shape2D(), x0(x0), y0(y0), x1(x1), y1(y1),
                                              x2(x2), y2(y2) {}
    18    std::string toString() const;
    19    double getArea() const;
    20 };
    21
    22 #endif
[motoki@x205a]$  cat -n Triangle2D.cpp
     1 /* 頂点の座標情報を保持する2次元三角形オブジェクト */
     2 /* のクラス Triangle2D                          (実装部) */
     3
     4 #include <sstream>
     5 #include <string>
     6 #include <cmath>
     7 #include "Triangle2D.h"
     8 using namespace std;
     9
```

```
10 // Rectangle2D型オブジェクトを操作するための関数群 -------
11 // オブジェクト内部に保持している三角形情報を
                                        // string型データとして返す
12 string Triangle2D::toString() const
13 {
14   ostringstream ostr;
15   ostr << "triangle[id=" << id << "] of vertices ("
16        << x0 << "," << y0 << "), ("
17        << x1 << "," << y1 << "), ("
18        << x2 << "," << y2 << ")";
19   return ostr.str();
20 }
21
22 // オブジェクト内部に保持している2次元座標上の
                                    // 三角形の面積を計算して返す
23 double Triangle2D::getArea() const
24 {                                                    //ヘロンの公式
```

```
25   double sideLeng1 = sqrt((x0-x1)*(x0-x1)
                                 +(y0-y1)*(y0-y1));
26   double sideLeng2 = sqrt((x1-x2)*(x1-x2)
                                 +(y1-y2)*(y1-y2));
27   double sideLeng3 = sqrt((x2-x0)*(x2-x0)
                                 +(y2-y0)*(y2-y0));
28   double s = (sideLeng1+sideLeng2+sideLeng3)/2.0;
29   return sqrt(s*(s-sideLeng1)*(s-sideLeng2)
                                   *(s-sideLeng3));
30 }
[motoki@x205a]$
```

これに関して、

- 上の様に Shape2D, Circle2D, Rectangle2D, Triangle2D が定義されていれば、それら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 useDerivedClassesFromShape2D.cpp

```
 1 /* Circle2D.h, Circle2D.cpp,                  */
 2 /* Rectangle2D.h, Rectangle.cpp,              */
 3 /* Triangle2D.h, Triangle2D.cpp の利用例 */
 4
 5 #include <iostream>
 6 #include "Circle2D.h"
 7 #include "Rectangle2D.h"
 8 #include "Triangle2D.h"
 9 using namespace std;
10
11 int main()
12 {
13   Shape2D* fig = new Circle2D(1.0, 0.0, 2.0);  // fig
14   cout << "fig = " << fig->toString() << endl
15        << "  ==> fig->getArea() = "
                          << fig->getArea() << endl;
16   delete fig;
```

多態変数として振舞う

↓

Circle2Dクラスのメンバ関数

```
17
18    fig = new Rectangle2D(0.0, 0.0, 1.0, 2.0);
19    cout << "fig = " << fig->toString() << endl
20         << "  ==> fig->getArea() =
                                  " << fig->getArea() << endl;
21    delete fig;
22
23    fig = new Triangle2D(0.0, 0.0, 2.0, 0.0, 1.0, 1.0);
24    cout << "fig = " << fig->toString() << endl
25         << "  ==> fig->getArea() =
                                  " << fig->getArea() << endl;
26    delete fig;
27 }
```

Rectangle2Dクラスのメンバ関数

Triangle2Dクラスのメンバ関数

```
[motoki@x205a]$  g++ useDerivedClassesFromShape2D.cpp
                                                  Circle2D.cpp
                 Rectangle2D.cpp Triangle2D.cpp Shape2D.cpp
[motoki@x205a]$  ./a.out
```

```
fig = circle[id=0] of center (1,0) and radius 2
  ==> fig->getArea() = 12.5664
fig = rectangle[id=1] of vertices (0,0), (1,0), (1,2), (0,2)
  ==> fig->getArea() = 2
fig = triangle[id=2] of vertices (0,0), (2,0), (1,1)
  ==> fig->getArea() = 1
[motoki@x205a]$
```

5–5 Makefileを用いた分割コンパイル

C++言語では、通常、
定義するクラス毎に.hファイルと.cppファイルの2つを用意する
⟶大量の個数のソースファイルを把握した上でのプログラミング作業

こんな時は、
Makefileを利用することもできる。

5–5–1 heapsort vs. bubblesort vs. llistsort

例題5. 8 (整列化モジュールに共通の枠組みを定める抽象クラス)
プログラミングAI例題14.4(C言語) で行った、

> 3つの整列化モジュール btree-heapsort.c, bubblesort.c, llistsort.c の外部仕様 (i.e. 外向けに提供する外部関数の名前や使い方) の統一

に相当することをC++の抽象クラスの考え方の下で行ってみよ。

(考え方) 例5.7に倣って、
① 整列化モジュールに共通の枠組みとして抽象クラスを考え、
② 個々の整列化手法ごとに、
その手法で整列化するモジュールのクラスを派生クラスとして定義すればよい。

より具体的には、

- 抽象クラスの設計に関して：

  プログラミング AI 例題 14.4 では

  整列化モジュール内で定義する関数仕様を次の様に統一した。

  ◇ `void sort_method(char *method)` ··· 引数の char 型配列に 整列化手法の名前を表す文字列を入れる

  ◇ `void sort(int a[], int size)` ··· 指定された配列内のデータを各々の整列化手法で小さい順に並べ替える

  この 2 つに相当するインタフェースを目的とする抽象クラスに設定すれば良い。

  関数 `sort()` については、

  単に、抽象クラスの定義の中に、

  ```
  virtual void sort(int a[], const int size) const = 0;
  ```

  という純仮想メンバ関数の宣言を入れて、利用の際のインタフェース統一を図れば十分である。

関数 sort_method() については、
関数呼び出しの際に十分な容量を持ったchar型配列を与える必要

$\Longrightarrow$ ここでは sort_method() と同等のメンバ関数を導入するのは避け、代わりに、

```
virtual std::string toString() const = 0;
```

という形の純仮想メンバ関数を導入して、利用の際のインタフェース統一を図ることにする。

このメンバ関数には 、
それぞれのモジュール(オブジェクト)の説明として整列化手法等も答えてもらう。

- 派生クラスの記述に関して：

  3つの整列化手法についてはCプログラム

  `btree-heapsort.c`(プログラミングAI例題13.2),
  `bubblesort.c`　(プログラミングAI例題14.4),
  `llistsort.c`　(プログラミングAI例題14.4)

  に記述されている通り

  $\Longrightarrow$ 大部分の記述が、各々のCプログラム内で定義されている`sort()`関数を派生クラス内のメンバ関数として取り込むだけで終わる。

  線形リストの実装 $\longrightarrow$ 例題4.5

(プログラミング)　ここで関連するクラスとして、

SortModuleForIntArray ... 整列化モジュール全体の抽象クラス,

HeapsortIntArray ... **heapsort**モジュールの派生クラス,

BubblesortIntArray ... **bubblesort**モジュールの派生クラス,

LListsortIntArray ... 連結リストへの挿入に基づく整列化モジュールの派生クラス,

SortedLinkedListOfInt ... `int`データを各節点に保持する連結リストオブジェクトのクラス

```
[motoki@x205a]$ cat -n SortModuleForIntArray.h
     1 /* int配列内の要素を昇順に並べ替える機能を備えた整列化 */
     2 /* モジュールに共通の枠組みを定める抽象基底クラス
                                  SortModuleForIntArray (仕様部) */
     3
     4 #ifndef ___Class_SortModuleForIntArray
     5 #define ___Class_SortModuleForIntArray
     6
     7 #include <string>
     8
     9 class SortModuleForIntArray {                    抽象クラス
    10 public:
    11   // 整列化モジュールの説明(主に手法)を答える
    12   virtual std::string toString() const = 0;
    13   // 引数で与えられた配列内の要素を昇順に並べ替える
    14   virtual void sort(int a[], const int size) const = 0;
    15 };
```

```
    16
    17 #endif
[motoki@x205a]$  cat -n HeapsortIntArray.h
     1 /* int配列内の要素をheapsort手法で昇順に並べ替える機能 */
     2 /* を備えた整列化モジュールを作り出すためのクラス
                                        HeapsortIntArray (仕様部) */
     3
     4 #ifndef ___Class_HeapsortIntArray
     5 #define ___Class_HeapsortIntArray
     6
     7 #include <string>
     8 #include "SortModuleForIntArray.h"
     9
    10 class HeapsortIntArray
                              : public SortModuleForIntArray {
    11 public:
    12   // 整列化モジュールの説明(主に手法)を答える
```

```
13    std::string toString() const {
                                    return "Heapsort module"; }
14    // 引数で与えられた配列内の要素を昇順に並べ替える
15    void sort(int a[], const int size ) const;
16 private:
17    void heapify(int a[], const int treeSize, int hole,
                                  const int newElement) const;
18 };
19
20 #endif
[motoki@x205a]$  cat -n HeapsortIntArray.cpp
 1 /* int配列内の要素をheapsort手法で昇順に並べ替える機能 */
 2 /* を備えた整列化モジュールを作り出すためのクラス
                                    HeapsortIntArray (実装部) */
 3
 4 #include "HeapsortIntArray.h"
 5
```

```
 6 // HeapsortIntArray型オブジェクト内に用意された関数群 ====
 7
 8 /*-----------------------------------------------------
 9 /* 配列要素を小さい順に並べ替える (heapsort)
10 /*-----------------------------------------------------
11 /* (仮引数) a    : int型配列
12 /*           size : int型配列 a の大きさ
13 /* (関数値)  :  なし
14 /* (機能)  :  heapsortアルゴリズムを使って、配列要素
15 /*                a[0],a[1],a[2], ..., a[size-1]
16 /*           を値の小さい順に並べ替える。
17 /*-----------------------------------------------------
18 void HeapsortIntArray::sort(int a[], const int size) co
19 {
20   //下からheapを構築してゆく
21   for (int k=size/2-1; k>=0; --k)
22     heapify(a, size, k, a[k]);
```

```
23
24    //大きい順にheapから取り出してゆく
25    for (int k=size-1; k>=1; --k) {
26        int tmp = a[k];
27        a[k]    = a[0];
28        heapify(a, k, 0, tmp);
29    }
30 }
31
32 /*-----------------------------------------
33 /* 番号 hole の節点より下の部分がheapの条件を満たす時 */
34 /* 新要素を加えてhole以下の部分がheapの条件を満たす様...*/
35 /*-----------------------------------------
36 /* (仮引数) a     : int型配列
37 /*      tree_size : 2分木と見做す部分配列 a[0],a[1],...
                                              の大きさ*/
38 /*           hole : a[0]～a[tree_size]の表す2分木内の
```

```
                                                    節点番号*/
39 /*    new_element : 2分木の節点に振り分けていない値
40 /* (関数値)   :   なし
41 /* (機能) :   番号 hole の節点のデータ記憶域は空で、その分
42 /*              new_element という値がどの節点にも記録され *
43 /*              ていない、また、hole より下の部分がheapの   *
44 /*              条件を満たしている、という状況を想定する。 */
45 /*              この様な状況の時に、hole より下にあるデータ *
46 /*              を上に shift upする操作を繰り返し行い、適当 */
47 /*              な時点で空の節点に新しい要素new_elementを  */
48 /*              割り当てることにより、hole 以下の部分が全面
                                    的に heapの条件を満たす様にする。*/
49 /*-------------------------------------------------------
50 void HeapsortIntArray::heapify(int a[],
                                          const int tree_size,
51         int hole, const int new_element) const
52 {
```

```
    53    int  siftup_cand;       /* siftup candidate */
    54
    55    while ((siftup_cand = hole*2+1) < tree_size) {
    56      if (siftup_cand+1<tree_size    /*右の子も居て右の*/
    57          && a[siftup_cand]<a[siftup_cand+1]) /*子の方*/
    58        ++siftup_cand;                        /*が大きい場合は */
    59                                  /*右の子がsiftupの候補 */
    60      if ( new_element >= a[siftup_cand])
    61        break;  /*new_elementをholeの場所に入れれば良い */
    62
    63      a[hole] = a[siftup_cand];      /* sift up */
    64      hole    = siftup_cand;
    65    }
    66    a[hole] = new_element;
    67 }
[motoki@x205a]$  cat -n BubblesortIntArray.h
     1 /* int配列内の要素をbubblesort手法で昇順に並べ替える */
```

```
 2 /* 機能を備えた整列化モジュールを作り出すためのクラス
                                    BubblesortIntArray (仕様部) */
 3
 4 #ifndef ___Class_BubblesortIntArray
 5 #define ___Class_BubblesortIntArray
 6
 7 #include <string>
 8 #include "SortModuleForIntArray.h"
 9
10 class BubblesortIntArray
                              : public SortModuleForIntArray {
11 public:
12   // 整列化モジュールの説明(主に手法)を答える
13   std::string toString() const {
                                  return "Bubblesort module"; }
14   // 引数で与えられた配列内の要素を昇順に並べ替える
15   void sort(int a[], const int size ) const;
```

```
16 };
17
18 #endif
[motoki@x205a]$  cat -n BubblesortIntArray.cpp
 1 /* int配列内の要素をbubblesort手法で昇順に並べ替える */
 2 /* 機能を備えた整列化モジュールを作り出すためのクラス
                                BubblesortIntArray (実装部) */
 3
 4 #include "BubblesortIntArray.h"
 5
 6 // BubblesortIntArray型オブジェクト内に用意された関数群 ==
 7
 8 /*-------------------------------------------
 9 /* 配列要素を小さい順に並べ替える (bubblesort)
10 /*-----------------------------------
11 /* (仮引数) a     : int型配列
12 /*            size : int型配列 a の大きさ
```

```
 13 /* (関数値)   :   なし
 14 /* (機能)   :   bubblesortアルゴリズムを使って、配列要素
 15 /*                    a[0],a[1],a[2], ..., a[size-1]
 16 /*                を値の小さい順に並べ替える。
 17 /*-------------------------------------------------------
 18 void BubblesortIntArray::sort(int a[], const int size)
 19 {
 20   for (int i=0; i<size-1; ++i)
 21     for (int j=size-1; j > i; --j)
 22       if (a[j-1] > a[j]) {        /* a[j-1] と a[j]  */
 23         int temp = a[j-1];        /* の大小を調べて、  */
 24         a[j-1]   = a[j];          /* 逆順なら交換する。*/
 25         a[j]     = temp;
 26       }
 27 }
[motoki@x205a]$  cat -n LListsortIntArray.h
  1 /* (1)int配列内の要素を次々と連結リストの大小順を保つ位置 *
```

```
 2 /* に挿入していき、それが終わったら、(2)連結リストに保持 */
 3 /* されたものを順にint配列内に移す、という手法で昇順に並
 4 /* べ替える機能を備えた整列化モジュールを作り出すための
                                    クラス LListsortIntArray (仕様部)*/
 5
 6 #ifndef ___Class_LListsortIntArray
 7 #define ___Class_LListsortIntArray
 8
 9 #include <string>
10 #include "SortModuleForIntArray.h"
11 #include "SortedLinkedListOfInt.h"
12
13 class LListsortIntArray
                              : public SortModuleForIntArray {
14 public:
15   // 整列化モジュールの説明(主に手法)を答える
16   std::string toString() const
```

```
    17      { return "sort module that is based on insertion ir
                                                          ; }
    18    // 引数で与えられた配列内の要素を昇順に並べ替える
    19    void sort(int a[], const int size ) const;
    20 };
    21
    22 #endif
[motoki@x205a]$  cat -n LListsortIntArray.cpp
     1 /* (1)int配列内の要素を次々と連結リストの大小順を保つ位置 *
     2 /* に挿入していき、それが終わったら、(2)連結リストに保持 */
     3 /* されたものを順にint配列内に移す、という手法で昇順に並
     4 /* べ替える機能を備えた整列化モジュールを作り出すための
                                  クラス LListsortIntArray (実装部)*/
     5
     6 #include "LListsortIntArray.h"
     7
     8 // LListsortIntArray型オブジェクト内に用意された関数群 ===
```

```
 9
10 /*-----------------------------------------------------
11 /* 配列要素を小さい順に並べ... (線形リスト上での挿入繰返し) *
12 /*-----------------------------------------------------
13 /* (仮引数) a    : int型配列
14 /*           size : int型配列 a の大きさ
15 /* (関数値)  :  なし
16 /* (機能)  :  配列要素
17 /*                a[0],a[1],a[2], ..., a[size-1]
18 /*           線形リスト上に昇順に挿入していき、その結果 */
19 /*           を配列に戻すことによって小さい順に並べ替える *
20 /*-----------------------------------------------------
21 void LListsortIntArray::sort(int a[], const int size)
                                                    const
22 {
23   SortedLinkedListOfInt list;
24
```

```
    25   for (int i=0; i<size; ++i)
    26     list.insertNodeOf(a[i]);
    27
    28   list.moveAllNodeInfoTo(a);
    29 }
[motoki@x205a]$  cat -n SortedLinkedListOfInt.h
     1 /* 整数を要素にもつNode群を整数の小さい順に連結した、 */
     2 /* 連結リストのクラス SortedLinkedListOfInt (仕様部)  */
     3
     4 #ifndef ___Class_SortedLinkedListOfInt
     5 #define ___Class_SortedLinkedListOfInt
     6
     7 #include <cstddef>   // for NULL
     8
     9 struct NodeInt {
    10   const int num;
    11   NodeInt* next;
```

```
12    NodeInt(int num=0, NodeInt* next=NULL)
                                    : num(num), next(next) {}
13 };
14
15 class SortedLinkedListOfInt {
16    NodeInt* head;
17 public:
18    SortedLinkedListOfInt(): head(NULL) {}
19    ~SortedLinkedListOfInt() { removeAllNodes(); }
20    int getHeadNodeInfo() const;
                                    //先頭Node内の情報を返す
21    int getNumOfNodesOf(const int num) const;
                                    //引数指定のNode数を返す
22    void printAllNodeInfo() const;
                                    //全てのNode内の情報を出力
23    void moveAllNodeInfoTo(int a[]);
                    //全てのNode内のint値を引数指定の配列に移動
```

```
24    void insertNodeOf(const int num);      //新データ挿入

25    int removeHeadNode();                  //先頭Nodeの削除
26    int remove1stNodeOf(const int num);
                                  //引数指定の最初のNodeを削除
27    int removeAllNodesOf(const int num);
                                  //引数指定の全Nodeを削除
28    void removeAllNodes();                 //全Nodeを削除
29 };
30
31 #endif
```

```
[motoki@x205a]$  cat -n SortedLinkedListOfInt.cpp
 1 /* 整数を要素にもつNode群を整数の小さい順に連結した、 */
 2 /* 連結リストのクラス SortedLinkedListOfInt (実装部)  */
 3
 4 #include <iostream>
 5 #include <iomanip>
```

```
 6 #include <cstddef>   // for NULL
 7 #include "SortedLinkedListOfInt.h"
 8 using namespace std;
 9
10 // 連結リストの先頭Nodeのnum値を返す
11 int SortedLinkedListOfInt::getHeadNodeInfo() const {
12   return head->num;
13 }
14
15 // 連結リスト中で引数をnum要素とするNodeの個数を返す
16 int SortedLinkedListOfInt::getNumOfNodesOf(
                                    const int num) const {
17   NodeInt* ptr = head;
18   while (ptr != NULL && ptr->num < num)
19     ptr = ptr->next;
20   int count;
21   for (count=0; ptr != NULL && ptr->num == num;
```

```
                                                    ++count)
22      ptr = ptr->next;
23    return count;
24  }
25
26  // 連結リスト内に保持されている内容を表の形に出力
27  void SortedLinkedListOfInt::printAllNodeInfo() const {
28    cout << "番号       num" << endl
29         << "----  ----------" << endl;
30    int count = 0;
31    for (NodeInt* ptr=head; ptr != NULL;
                                          ptr = ptr->next) {
32      cout << setw(4) << ++count
33           << setw(12) << ptr->num << endl;
34    }
35  }
36
```

```
37 // 連結リスト内の全int値を引数指定の配列に移動
                                        // (連結リストは空にする)
38 void SortedLinkedListOfInt::moveAllNodeInfoTo(int a[])
39   for (int n=0; head != NULL; ++n) {
40     a[n] = head->num;
41     NodeInt* tmp = head;
42     head = head->next;
43     delete tmp;
44   }
45 }
46
47 // 引数要素をもつNodeを連結リスト内のnum値の小さい順を
                                        // 保つ位置に挿入
48 void SortedLinkedListOfInt::insertNodeOf(
                                        const int num) {
49   NodeInt** ptrptr = &head;
50   while ((*ptrptr)!=NULL && (*ptrptr)->num <= num)
```

```
51      ptrptr = &((*ptrptr)->next);
52    *ptrptr = new NodeInt(num, *ptrptr);
53 }
54
55 //連結リスト中の先頭Nodeを削除し、削除したNode数を返す
56 int SortedLinkedListOfInt::removeHeadNode() {
57    if (head == NULL)
58      return 0;
59    NodeInt* tmp = head;
60    head = head->next;
61    delete tmp;
62    return 1;
63 }
64
65 // 連結リスト中で引数をnum要素とする最初のNode削除し、
                                        // 削除したNode数を返す
66 int SortedLinkedListOfInt::remove1stNodeOf(
```

```
                                                    const int num) {
67    NodeInt** ptrptr = &head;
68    while (*ptrptr != NULL && (*ptrptr)->num < num)
69      ptrptr = &((*ptrptr)->next);
70    if (*ptrptr != NULL && (*ptrptr)->num == num) {
71      NodeInt* tmp = *ptrptr;
72      *ptrptr = (*ptrptr)->next;
73      delete tmp;
74      return 1;
75    }else {
76      return 0;
77    }
78 }
79
80 // 連結リスト中で引数をnum要素とするNodeを全て削除し、
                                          // 削除したNode数を返す
81 int SortedLinkedListOfInt::removeAllNodesOf(
```

```
                                                const int num) {
82    NodeInt** ptrptr = &head;
83    while (*ptrptr != NULL && (*ptrptr)->num < num)
84      ptrptr = &((*ptrptr)->next);
85    int count = 0;
86    for (count=0; *ptrptr != NULL
                        && (*ptrptr)->num == num; ++count) {
87      NodeInt* tmp = *ptrptr;
88      *ptrptr = (*ptrptr)->next;
89      delete tmp;
90    }
91    return count;
92 }
93
94 // 連結リスト中のNodeを全て削除
95 void SortedLinkedListOfInt::removeAllNodes() {
96    while (head != NULL) {
```

```
    97      NodeInt* tmp = head;
    98      head = head->next;
    99      delete tmp;
   100    }
   101 }
[motoki@x205a]$
```

これに関して、

- 上記のプログラミング・デバッグの際に利用した
  Makefile(ここで関連した部分のみ),
  main()関数を含むテスト実行用のコード、および
  コンパイル・実行の様子

  を次に示す。

```
[motoki@x205a]$  cat -n Makefile
     1 # Makefile for clocking or testing 3 sorting methods:
     2 # (1) Heapsort
     3 # (2) Bubblesort
```

```
 4 # (3) Sort by insertion over linked-list
 5
 6 CC     = g++
```

改行すると必要になるので追加

煩わしさを避けるため

```
    (途中省略)
36 SRCOBJS_TEST_HEAP_LIGHT  = \
                        testHeapsortIntArray_light.cpp \
37                      HeapsortIntArray.o
38 SRCOBJS_TEST_BUBBL_LIGHT = \
                      testBubblesortIntArray_light.cpp \
39                      BubblesortIntArray.o
40 SRCOBJS_TEST_LLIST_LIGHT = \
                      testLListsortIntArray_light.cpp \
41                      LListsortIntArray.o \
                                SortedLinkedListOfInt.o
    (途中省略)
63 test_heap_light: ${SRCOBJS_TEST_HEAP_LIGHT}
64 tab ${CC} -o test_heap_light \
```

```
                                        $SRCOBJS_TEST_HEAP_LIGHT
65 test_bubble_light: ${SRCOBJS_TEST_BUBBL_LIGHT}
66 tab ${CC} -o test_bubble_light \
                                      $SRCOBJS_TEST_BUBBL_LIGHT
67 test_llist_light: ${SRCOBJS_TEST_LLIST_LIGHT}
68 tab ${CC} -o test_llist_light \
                                      $SRCOBJS_TEST_LLIST_LIGHT
```

(途中省略)

```
77 #-----------------------------------------------------------
```

(途中省略)

```
85 HeapsortIntArray.o: HeapsortIntArray.cpp \
                                         HeapsortIntArray.h \
                                    SortModuleForIntArray.h
86 tab ${CC} -c HeapsortIntArray.cpp
87 BubblesortIntArray.o: BubblesortIntArray.cpp \
                                       BubblesortIntArray.h \
                                    SortModuleForIntArray.h
```

```
 88 tab ${CC} -c BubblesortIntArray.cpp
 89 LListsortIntArray.o: LListsortIntArray.cpp \
                                        LListsortIntArray.h \
        SortModuleForIntArray.h SortedLinkedListOfInt.h
 90 tab ${CC} -c LListsortIntArray.cpp
 91 SortedLinkedListOfInt.o: SortedLinkedListOfInt.cpp \
                                      SortedLinkedListOfInt.h
 92 tab ${CC} -c SortedLinkedListOfInt.cpp
 93
 94 #-------------------------------------------------------
 95 clean:
 96 tab for i in *.o clock_all clock_heap clock_bubble \
                                              clock_llist \
 97 tab      test_all test_heap test_bubble test_llist \
 98 tab      test_all_light test_heap_light \
 99 tab      test_bubble_light test_llist_light ; do \
100 tab   if [ -f $$i ] ; then rm $$i ; fi \
```

```
   101 tab done
[motoki@x205a]$  cat -n testHeapsortIntArray_light.cpp
     1 /* HeapsortIntArray.h, HeapsortIntArray.cpp の利用例 >
     2
     3 #include <iostream>
     4 #include "HeapsortIntArray.h"
     5 using namespace std;
     6
     7 int main()
     8 {
     9   int a[10] = {9, 8, 6, 7, 5, 3, 1, 2, 4, 0};
    10   SortModuleForIntArray* ptrSortModule
                                  = new HeapsortIntArray();
    11
    12   ptrSortModule->sort(a, 10);
    13   cout << "after sorting ("
              << ptrSortModule->toString() << ")" << endl;
```

```
    14   cout << "  a = {";
    15   for (int i=0; i<9; ++i)
    16     cout << a[i] << ", ";
    17   cout << a[9] << "}" << endl;
    18 }
[motoki@x205a]$  make test_heap_light
g++ -c HeapsortIntArray.cpp
g++ -o test_heap_light testHeapsortIntArray_light.cpp
                                                HeapsortIntArray.o
[motoki@x205a]$  ./test_heap_light
after sorting (Heapsort module)
  a = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
[motoki@x205a]$  cat -n testBubblesortIntArray_light.cpp
     1 /* BubblesortIntArray.h, BubblesortIntArray.cpp
                                                の利用例 */
     2
     3 #include <iostream>
```

```
 4 #include "BubblesortIntArray.h"
 5 using namespace std;
 6
 7 int main()
 8 {
 9   int a[10] = {9, 8, 6, 7, 5, 3, 1, 2, 4, 0};
10   SortModuleForIntArray* ptrSortModule
                              = new BubblesortIntArray();
11
12   ptrSortModule->sort(a, 10);
13   cout << "after sorting ("
          << ptrSortModule->toString() << ")" << endl;
14   cout << "  a = {";
15   for (int i=0; i<9; ++i)
16     cout << a[i] << ", ";
17   cout << a[9] << "}" << endl;
18 }
```

```
[motoki@x205a]$  make test_bubble_light
g++ -c BubblesortIntArray.cpp
g++ -o test_bubble_light testBubblesortIntArray_light.cpp
                                                BubblesortIntArray.o
[motoki@x205a]$  ./test_bubble_light
after sorting (Bubblesort module)
  a = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
[motoki@x205a]$  cat -n testLListsortIntArray_light.cpp
     1 /* LListsortIntArray.h, LListsortIntArray.cpp
                                                の利用例 */
     2
     3 #include <iostream>
     4 #include "LListsortIntArray.h"
     5 using namespace std;
     6
     7 int main()
     8 {
```

```
 9   int a[10] = {9, 8, 6, 7, 5, 3, 1, 2, 4, 0};
10   SortModuleForIntArray* ptrSortModule
                                = new LListsortIntArray();
11
12   ptrSortModule->sort(a, 10);
13   cout << "after sorting ("
          << ptrSortModule->toString() << ")" << endl;
14   cout << "  a = {";
15   for (int i=0; i<9; ++i)
16     cout << a[i] << ", ";
17   cout << a[9] << "}" << endl;
18 }
[motoki@x205a]$  make test_llist_light
g++ -c LListsortIntArray.cpp
g++ -c SortedLinkedListOfInt.cpp
g++ -o test_llist_light testLListsortIntArray_light.cpp
                LListsortIntArray.o SortedLinkedListOfInt.o
```

```
[motoki@x205a]$  ./test_llist_light
after sorting (sort module that is based on insertion in a l
  a = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
[motoki@x205a]$
```

例題5. 9 (整列化モジュールの動作テストを担当するモジュール)
例題5.8で定義した3つのクラス HeapsortIntArray, BubblesortIntArray, LListsortIntArray は共通の抽象基底クラスをもっている。従って、この基底クラス型へのポインタを通して多態的にメンバ関数の呼び出しを行うことにより、これらのインスタンス(整列化モジュール)を統合的に扱うことが出来る。 これを体験するために 、この種の整列化モジュールの提供する「int 配列内の要素を昇順に並べ替える機能」が正しく動作するかどうかをプログラミング AI 例題 13.2 の check-sort-program.c に倣ってテストする機能、すなわち

> ①0~999の間のランダムな整数を要素とする
> 大きさ100の配列を生成し、
> ②それに対して与えられた整列化モジュールを適用して
> 並べ替え作業を行い、
> ③その結果を出力する、

という風にテストする機能を備えたモジュールのクラスを定義してみよ。

(考え方)

共通の抽象基底クラスが SortModuleForIntArray

⇨ SortModuleForIntArray*型の変数を引数に持ち、
引数で与えられた整列化モジュールに対して
所定の動作テストを施すメンバ関数
を備えたモジュールのクラスを定義すれば良い。

このメンバ関数の施す動作テストの処理内容については、
Cのコード check-sort-program.c を C++風に書き直すだけ

(プログラミング)

```
[motoki@x205a]$ cat -n TesterForSortModuleIntArray.h
     1 /* SortModuleForIntArrayモジュールの提供する
     2 /* 「int配列内の要素を昇順に並べ替える機能」が正しく動作 */
     3 /* するかどうかをテストする機能を備えたモジュールを作り
                                        出すためのクラス (仕様部) */
     4
     5 #ifndef ___Class_TesterForSortModuleIntArray
     6 #define ___Class_TesterForSortModuleIntArray
     7
     8 #include <string>
     9 #include "SortModuleForIntArray.h"
    10
    11 class TesterForSortModuleIntArray {
    12   static const int SIZE;
                          // runOnRandomData()の動作パラメータ
    13   static const int WIDTH;
```

```
                                    // runOnRandomDataの動作パラメータ
14 public:
15   // オブジェクトの説明をstringデータとして返す
16   std::string toString() const
17     { return "Tester for module that is to sort int dat
                                                        ; }
18   // SIZE個のランダムなデータから成る配列に対して
19   // 引数で与えられた整列化モジュールを実行してみる
20   void runOnRandomData(const SortModuleForIntArray*
                                              sortModule) const;
21 private:
22   // 引数で与えられた配列の要素を順に全て出力
                                          // (1行にWIDTH個ずつ)
23   void prettyPrint(const int a[]) const;
24 };
25
26 #endif
```

```
[motoki@x205a]$  cat -n TesterForSortModuleIntArray.cpp
     1 /* SortModuleForIntArrayモジュールの提供する
     2 /* 「int配列内の要素を昇順に並べ替える機能」が正しく動作 */
     3 /* するかどうかをテストする機能を備えたモジュールを作り
                                        出すためのクラス (実装部) */
     4
     5 #include <iostream>
     6 #include <iomanip>
     7 #include <cstdlib>
     8 #include "TesterForSortModuleIntArray.h"
     9 using namespace std;
    10
    11 const int TesterForSortModuleIntArray::SIZE  = 100;
    12 const int TesterForSortModuleIntArray::WIDTH = 10;
    13
    14 // TesterForSortModuleIntArray型オブジェクト内に
                                        // 用意された関数群 =========
```

```
15
16 // SIZE個のランダムなデータから成る配列に対して
17 // 引数で与えられた整列化モジュールを実行してみる
18 void TesterForSortModuleIntArray::
19         runOnRandomData(const SortModuleForIntArray*
                                                sortModule) const
20 {
21   int a[SIZE], seed;
22
23   //擬似乱数の設定
24   cout << "擬似乱数の初期シード(int値): ";
25   cin >> seed;
26   if (!cin) {
27     cout << "入力ミス --> 乱数シードの(再)設定なし"
                                                  << endl;
28   }else {
29     srand(seed);
```

```
30      cout << "乱数シードを" << seed << "に設定" << endl;
31    }
32
33    //配列aの各々の要素に0~ 999の乱数値を設定
34    for (int i=0; i<SIZE; ++i)
35      a[i] = rand() % 1000;
36
37    //整列化前の配列の内容を表示
38    cout << endl << "before sorting:" << endl;
39    prettyPrint(a);
40
41    //整列化
42    sortModule->sort(a, SIZE);
43
44    //整列化後の配列の内容を表示
45    cout << endl << "after sorting("
                << sortModule->toString() << "):" << endl;
```

```
46   prettyPrint(a);
47 }
48
49 // 引数で与えられた配列の要素を順に全て出力
                                        // (1行にWIDTH個ずつ)
50 void TesterForSortModuleIntArray::prettyPrint(
                                        const int a[]) const
51 {
52   int NumOfEleInLine=0;
53
54   for (int i=0; i<SIZE; ++i) {
55     cout << setw(7) << a[i];
56     ++NumOfEleInLine;
57     if (NumOfEleInLine >= WIDTH) {
58       cout << endl;
59       NumOfEleInLine = 0;
60     }
```

```
    61    }
    62    if (NumOfEleInLine > 0)
    63       cout << endl;
    64 }
[motoki@x205a]$
```

これに関して、

- 上記のプログラミング・デバッグの際に利用した
  Makefile(ここで関連した部分のみ),
  main()関数を含むコード、および
  コンパイル・実行の様子

  を次に示す。

```
[motoki@x205a]$  cat -n Makefile
     1 # Makefile for clocking or testing 3 sorting methods:
     2 # (1) Heapsort
     3 # (2) Bubblesort
     4 # (3) Sort by insertion over linked-list
```

```
 5
 6 CC     = g++
      (途中省略)
22 SRCOBJS_TEST_ALL    = \
                        testSortModulesIntArray_random.cpp \
23           TesterForSortModuleIntArray.o \
24           HeapsortIntArray.o BubblesortIntArray.o \
25           LListsortIntArray.o SortedLinkedListOfInt.o
      (途中省略)
52 test_all: ${SRCOBJS_TEST_ALL}
53 tab ${CC} -o test_all ${SRCOBJS_TEST_ALL}
      (途中省略)
77 #-----------------------------------------------------
      (途中省略)
82 TesterForSortModuleIntArray.o: \
                      TesterForSortModuleIntArray.cpp \
                         TesterForSortModuleIntArray.h
```

```
83 tab ${CC} -c TesterForSortModuleIntArray.cpp
84
85 HeapsortIntArray.o: HeapsortIntArray.cpp \
                                    HeapsortIntArray.h \
                                  SortModuleForIntArray.h
86 tab ${CC} -c HeapsortIntArray.cpp
87 BubblesortIntArray.o: BubblesortIntArray.cpp \
                                   BubblesortIntArray.h \
                                  SortModuleForIntArray.h
88 tab ${CC} -c BubblesortIntArray.cpp
89 LListsortIntArray.o: LListsortIntArray.cpp \
                                    LListsortIntArray.h \
        SortModuleForIntArray.h SortedLinkedListOfInt.h
90 tab ${CC} -c LListsortIntArray.cpp
91 SortedLinkedListOfInt.o: SortedLinkedListOfInt.cpp \
                                  SortedLinkedListOfInt.h
92 tab ${CC} -c SortedLinkedListOfInt.cpp
```

```
 93
 94 #-----------------------------------------------------
 95 clean:
 96 tab for i in *.o clock_all clock_heap clock_bubble \
                                              clock_llist \
 97 tab      test_all test_heap test_bubble test_llist \
 98 tab      test_all_light test_heap_light \
 99 tab      test_bubble_light test_llist_light ; do \
100 tab   if [ -f $$i ] ; then rm $$i ; fi \
101 tab done
```

[motoki@x205a]$ cat -n testSortModulesIntArray_random.cpp

```
 1 /* int配列内の要素を昇順に並べ替える機能を備えた整列化
                                          モジュールとして */
 2 /*  ・HeapsortIntArrayオブジェクト,                    */
 3 /*  ・BubblesortIntArrayオブジェクト,                  */
 4 /*  ・LListsortIntArrayオブジェクト                    */
 5 /* の3つを考えこれらが正しく整列化動作をするかどうかを */
```

```
 6 /*       整列化モジュールをテストする機能を備えた
 7 /*       TesterForSortModuleIntArrayオブジェクト
 8 /* を用いてテストするC++プログラム
 9
10
11 #include <iostream>
12 #include "HeapsortIntArray.h"
13 #include "BubblesortIntArray.h"
14 #include "LListsortIntArray.h"
15 #include "TesterForSortModuleIntArray.h"
16 using namespace std;
17
18 int main()
19 {
20   TesterForSortModuleIntArray tester;
21
22   //HeapsortIntArrayオブジェクトの動作テスト
```

```
23    tester.runOnRandomData(new HeapsortIntArray());
```

HeapsortIntArrayクラスのsort(),...が利用される

```
24    cout << "---" << endl;
25
26    //BubblesortIntArrayオブジェクトの動作テスト
27    tester.runOnRandomData(new BubblesortIntArray());
```

BubblesortIntArrayクラスのsort(),...が利用される

```
28    cout << "---" << endl;
29
30    //LListsortIntArrayオブジェクトの動作テスト
31    tester.runOnRandomData(new LListsortIntArray());
32 }
```

LListsortIntArrayクラスのsort(),...が利用される

1つのソース内で3種のsort(),...を利用できる

↑

↓

```
[motoki@x205a]$  make test_all
g++ -c TesterForSortModuleIntArray.cpp
g++ -o test_all testSortModulesIntArray_random.cpp
        TesterForSortModuleIntArray.o HeapsortIntArray.o
                                      BubblesortIntArray.o
```

```
                    LListsortIntArray.o SortedLinkedListOfInt.o
[motoki@x205a]$  ./test_all
擬似乱数の初期シード(int値):  333
乱数シードを333に設定

before sorting:
    556    289    435    368    666    319    214    273
     64    943    869    956     50    298    112    218
    603    936    515    385    671    776    137    886
    718    913    204    153    281    870    473    495
    432    208    548    653    517    950    951    629
    630    476    893    498    861    917    626    998
    913    521    544    470     27    825    340    500
    105    104    397      6    110    914    308     61
     18    878    305    264    376    518    181    354
    985    782    857    881    252    236    706    945
```

```
after sorting(Heapsort module):
      4       5       6      18      27      50      61      64
    110     112     132     137     144     153     181     204
    218     236     252     264     273     281     289     298
    319     336     340     354     368     376     385     397
    470     473     476     495     498     500     515     517
    520     521     544     548     556     563     585     603
    629     630     631     649     653     666     671     672
    730     736     776     782     803     825     829     836
    869     870     878     881     886     893     895     913
    917     936     943     945     950     951     956     957
---
```

擬似乱数の初期シード(int値): 333
乱数シードを333に設定

```
before sorting:
    556     289     435     368     666     319     214     273
```

```
  64   943   869   956    50   298   112   218
 603   936   515   385   671   776   137   886
 718   913   204   153   281   870   473   495
 432   208   548   653   517   950   951   629
 630   476   893   498   861   917   626   998
 913   521   544   470    27   825   340   500
 105   104   397     6   110   914   308    61
  18   878   305   264   376   518   181   354
 985   782   857   881   252   236   706   945

after sorting(Bubblesort module):
   4     5     6    18    27    50    61    64
 110   112   132   137   144   153   181   204
 218   236   252   264   273   281   289   298
 319   336   340   354   368   376   385   397
 470   473   476   495   498   500   515   517
 520   521   544   548   556   563   585   603
```

```
   629   630   631   649   653   666   671   672
   730   736   776   782   803   825   829   836
   869   870   878   881   886   893   895   913
   917   936   943   945   950   951   956   957
---
擬似乱数の初期シード (int 値):  333
乱数シードを 333 に設定

before sorting:
   556   289   435   368   666   319   214   273
    64   943   869   956    50   298   112   218
   603   936   515   385   671   776   137   886
   718   913   204   153   281   870   473   495
   432   208   548   653   517   950   951   629
   630   476   893   498   861   917   626   998
   913   521   544   470    27   825   340   500
   105   104   397     6   110   914   308    61
```

```
    18    878    305    264    376    518    181    354
   985    782    857    881    252    236    706    945

after sorting(sort module that is based on insertion in a li
     4      5      6     18     27     50     61     64
   110    112    132    137    144    153    181    204
   218    236    252    264    273    281    289    298
   319    336    340    354    368    376    385    397
   470    473    476    495    498    500    515    517
   520    521    544    548    556    563    585    603
   629    630    631    649    653    666    671    672
   730    736    776    782    803    825    829    836
   869    870    878    881    886    893    895    913
   917    936    943    945    950    951    956    957
[motoki@x205a]$
```

例題5. 10 (時間計測モジュール, 整列化モジュールの動作速度を計測する
プログラミング AI 例題 14.3 では、時間計測のためのモジュール consumed_time.c を C 言語で実装した。これに相当するオブジェクトのクラスを C++で実装せよ。更に、例題 5.8 で (抽象クラス SortModuleForIntArray から派生させて) 定義した 3 つのクラスのインスタンス (整列化モジュール) に対して、「int 配列内の要素を昇順に並べ替える手順」の動作速度をプログラミング AI 例題 14.4 の clock-sort-5-10-etc.c に倣って計測する機能、すなわち

> 要素数が 5, 10, 25, 50, 100, 200 の場合に対して
> ①問題例のランダムな設定,
> ②整列化プログラムの実行
> を各々 800000回, 400000回, 160000回, 80000回, 40000回, 20000回 繰り返して1回あたりの計算時間を求め、その結果を出力する、

という機能を備えたモジュールのクラスを定義してみよ。

(考え方)

時間計測モジュールに関しては、

単に

C言語の consumed_time.c に相当するオブジェクトを
インスタンスとするクラス

を定義すれば良いだけである。

ただ、C++では、

◇定義する構造体名/クラス名には汎用性が求められ、また

◇オブジェクト自体にも名前が付く

ので、構造体名／関数名を次の様に変更する。

C言語		C++言語
データ型名 Second	⟶	構造体名 ProcessTime_realTime
関数 void start_timekeeper();	⟶	メンバ関数 void start();
関数 Second consumed_time();	⟶	メンバ関数 ProcessTime_realTime getLastIntervalTimes();

整列化モジュールの動作速度を計測するモジュールに関しては、
例題5.9と同様に考えて、

> SortModuleForIntArray型の変数を引数に持ち、
> 引数で与えられた整列化モジュールに対して所定の方法で動作速度の計測を行うメンバ関数

を備えたモジュールのクラスを定義すれば良い。

その際、
動作速度の計測を行う手順については、
$\longrightarrow$ C言語で同様の処理を行っている
clock-sort-5-10-etc.c (プログラミングAI例題14.4)
を参考にすれば良い。

(プログラミング)

StopWatch ··· 時間計測モジュールのクラス,

TimerForSortModuleIntArray ··· 整列化モジュールの動作速度を計測するモジュールのクラス

を次の様に定義した。

```
[motoki@x205a]$  cat -n StopWatch.h
     1 /* ストップウォッチ風に時間を測るためのオブジェクトの
                                        クラス StopWatch (仕様部) */
     2
     3 #ifndef ___Class_StopWatch
     4 #define ___Class_StopWatch
     5
     6 #include <string>
     7 #include <ctime>
     8
     9 struct ProcessTime_realTime {
    10   double processTime;
```

```
11    double realTime;
12 };
13
14 class StopWatch {
15    clock_t previousClock;   //前回のclock()値
16    time_t  previousTime;    //前回のtime(NULL)値
17    clock_t currentClock;    //現在のclock()値
18    time_t  currentTime;     //現在のtime(NULL)値
19 public:
20    StopWatch(): previousClock(-1), previousTime(-1.0),
21      currentClock(-1), currentTime(-1.0) {}
22    std::string toString() { return
                    "module for measuring consumed time"; }
23    //時間計測開始
24    void start() ;
25    //前回のマーク時点からの経過時間
                    // (プロセス時間と実時間の組,単位秒)を返す
```

```
    26   ProcessTime_realTime getLastIntervalTimes();
    27 };
    28
    29 #endif
[motoki@x205a]$  cat -n StopWatch.cpp
     1 /* ストップウォッチ風に時間を測るためのオブジェクト
                                        のクラス StopWatch (実装部) */
     2
     3 #include <cstddef>
     4 #include "StopWatch.h"
     5
     6 // StopWatch型オブジェクト内に用意された関数群 ==========
     7
     8 //時間計測開始
     9 void StopWatch::start()
    10 {
    11   previousClock = clock();
```

```
12   previousTime  = time(NULL);
13 }
14
15 //前回のマーク時点からの経過時間
                    // (プロセス時間と実時間の組,単位秒)を返す
16 ProcessTime_realTime StopWatch::getLastIntervalTimes()
17 {
18   ProcessTime_realTime intervalTimes;
19
20   currentClock = clock();
21   currentTime  = time(NULL);
22   intervalTimes.processTime
23     = static_cast<double>(currentClock
                          - previousClock) / CLOCKS_PER_SEC;
24   intervalTimes.realTime = difftime(currentTime,
                                       previousTime);
25   previousClock = currentClock;
```

```
    26    previousTime  = currentTime;
    27    return intervalTimes;
    28 }
[motoki@x205a]$  cat -n TimerForSortModuleIntArray.h
     1 /* SortModuleForIntArrayモジュールの提供する
     2 /* 「int配列内の要素を昇順に並べ替える機能」の
                                          動作速度を計測する機能 */
     3 /* を備えたモジュールを作り出すためのクラス　(仕様部) */
     4
     5 #ifndef ___Class_TimerForSortModuleIntArray
     6 #define ___Class_TimerForSortModuleIntArray
     7
     8 #include <string>
     9 #include "SortModuleForIntArray.h"
    10
```

```
11 //struct Problem {
12 //   const int size;
13 //   const int iterationNum;
14 //};
15
16 class TimerForSortModuleIntArray {
17 public:
18   // オブジェクトの説明をstringデータとして返す
19   std::string toString() const
20     { return "Timer for module that is to sort int data
                                                    ; }
21   // 要素数が 5, 10, 25, 50, 100, 200 の場合について、
22   // 引数で与えられた整列化モジュールの平均実行時間を計る
23   void clockingOnVariousProblems(const
                SortModuleForIntArray* sortModule) const;
24 private:
```

汎用性なし→メンバ関数の中で局所的に定義

```
    25   void setAnArrayRandom(int a[],
                               const int size) const;
    26 };
    27
    28 #endif
[motoki@x205a]$  cat -n TimerForSortModuleIntArray.cpp
     1 /* SortModuleForIntArrayモジュールの提供する
     2 /* 「int配列内の要素を昇順に並べ替える機能」の
                                   動作速度を計測する機能 */
     3 /* を備えたモジュールを作り出すためのクラス　(実装部) */
     4
     5 #include <iostream>
     6 #include <iomanip>
     7 #include <cstdlib>
     8 #include <ctime>
     9 #include "StopWatch.h"
    10 #include "TimerForSortModuleIntArray.h"
```

```
11 using namespace std;
12
13 // TesterForSortModuleIntArray型オブジェクト内に
                                        // 用意された関数群 =========
14
15 /*****************************************************
16 /* 要素数が 5, 10, 25, 50, 100, 200 の場合について、      */
17 /* 整列化プログラムの平均実行時間を計る                    *
18 /*-----------------------------------
19 /*    要素数が 5, 10, 25, 50, 100, 200 の場合について、そ */
20 /*    れぞれ 800000回, 400000回, 160000回, 80000回, 40000
21 /*    回,20000回 次の作業を繰り返して所用時間を計り、後で */
22 /*    1回当りの計算時間を割り出す。
23 /*        |配列上に要素数分だけランダムに整数を生成し、    */
24 /*        |その配列要素を別途用意された整列化プログラムを */
25 /*        |使って昇順に並べ替える。                        */
26 /*****************************************************
```

```
27 void TimerForSortModuleIntArray::
28         clockingOnVariousProblems(const
                SortModuleForIntArray* sortModule) const
29 {
30   const int MAX_SIZE = 200;
31   const int PROB_NUM = 6;
32   struct Problem {
33     const int size;
34     const int iterationNum;
35   };
36   const Problem prob[PROB_NUM] = {{5,800000},
                                    {10,400000}, {25,160000},
37                                      {50,80000},
                                   {100,40000}, {200,20000}};
38   int a[MAX_SIZE], seed;
39   StopWatch timer;
40   ProcessTime_realTime timeForSort[PROB_NUM],
```

```
                                                timeForInit[PROB_NUM];
41
42    cout << "Clocking the average execution time of the p
                                                        << endl
43         << "that sorts 5, 10, 25, 50, 100, or 200 elemer
                                                        << endl
44         << "  (*** " << sortModule->toString()
                                                  << " ***)" << endl
45         << "Input a random seed (0 - " << RAND_MAX
                                                      << "):  ";
46    cin >> seed;
47
48    //ソート以外の部分の計算時間を測定
49    srand(seed);
50    for (int k=0; k<PROB_NUM; ++k) {
51      timer.start();
52      for (int i=0; i<prob[k].iterationNum; ++i) { /* こ
```

```
53        setAnArrayRandom(a, prob[k].size); /* の部分の計
54      }                                    /* 算時間を参考のために測る。*/
55      timeForInit[k] = timer.getLastIntervalTimes();
56    }
57
58    //ソートプログラムの平均計算時間を測定
59    srand(seed);
60    for (int k=0; k<PROB_NUM; ++k) {
61      timer.start();
62      for (int i=0; i<prob[k].iterationNum; ++i) { /* こ
63        setAnArrayRandom(a, prob[k].size);   /* の部分の
64        sortModule->sort(a, prob[k].size);   /* 計算時間
65      }                                      /* を測る。 *
66      timeForSort[k] = timer.getLastIntervalTimes();
67                /* 次にsort部分だけの計算時間を割り出す。 */
68      timeForSort[k].processTime
                              -= timeForInit[k].processTime;
```

```
69      timeForSort[k].realTime
                                -= timeForInit[k].realTime;
70    }
71
72    // 測定結果の出力
73    cout << endl
74         << "          ** time for sort **     **time for in
                                                    << endl
75         << "size     process_t  real_time     process_t  r
                                                    << endl
76         << "           (m sec)    (m sec)       (m sec)
                                                    << endl
77         << "----     ---------  ---------     ---------  -
                                                   << endl;
78    for (int k=0; k<PROB_NUM; ++k)
79      cout << setw(4) << prob[k].size
                                << fixed << setprecision(5)
```

```
80            << "    "
81            << setw(9) << timeForSort[k].processTime
                            *1000.0/prob[k].iterationNum
82            << "  "
83            << setw(9) << timeForSort[k].realTime
                            *1000.0/prob[k].iterationNum
84            << "    "
85            << setw(9) << timeForInit[k].processTime
                            *1000.0/prob[k].iterationNum
86            << "  "
87            << setw(9) << timeForInit[k].realTime
                            *1000.0/prob[k].iterationNum
88            << endl;
89 }
90
91 /*-----------------------------------------------
92 /* 引数で与えられた配列の各要素をランダムに設定する
```

```
 93 /*-------------------------------------
 94 /* (仮引数) a     : int型配列
 95 /*           size : int型配列 a の大きさ
 96 /* (関数値)  :  なし
 97 /* (機能) :  配列要素 a[0]～a[size-1] に 0～999 の間の  *
 98 /*            乱数を設定する。
 99 /*-------------------------------------------------------
100 void TimerForSortModuleIntArray::
          setAnArrayRandom(int a[], const int size) const
101 {
102   for (int i=0; i<size; ++i)
103     a[i] = rand() % 1000;
104 }
[motoki@x205a]$
```

これに関して、

- 上記のプログラミング・デバッグの際に利用した
  Makefile(ここで関連した部分のみ),
  main()関数を含むコード、および
  コンパイル・実行の様子
  を次に示す。

```
[motoki@x205a]$  cat -n Makefile
     1 # Makefile for clocking or testing 3 sorting methods:
     2 # (1) Heapsort
     3 # (2) Bubblesort
     4 # (3) Sort by insertion over linked-list
     5
     6 CC     = g++
     7
     8 SRCOBJS_CLOCK_ALL    = clockSortModulesIntArray.cpp \
     9          TimerForSortModuleIntArray.o StopWatch.o \
    10          HeapsortIntArray.o BubblesortIntArray.o \
    11          LListsortIntArray.o SortedLinkedListOfInt.o
```

```
      (途中省略)
43 clock_all: ${SRCOBJS_CLOCK_ALL}
44 tab ${CC} -o clock_all ${SRCOBJS_CLOCK_ALL}
      (途中省略)
77 #-----------------------------------------------
78 TimerForSortModuleIntArray.o: \
                          TimerForSortModuleIntArray.cpp \
                             TimerForSortModuleIntArray.h
79 tab ${CC} -c TimerForSortModuleIntArray.cpp
80 StopWatch.o: StopWatch.cpp StopWatch.h
81 tab ${CC} -c StopWatch.cpp
      (途中省略)
85 HeapsortIntArray.o: HeapsortIntArray.cpp \
                                  HeapsortIntArray.h \
                               SortModuleForIntArray.h
86 tab ${CC} -c HeapsortIntArray.cpp
87 BubblesortIntArray.o: BubblesortIntArray.cpp \
```

```
                                                BubblesortIntArray.h \
                                               SortModuleForIntArray.h
88 tab ${CC} -c BubblesortIntArray.cpp
89 LListsortIntArray.o: LListsortIntArray.cpp \
                                                 LListsortIntArray.h \
       SortModuleForIntArray.h SortedLinkedListOfInt.h
90 tab ${CC} -c LListsortIntArray.cpp
91 SortedLinkedListOfInt.o: SortedLinkedListOfInt.cpp \
                                               SortedLinkedListOfInt.h
92 tab ${CC} -c SortedLinkedListOfInt.cpp
93
94 #-----------------------------------------------------------
95 clean:
96 tab for i in *.o clock_all clock_heap clock_bubble \
                                                       clock_llist \
97 tab       test_all test_heap test_bubble test_llist \
98 tab       test_all_light test_heap_light \
```

```
 99 tab       test_bubble_light test_llist_light ; do \
100 tab   if [ -f $$i ] ; then rm $$i ; fi \
101 tab done
[motoki@x205a]$  cat -n clockSortModulesIntArray.cpp
     1 /* int配列内の要素を昇順に並べ替える機能を備えた整列化
                                                モジュールとして */
     2 /*  ・HeapsortIntArrayオブジェクト,                    */
     3 /*  ・BubblesortIntArrayオブジェクト,                  */
     4 /*  ・LListsortIntArrayオブジェクト                    */
     5 /* の3つを考え、これら動作速度を                        */
     6 /*       整列化モジュールの動作速度を計測する機能を備え */
     7 /*       たTimerForSortModuleIntArrayオブジェクト   */
     8 /* を用いて調べるC++プログラム                          */
     9
    10 #include <iostream>
    11 #include "HeapsortIntArray.h"
    12 #include "BubblesortIntArray.h"
```

```
13 #include "LListsortIntArray.h"
14 #include "TimerForSortModuleIntArray.h"
15 using namespace std;
16
17 int main()
18 {
19   TimerForSortModuleIntArray timer;
20
21   //HeapsortIntArrayオブジェクトの動作速度を計測
22   timer.clockingOnVariousProblems(
                                    new HeapsortIntArray());
        HeapsortIntArrayクラスのsort(),...が利用される
23   cout << "---" << endl;                          ↑
24          1つのソース内で3種のsort(),...を利用できる
25   //BubblesortIntArrayオブジェクトの動作速度を計測↓
26   timer.clockingOnVariousProblems(                ↓
                                  new BubblesortIntArray());
```

BubblesortIntArrayクラスのsort(),...が利用される

```
27    cout << "---" << endl;
28
29    //LListsortIntArrayオブジェクトの動作速度を計測
30    timer.clockingOnVariousProblems(
                                    new LListsortIntArray());
31 }
```

LListsortIntArrayクラスのsort(),...が利用される

```
[motoki@x205a]$  make clock_all
g++ -c TimerForSortModuleIntArray.cpp
g++ -c StopWatch.cpp
g++ -o clock_all clockSortModulesIntArray.cpp
                                    TimerForSortModuleIntArray.o
        StopWatch.o HeapsortIntArray.o BubblesortIntArray.o
                LListsortIntArray.o SortedLinkedListOfInt.o
[motoki@x205a]$  ./clock_all
Clocking the average execution time of the program
that sorts 5, 10, 25, 50, 100, or 200 elements.
```

```
  (*** Heapsort module ***)
Input a random seed (0 - 2147483647):   333

        ** time for sort **     **time for initialize**
size     process_t  real_time     process_t  real_time
          (m sec)    (m sec)       (m sec)    (m sec)
----     ---------  ---------     ---------  ---------
   5       0.00007    0.00000       0.00008    0.00000
  10       0.00026    0.00250       0.00012    0.00000
  25       0.00104    0.00000       0.00029    0.00000
  50       0.00263    0.00000       0.00056    0.00000
 100       0.00634    0.00000       0.00111    0.00000
 200       0.01490    0.05000       0.00222    0.00000
---
Clocking the average execution time of the program
that sorts 5, 10, 25, 50, 100, or 200 elements.
  (*** Bubblesort module ***)
```

```
Input a random seed (0 - 2147483647):   333

        ** time for sort **    **time for initialize**
size    process_t  real_time    process_t  real_time
         (m sec)    (m sec)      (m sec)    (m sec)
----    ---------  ---------    ---------  ---------
   5      0.00005    0.00125      0.00007    0.00000
  10      0.00027    0.00000      0.00012    0.00000
  25      0.00162    0.00000      0.00029    0.00000
  50      0.00595    0.01250      0.00059    0.00000
 100      0.02169    0.00000      0.00111    0.00000
 200      0.07558    0.10000      0.00222    0.00000
---
Clocking the average execution time of the program
that sorts 5, 10, 25, 50, 100, or 200 elements.
  (*** sort module that is based on insertion in a linked li
Input a random seed (0 - 2147483647):   333
```

```
        ** time for sort **     **time for initialize**
size     process_t  real_time     process_t  real_time
          (m sec)    (m sec)       (m sec)    (m sec)
----     ---------  ---------     ---------  ---------
   5       0.00022    0.00000       0.00007    0.00000
  10       0.00055    0.00250       0.00012    0.00000
  25       0.00170    0.00000       0.00029    0.00000
  50       0.00446    0.00000       0.00056    0.00000
 100       0.01321    0.02500       0.00111    0.00000
 200       0.04305    0.05000       0.00223    0.00000
[motoki@x205a]$
```

5–5–2 predator-preyシミュレーション

例題5. 11 (predator-preyシミュレーション)　16×16のマス目が2次元トーラス状に繋がり、その中の各々のマス目は①狐が1匹いるか、②兎が1匹いるか、③草が1株生えているか、④何も生息していないか、の状態にある。そして、次の単位時間後には各マスの状態は、自マスも含めた周囲9マスの状態に基づいて次の様に変化する。

- 現在、狐がいる時 、

$$\text{次の状態} = \begin{cases} \text{空 (i.e. 生息なし)} & \textbf{if } (\text{周囲9マスの狐の数}) > 5 \\ \text{空 (i.e. 生息なし)} & \textbf{if } (\text{狐の年齢}) > 6 \\ \text{狐 (歳を重ねて継続)} & \textbf{otherwise} \end{cases}$$

- 現在、兎がいる時 、

$$\text{次の状態} = \begin{cases} \text{空 (i.e. 生息なし)} & \textbf{if } (\text{周囲9マスの狐の数}) \\ & \qquad \geq (\text{周囲9マスの兎の数}) \\ \text{空 (i.e. 生息なし)} & \textbf{if } (\text{兎の年齢}) > 3 \\ \text{兎 (歳を重ねて継続)} & \textbf{otherwise} \end{cases}$$

- 現在、草が生息している時 、

$$\text{次の状態} = \begin{cases} \text{空 (i.e. 生息なし)} & \text{if (周囲9マスの兎の数)} \\ & \quad \geq \text{(周囲9マスの草の数)} \\ \text{草 (継続)} & \text{otherwise} \end{cases}$$

- 現在、何も生息していない時 、

$$\text{次の状態} = \begin{cases} \text{狐 (0歳)} & \text{if (周囲9マスの狐の数)} > 1 \\ \text{兎 (0歳)} & \text{if (周囲9マスの狐の数)} \leq 1, \\ & \quad \text{(周囲9マスの兎の数)} > 1 \\ \text{草 (新規)} & \text{if (周囲9マスの狐の数)} \leq 1, \\ & \quad \text{(周囲9マスの兎の数)} \leq 1, \\ & \quad \text{(周囲9マスの草の数)} > 0 \\ \text{空 (i.e. 生息なし)} & \text{otherwise} \end{cases}$$

以上の状態遷移規則の下で各マスの状態が変化するとして、各時点での16×16のマス目の状態を観測するC++プログラムを作成せよ。

(考え方)　シミュレーションの各時点で登場する、狐、兎、草、空(i.e. 生息なし)、マス、16×16のマス目全体、のそれぞれをプログラム上でオブジェクトとして扱うために、次の7つのクラスを用意する。

- Cell　... マス目のクラスで、次の主要メンバをもつ。
  - ◇ LivingThing* life ... マス目に生息する生物等へのポインタ
  - ◇ Cell* ptrNeighborCell[3][3] ... 周囲9マスへのポインタを要素とする2次元配列
- LivingThing　... 狐、兎、草、空(i.e.生息なし)を表すオブジェクト群を統一的に扱うための抽象クラスで、次の主要メンバをもつ。
  - ◇ LifeType type ... 生命の種類 (LifeTypeは列挙型)
  - ◇ nextLife(Cell*) ... 引数で与えられた周囲9マスへのポインタ情報を基に、次の単位時間後にマスに入る生物オブジェクト等へのポインタを返す純仮想関数
- Fox　... LivingThingから派生させた狐のクラスで、次の追加主要メンバをもつ。
  - ◇ int age ... 年齢
- Rabbit　... LivingThingから派生させた兎のクラスで、次の追加主要

メンバをもつ。

◇ int age ... 年齢

- Grass ... LivingThingから派生させた草のクラス。
- NoLife ... LivingThingから派生させた空(i.e.生息なし)のクラス。
- PredatorPreyWorld ... 16×16のマス目全体を表すオブジェクトのクラスで、次の主要メンバをもつ。
  - ◇ Cell place[16][16] ... マス目を要素とする2次元配列
  - ◇ getNextLifeAtPlace(int, int) ... 引数で指定された座標のマス目に次の単位時間後に入る生物オブジェクト等へのポインタを返す関数
  - ◇ setLifeAtPlace(int, int , LivingThing*) ... 引数で指定された座標のマス目に引数の生物オブジェクト等へのポインタを組み込む関数
  - ◇ setLifeInformationEmpty() ... 全てのマス目内の生存生物情報を消去する関数
  - ◇ printConfiguration() ... 16×16のマス目全体の生物生息の分布を出力する関数

そして、これらのクラスを利用して、$\mathbf{16}\times\mathbf{16}$のマス目の状態を連続的に状態遷移させるシミュレーションを行うコードを書けば良い。

(7つのクラスの定義)　構成した7つのクラスCell、LivingThing、Fox、Rabbit、Grass、NoLife、PredatorPreyWorldのコードを次に示す。

```
[motoki@x205a]$  cat -n Cell.h
     1 /* Predator-prey-simulationにおいて                          */
     2 /* 仮想生物1個体の生存する場所のクラス Cell (仕様部) */
     3
     4 #ifndef ___Class_Cell
     5 #define ___Class_Cell
     6
     7 #include "LivingThing.h"
     8
```

```
 9 enum { SOUTH=0, WEST=0, NEUTRAL, NORTH, EAST=2 };
10
11 class Cell {
12 public:
13   LivingThing* life;                  //              north
14   Cell* ptrNeighborCell[3][3];        //       [2][0] [2][1] |
15                                       // west  [1][0]  自分   [
16                                       //       [0][0] [0][1] |
17                                       //              south
18   LivingThing* nextLife() {
                 return life->nextLife(ptrNeighborCell); }
19 };
20
21 #endif
[motoki@x205a]$  cat -n LivingThing.h
 1 /* 1つの場所(cell)内に存在する仮想生物に共通の枠組み  */
```

```
 2 /* を定める抽象基底クラス LivingThing (仕様部) */
 3
 4 #ifndef ___Class_LivingThing
 5 #define ___Class_LivingThing
 6
 7 //#include "Cell.h"     // #include "Cell.h" としても、
 8 class Cell;    //    LivingThing.h --(include)-> Cell.h
 9                //       --(include,回避)-> LivingThing.h
10                // という風にインクルードガードによって
11                // includeの連鎖は回避され、クラスLiving
12                // Thingが未定義の状態でクラスCellの定義
13                // を行おうとしてエラーになるだけである。
14                // そこで、ここでは、左の様に、Cellがク
                  // ラス名であることだけを(前方)宣言する。
15
16 enum LifeType { NO_LIFE, GRASS, RABBIT, FOX };
17 const int NUM_OF_POSSIBLE_LIFE_TYPES = 4;
```

```
18
19 class LivingThing {
20 protected:
21   LifeType type;
22   LivingThing(LifeType type = NO_LIFE): type(type) {}
23 public:
24   LifeType getType() const { return type; }
25   virtual LivingThing* nextLife(Cell*
                                   ptrNeighborCell[3][3]) = 0;
26 };
27
28 #endif
```

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

```
 1 /* Predator-prey-simulationにおいて、1つの場所(cell)
 2 /* 内に存在する「狐(仮想生物)」のクラス Fox (仕様部) */
 3
 4 #ifndef ___Class_Fox
```

```
 5 #define ___Class_Fox
 6
 7 #include "LivingThing.h"
 8 #include "Cell.h"
 9
10 const int FOX_LIFE_LENGTH = 6;
11
12 class Fox : public LivingThing {
13 protected:
14   int age;
15 public:
16   Fox(int age = 0): LivingThing(FOX), age(age) {}
17   LivingThing* nextLife(Cell* ptrNeighborCell[3][3]);
18 };
19
20 #endif
[motoki@x205a]$  cat -n Fox.cpp
```

```
 1 /* Predator-prey-simulationにおいて、1つの場所(cell) */
 2 /* 内に存在する「狐(仮想生物)」のクラス Fox (実装部) */
 3
 4 #include "Fox.h"
 5 #include "NoLife.h"
 6
 7 // Fox型オブジェクトを操作するための関数群 ---------------
 8 LivingThing* Fox::nextLife(Cell*
                                    ptrNeighborCell[3][3])
 9 {
10   int numOf[NUM_OF_POSSIBLE_LIFE_TYPES];
11
12   numOf[NO_LIFE] = numOf[GRASS] = numOf[RABBIT]
                                  = numOf[FOX] = 0;
13   for (int i=0; i<3; ++i) {
14     for (int j=0; j<3; ++j) {
15       ++numOf[ptrNeighborCell[i][j]->life->getType()];
```

```
16      }
17    }
18
19    if (numOf[FOX] > 5)                    // too many foxes
20      return (new NoLife());
21    else if (age > FOX_LIFE_LENGTH)        // natural death
22      return (new NoLife());
23    else
24      return (new Fox(age+1));
25  }
```

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

```
 1 /* Predator-prey-simulationにおいて、1つの場所(cell) */
 2 /* 内に存在する「兎(仮想生物)」のクラス Fox (仕様部) */
 3
 4 #ifndef ___Class_Rabbit
 5 #define ___Class_Rabbit
 6
```

```
 7 #include "LivingThing.h"
 8 #include "Cell.h"
 9
10 const int RABBIT_LIFE_LENGTH = 3;
11
12 class Rabbit : public LivingThing {
13 protected:
14   int age;
15 public:
16   Rabbit(int age=0): LivingThing(RABBIT), age(age) {}
17   LivingThing* nextLife(Cell* ptrNeighborCell[3][3]);
18 };
19
20 #endif
[motoki@x205a]$  cat -n Rabbit.cpp
 1 /* Predator-prey-simulationにおいて、1つの場所(cell) */
 2 /* 内に存在する「兎(仮想生物)」のクラス Fox (実装部) */
```

```
 3
 4 #include "Rabbit.h"
 5 #include "NoLife.h"
 6
 7 // Rabbit型オブジェクトを操作するための関数群 ------------
 8 LivingThing* Rabbit::nextLife(Cell*
                                  ptrNeighborCell[3][3])
 9 {
10   int numOf[NUM_OF_POSSIBLE_LIFE_TYPES];
11
12   numOf[NO_LIFE] = numOf[GRASS] = numOf[RABBIT]
                                    = numOf[FOX] = 0;
13   for (int i=0; i<3; ++i) {
14     for (int j=0; j<3; ++j) {
15       ++numOf[ptrNeighborCell[i][j]->life->getType()];
16     }
17   }
```

```
    18
    19    if (numOf[FOX] > numOf[RABBIT])      // eaten by fox
    20      return (new NoLife());
    21    else if (age > RABBIT_LIFE_LENGTH)   // natural death
    22      return (new NoLife());
    23    else
    24      return (new Rabbit(age+1));
    25  }
[motoki@x205a]$  cat -n Grass.h
     1  /* Predator-prey-simulationにおいて、1つの場所(cell) */
     2  /* 内に存在する「草(仮想植物)」のクラス Fox (仕様部) */
     3
     4  #ifndef ___Class_Grass
     5  #define ___Class_Grass
     6
     7  #include "LivingThing.h"
     8  #include "Cell.h"
```

```
 9
10 class Grass : public LivingThing {
11 public:
12   Grass(): LivingThing(GRASS) {}
13   LivingThing* nextLife(Cell* ptrNeighborCell[3][3]);
14 };
15
16 #endif
[motoki@x205a]$  cat -n Grass.cpp
 1 /* Predator-prey-simulationにおいて、1つの場所(cell) */
 2 /* 内に存在する「草(仮想植物)」のクラス Fox (実装部) */
 3
 4 #include "Grass.h"
 5 #include "NoLife.h"
 6
 7 // Grass型オブジェクトを操作するための関数群 ------------
 8 LivingThing* Grass::nextLife(Cell*
```

```
                                                ptrNeighborCell[3][3])
 9 {
10   int numOf[NUM_OF_POSSIBLE_LIFE_TYPES];
11
12   numOf[NO_LIFE] = numOf[GRASS] = numOf[RABBIT]
                                          = numOf[FOX] = 0;
13   for (int i=0; i<3; ++i) {
14     for (int j=0; j<3; ++j) {
15       ++numOf[ptrNeighborCell[i][j]->life->getType()];
16     }
17   }
18
19   if (numOf[RABBIT] >= numOf[GRASS]) //eaten by rabbit
20     return (new NoLife());
21   else
22     return (new Grass());
23 }
```

```
[motoki@x205a]$  cat -n NoLife.h
     1 /* Predator-prey-simulationにおいて、1つの場所(cell) */
     2 /* 内に存在する「生命なし(仮想生物)」のクラス NoLife
                                                (仕様部) */
     3
     4 #ifndef ___Class_NoLife
     5 #define ___Class_NoLife
     6
     7 #include "LivingThing.h"
     8 #include "Cell.h"
     9
    10 class NoLife : public LivingThing {
    11 public:
    12   NoLife(): LivingThing(NO_LIFE) {}
    13   LivingThing* nextLife(Cell* ptrNeighborCell[3][3]);
    14 };
    15
```

```
    16 #endif
[motoki@x205a]$  cat -n NoLife.cpp
     1 /* Predator-prey-simulationにおいて、1つの場所(cell) */
     2 /* 内に存在する「生命なし(仮想生物)」のクラス NoLife
                                                (実装部) */
     3
     4 #include "NoLife.h"
     5 #include "Fox.h"
     6 #include "Rabbit.h"
     7 #include "Grass.h"
     8
     9 // NoLife型オブジェクトを操作するための関数群 ------------
    10 LivingThing* NoLife::nextLife(Cell*
                                    ptrNeighborCell[3][3])
    11 {
    12   int numOf[NUM_OF_POSSIBLE_LIFE_TYPES];
    13
```

```
14    numOf[NO_LIFE] = numOf[GRASS] = numOf[RABBIT]
                                    = numOf[FOX] = 0;
15    for (int i=0; i<3; ++i) {
16      for (int j=0; j<3; ++j) {
17        ++numOf[ptrNeighborCell[i][j]->life->getType()];
18      }
19    }
20
21    if (numOf[FOX] > 1)
22      return (new Fox());
23    else if (numOf[RABBIT] > 1)
24      return (new Rabbit());
25    else if (numOf[GRASS] > 0)
26      return (new Grass());
27    else
28      return (new NoLife());
29  }
```

```
[motoki@x205a]$ cat -n PredatorPreyWorld.h
     1 /* Predator-prey-simulationにおいて仮想生物達の生存 */
     2 /* する空間(トーラス状に繋がった2次元グリッド空間)の
                                クラス PredatorPreyWorld (仕様部) */
     3
     4 #ifndef ___Class_PredatorPreyWorld
     5 #define ___Class_PredatorPreyWorld
     6
     7 #include "LivingThing.h"
     8 #include "Cell.h"
     9
    10 const int WORLD_SIZE = 16;
    11
    12 class PredatorPreyWorld {
    13   Cell place[WORLD_SIZE][WORLD_SIZE];
    14 public:
    15   PredatorPreyWorld();
```

```
    16   LivingThing* getNextLifeAtPlace(const int longitude,
                                         const int latitude);
    17   void setLifeAtPlace(const int longitude,
                             const int latitude, LivingThing* life);
    18   void setLifeInformationEmpty();
    19   void printConfiguration() const;
    20 };
    21
    22 #endif
[motoki@x205a]$  cat -n PredatorPreyWorld.cpp
     1 /* Predator-prey-simulationにおいて仮想生物達の生存 */
     2 /* する空間(トーラス状に繋がった2次元グリッド空間)の
                              クラス PredatorPreyWorld (実装部) */
     3
     4 #include <iostream>
     5 #include <cstddef>
     6 #include "PredatorPreyWorld.h"
```

```
 7 using namespace std;
 8
 9 // コンストラクタ ------------------------------------
10 PredatorPreyWorld::PredatorPreyWorld()
11 {
12   for (int i=0; i<WORLD_SIZE; ++i) {
13     for (int j=0; j<WORLD_SIZE; ++j) {
14       place[i][j].life = NULL;
15       for (int x=WEST; x<=EAST; ++x) {
16         for (int y=SOUTH; y<=NORTH; ++y) {
17           place[i][j].ptrNeighborCell[x][y]
18             = &place[(i+x-1+WORLD_SIZE) % WORLD_SIZE]
19                     [(j+y-1+WORLD_SIZE) % WORLD_SIZE];
20         }
21       }
22     }
23   }
```

```
24 }
25
26 // PredatorPreyWorld型オブジェクトを操作するための関数群 -
27
28 //World内の引数で指定された場所に次の時点で生存する
                                        //仮想生物を割り出す
29 LivingThing* PredatorPreyWorld::getNextLifeAtPlace(
                const int longitude, const int latitude)
30 {
31   return place[longitude][latitude].nextLife();
32 }
33
34 //World内の引数で指定された場所に
                          //指定された仮想生物オブジェクトを配置
35 void PredatorPreyWorld::setLifeAtPlace(
                const int longitude, const int latitude,
36                                      LivingThing* life)
```

```
37 {
38   place[longitude][latitude].life = life;
39 }
40
41 //World内の仮想生物オブジェクトを全て消去する
42 void PredatorPreyWorld::setLifeInformationEmpty()
43 {
44   for (int i=0; i<WORLD_SIZE; ++i) {
45     for (int j=0; j<WORLD_SIZE; ++j) {
46       delete place[i][j].life;
47       place[i][j].life = NULL;
48     }
49   }
50 }
51
52 //Worldの状態を出力
53 void PredatorPreyWorld::printConfiguration() const
```

```
54 {
55   char printName[NUM_OF_POSSIBLE_LIFE_TYPES]
                                  = { '_', ':', 'r', 'F' };
56                 それぞれ 空(i.e.生息なし), 草, 兎, 狐 を.
57   for (int i=WORLD_SIZE-1; i>=0; --i) {
58     for (int j=0; j<WORLD_SIZE; ++j) {
59       if (place[i][j].life != NULL)
60         cout << printName[place[i][j].life
                                      ->getType()];
61       else
62         cout << '?';
63     }
64     cout << endl;
65   }
66 }
[motoki@x205a]$
```

(7つのクラスを利用してシミュレーションを行うコード)

プログラミング・デバッグの際に利用した

Makefile、

simulateBirthDeathProcess.cpp

... main()関数を含み状態遷移のシミュレーションを行うコード、

setInitialWorld1.cpp

... 16×16のマス目の初期状態を設定する関数を含むコード、

および

コンパイル・実行の様子

を次に示す。

```
[motoki@x205a]$  cat -n Makefile
     1 # Makefile for Predator-Prey simulation:
     2
     3 CC     = g++
     4
     5 OBJS1  = simulateBirthDeathProcess.o \
                                        setInitialWorld1.o \
```

```
 6           Fox.o  Rabbit.o  Grass.o  NoLife.o \
 7           PredatorPreyWorld.o
 8
 9 OBJS2  = simulateBirthDeathProcess.o \
                                      setInitialWorld2.o \
10           Fox.o  Rabbit.o  Grass.o  NoLife.o \
11           PredatorPreyWorld.o
12
13 OBJS3  = simulateBirthDeathProcess.o \
                                      setInitialWorld3.o \
14           Fox.o  Rabbit.o  Grass.o  NoLife.o \
15           PredatorPreyWorld.o
16
17 HEADERS = LivingThing.h Fox.h Rabbit.h \
                                        Grass.h NoLife.h \
18            Cell.h PredatorPreyWorld.h
19
```

```
20 exec_sim1: ${OBJS1}
21 tab ${CC} -o exec_sim1 ${OBJS1}
22 exec_sim2: ${OBJS2}
23 tab ${CC} -o exec_sim2 ${OBJS2}
24 exec_sim3: ${OBJS3}
25 tab ${CC} -o exec_sim3 ${OBJS3}
26                                  実行ファイルを全て生成する規則
27 all: exec_sim1 exec_sim2 exec_sim3        ←
28
29 simulateBirthDeathProcess.o : \
                    simulateBirthDeathProcess.cpp ${HEADERS}
30 tab ${CC} -c simulateBirthDeathProcess.cpp
31 setInitialWorld1.o : setInitialWorld1.cpp ${HEADERS}
32 tab ${CC} -c setInitialWorld1.cpp
33 setInitialWorld2.o : setInitialWorld2.cpp ${HEADERS}
34 tab ${CC} -c setInitialWorld2.cpp
35 setInitialWorld3.o : setInitialWorld3.cpp ${HEADERS}
```

```
36 tab ${CC} -c setInitialWorld3.cpp
37 Fox.o : Fox.cpp Fox.h LivingThing.h Cell.h NoLife.h
38 tab ${CC} -c Fox.cpp
39 Rabbit.o : Rabbit.cpp Rabbit.h LivingThing.h \
                                        Cell.h NoLife.h
40 tab ${CC} -c Rabbit.cpp
41 Grass.o : Grass.cpp Grass.h LivingThing.h \
                                        Cell.h NoLife.h
42 tab ${CC} -c Grass.cpp
43 NoLife.o : NoLife.cpp NoLife.h Fox.h Rabbit.h \
                                Grass.h LivingThing.h Cell.h
44 tab ${CC} -c NoLife.cpp
45 PredatorPreyWorld.o: PredatorPreyWorld.cpp \
                                    PredatorPreyWorld.h \
                                     LivingThing.h Cell.h
46 tab ${CC} -c PredatorPreyWorld.cpp
47
```

```
48 clean:
49 [tab] for i in *.o exec_sim1 exec_sim2 exec_sim3 ; do \
50 [tab]   if [ -f $$i ] ; then rm $$i ; fi \
51 [tab] done
[motoki@x205a]$  cat -n simulateBirthDeathProcess.cpp
 1 /* トーラス状に繋がった2次元グリッド空間において、        */
 2 /* 仮想生物達(狐,兎,草)の出生･死滅の過程をシミュレート... */
 3
 4 #include <iostream>
 5 #include "LivingThing.h"
 6 #include "Fox.h"
 7 #include "Rabbit.h"
 8 #include "Grass.h"
 9 #include "NoLife.h"
10 #include "PredatorPreyWorld.h"
11 using namespace std;
12
```

```
13 int setInitialWorld(PredatorPreyWorld* ptrWorld);
14
15 const int MAX_TIME = 5;
16
17 int main()
18 {
19   PredatorPreyWorld world1, world2;
20   PredatorPreyWorld* ptrCurrentWorld=&world1,
                          *ptrNextWorld=&world2, *ptrTmp;
21   LivingThing* ptrLife;
22
23   // initialize current world
24   setInitialWorld(ptrCurrentWorld);
25
26   //出生・死滅の過程をシミュレート
27   for (int time=0; time<=MAX_TIME; ++time) {
28     //現在のWorldの状態を出力
```

```
29      cout << "---" << endl
30           << "time=" << time << endl;
31      ptrCurrentWorld->printConfiguration();
32      //次の時点のWorldの状態を割り出す
33      for (int i=0; i<WORLD_SIZE; ++i) {
34        for (int j=0; j<WORLD_SIZE; ++j) {
35          ptrLife = ptrCurrentWorld
                                ->getNextLifeAtPlace(i, j);
36          ptrNextWorld->setLifeAtPlace(i, j, ptrLife);
37        }
38      }
39      //時間を進める
40      ptrTmp          = ptrCurrentWorld;
41      ptrCurrentWorld = ptrNextWorld;
42      ptrNextWorld    = ptrTmp;
43      ptrNextWorld->setLifeInformationEmpty();
44    }
```

```
    45 }
[motoki@x205a]$  cat -n setInitialWorld1.cpp
     1 /* トーラス状に繋がった2次元グリッド空間において、          */
     2 /* 仮想生物達(狐,兎,草)の出生・死滅の過程をシミュレート  */
     3 /* する際の、空間内の仮想生物達の初期配置を設定する関数
                                        を保有するモジュール */
     4
     5 #include "LivingThing.h"
     6 #include "Fox.h"
     7 #include "Rabbit.h"
     8 #include "Grass.h"
     9 #include "NoLife.h"
    10 #include "PredatorPreyWorld.h"
    11
    12 int setInitialWorld(PredatorPreyWorld* ptrWorld)
    13 {
    14   for (int i=0; i<WORLD_SIZE; ++i) {
```

```
15      for (int j=0; j<WORLD_SIZE; ++j) {
16        switch ((i/4+j/4) % 4) {
17        case 0:
18          ptrWorld->setLifeAtPlace(i, j, new Fox());
19          break;
20        case 1:
21          ptrWorld->setLifeAtPlace(i, j, new Grass());
22          break;
23        case 2:
24          ptrWorld->setLifeAtPlace(i, j, new Rabbit());
25          break;
26        case 3:
27          ptrWorld->setLifeAtPlace(i, j, new NoLife());
28          break;
29        }
30      }
31    }
```

```
    32 }
[motoki@x205a]$  make exec_sim1
g++ -c simulateBirthDeathProcess.cpp
g++ -c setInitialWorld1.cpp
g++ -c Fox.cpp
g++ -c Rabbit.cpp
g++ -c Grass.cpp
g++ -c NoLife.cpp
g++ -c PredatorPreyWorld.cpp
g++ -o exec_sim1 simulateBirthDeathProcess.o
                                    setInitialWorld1.o Fox.o
            Rabbit.o  Grass.o  NoLife.o PredatorPreyWorld.o
[motoki@x205a]$  ./exec_sim1
---
time=0
____FFFF:::rrrr
____FFFF:::rrrr
```

```
____FFFF:::rrrr
____FFFF:::rrrr
rrrr____FFFF:::
rrrr____FFFF:::
rrrr____FFFF:::
rrrr____FFFF:::
::::rrrr____FFFF
::::rrrr____FFFF
::::rrrr____FFFF
::::rrrr____FFFF
FFFF:::rrrr____
FFFF:::rrrr____
FFFF:::rrrr____
FFFF:::rrrr____
___
time=1
FFFFF__F:::_rrrr
```

```
r__F____::::rrrr
r__F____::::rrrr
rrrFF__F::::rrrr
rrrrFFFFF__F:::_
rrrrr__F____::::
rrrrr__F____::::
rrrrrrrFF__F::::
:::_rrrrFFFFF__F
::::rrrrr__F____
::::rrrrr__F____
::::rrrrrrrFF__F
F__F:::_rrrrFFFF
____::::rrrrr__F
____::::rrrrr__F
F__F::::rrrrrrrF
---
time=2
```

```
FFFFF::F:::rrrrr
rFFFF__:::::rrrr
rrFFF__:::::rrrr
rrrFFFFF:::rrrr
rrrrFFFFF::F:::r
rrrrrFFFF__:::::
rrrrrrFFF__:::::
rrrrrrrFFFFF:::::
:::rrrrrFFFFF::F
::::rrrrrFFFF__:
::::rrrrrrFFF__:
::::rrrrrrrFFFFF
F::F:::rrrrrFFFF
F__:::::rrrrrFFF
F__:::::rrrrrrFF
FFFF:::rrrrrrrF
___
```

```
time=3
____F::F:::rrrrr
r____F::::::rrrr
rr___FF::::::rrrr
rrr____F::::rrrr
rrrr____F::F:::r
rrrrr____F::::::
rrrrrr___FF::::::
rrrrrrr____F::::
:::rrrrr____F::F
::::rrrrr____F::
::::rrrrrr___FF:
::::rrrrrrr____F
F::F:::rrrrr____
_F::::::rrrrr___
_FF::::::rrrrrr__
___F::::rrrrrrr_
```

```
---
time=4
r__FF::F:::rrrrr
rr__FF::::::rrrr
rrr_FFF::::rrrr
rrrr_FFF:::rrrr
rrrrr__FF::F:::r
rrrrrr__FF::::::
rrrrrrr_FFF::::
rrrrrrrr_FFF:::
:::rrrrrr__FF::F
::::rrrrrr__FF::
::::rrrrrrr_FFF:
::::rrrrrrrr_FFF
F::F:::rrrrrr__F
FF::::::rrrrrr__
FFF::::rrrrrrr_
```

```
_FFF:::rrrrrrrr
___
time=5
rFFFF::F:::r____
rrrF__::::::____
rrrFF__:::::____
rrrrFF_F::::____
____rFFFF::F:::r
____rrrF__::::::
____rrrFF__:::::
____rrrrFF_F::::
:::r____rFFFF::F
::::____rrrF__::
::::____rrrFF__:
::::____rrrrFF_F
F::F:::r____rFFF
__::::::____rrrF
```

```
F__:::::____rrrF
FF_F::::____rrrr
[motoki@x205a]$
```