



C++ - モジュール 02

アドホック・ポリモーフィズム、演算子オーバーローディング、オーソドックスなカ
ノニカル・クラス形式

概要

このドキュメントには、C++ モジュールのモジュール02の練習問題が含まれています。

バージョン: 8

内容

I	はじめに	2
II	一般規定	3
III	新ルール	5
IV	練習00: オーソドックスなカノン形式での初めての授業	6
V	練習問題01: より有用な固定小数点数クラスを目指して	8
VI	練習02: 今、私たちは話している	10
VII	練習03: BSP	12
VIII	提出と相互評価	14

第一章 はじめ

に

C++は、*Bjarne Stroustrup*（ビャルネ・ストルストラップ）により、C言語の発展形として開発された汎用プログラミング言語である（出典：[Wikipedia](#)）。

これらのモジュールの目的は、**オブジェクト指向プログラミング**を紹介することです。これはあなたのC++の旅の出発点となるでしょう。OOPを学ぶには多くの言語が推奨されています。これは複雑な言語であり、物事をシンプルに保つために、あなたのコードはC++98標準に準拠することになります。

私たちは、現代のC++が多くの面で大きく異なっていることを認識しています。ですから、もしあなたが熟達したC++開発者になりたいのであれば、42のコン・コアの後にさらに進むのはあなた次第です！

第II章 一般規定

コンパイル

- c++と-Wall -Wextra -Werrorフラグでコードをコンパイルする。
- フラグ-std=c++98を追加しても、コードはコンパイルできるはずだ。

フォーマットと命名規則

- 演習用のディレクトリは次のように命名されます： ex00, ex01, ... , exn. exn
- ファイル名、クラス名、関数名、メンバ関数名、属性名は、ガイドラインの要求に従ってください。
- クラス名はUpperCamelCase形式で記述する。クラス・コードを含むファイルは常にクラス名に従って命名されます。例えば例えば、
ClassName.hpp/ClassName.h、ClassName.cpp、ClassName.hpp。例えば、
ClassName.hpp/ClassName.hpp/ClassName.cpp/ClassName.hppのようになります。
- 特に指定がない限り、すべての出力メッセージは改行文字で終了し、標準出力に表示されなければならない。
- さようならノルミネットC++モジュールでは、コーディング・スタイルは強制されません。あなたの好きなスタイルに従ってください。しかし、相互評価者が理解できないコードは、評価できないコードであることを心に留めておいてください。きれいで読みやすいコードを書くよう、ベストを尽くしてください。

許可/禁止

あなたはもうC言語でコーディングしていない。C++の出番だ！ だから

- 標準ライブラリのほとんどすべてを使うことが許されている。したがって、すでに知っていることに固執するのではなく、使い慣れたC関数のC++っぽいバージョンをできるだけ使うのが賢いやり方だろう。
- ただし、それ以外の外部ライブラリーは使用できない。つまり、C++11（および派生形式）とBoostライブラリは禁止されている。以下の関数も禁止されている：*printf()、*alloc()、free()です。これらを使用した場合、あなたの成績は0点となり、それで終わりです。

- 明示的に指定されていない限り、using 名前空間 `<ns_name>` と友達キーワードは禁止。さもないと、あなたの成績は-42点となる。
- **モジュール 08 と 09 でのみ STL を使用できます。**つまり、それまでは**コンテナ**（vector/list/map/など）や**アルゴリズム**（`<algorithm>`ヘッダーを含む必要があるもの）を使用しないこと。そうでなければ、成績は-42点となる。

いくつかの設計要件

- メモリ・リークはC++でも発生する。メモリを確保するときに（new キーワード）、**メモリー・リーク**を避けなければならない。
- モジュール02からモジュール09まで、**特に明記されている場合を除き**、あなたのクラスは**正教会正書式**でデザインされなければなりません。
- ヘッダー・ファイルに書かれた関数の実装は（関数テンプレートを除いて）、練習にとっては0を意味する。
- それぞれのヘッダーは、他のヘッダーから独立して使えるようにしなければならない。したがって、必要な依存関係をすべてインクルードしなければならない。しかし、**インクルード・ガード**を追加することで、二重インクルードの問題を避けなければなりません。そうでなければ、あなたの成績は0点になってしまいます。

続きを読む

- 必要であれば（コードを分割するためなど）ファイルを追加しても構いません。これらの課題はプログラムによって検証されるわけではないので、必須ファイルを提出する限り、自由に行ってください。
- 時には、練習のガイドラインが短く見えても、例題を見れば、指示には明示的に書かれていない要件がわかることもある。
- 始める前に各モジュールを完全に読むこと！ 本当にそうしてください。

- オーディンによって、ソーによって！頭を使え！



たくさんのクラスを実装しなければならない。これは、お気に入りのテキストエディタでスクリプトを書くことができない限り、退屈に思えるかもしれない。



練習をこなすにはある程度の自由が与えられている。しかし、義務的なルールは守り、怠けてはいけません。多くの有益な情報を見逃すことになります！理論的な概念を読むことをためらわないでください。

第III章 新規則


これからは、特に断りのない限り、すべてのクラスは**オーソドックスな正準形式**で設計されなければならない。そして、以下の4つの必須メンバー関数を実装しなければならない：

- デフォルトコンストラクタ
- コピーコンストラクタ
- コピー代入演算子
- デストラクタ

クラス・コードを2つのファイルに分割します。ヘッダーファイル（.hpp/.h）にはクラスの定義が含まれ、ソースファイル（.cpp）には実装が含まれます。

第四章

練習00：オーソドックスなカノン形式での初めての授業

	エクササイズ：00
オーソドックスなカノン形式での初めての授業	
ターンイン・ディレクトリ：ex00/	
提出ファイル：Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp	
禁止されている関数：なし	

整数と浮動小数点数を知っているつもりになっている。なんて可愛いんだ。

この3ページの記事（[1](#)、[2](#)、[3](#)）を読んで、そうでないことを発見してほしい。さあ、読んでください。

今日まで、あなたがコードで使う数値は、基本的に整数か浮動小数点数、あるいはそれらの変形（short、char、long、doubleなど）のいずれかだった。上の記事を読めば、整数と浮動小数点数は正反対の性質を持っていると考えて間違いないだろう。

しかし今日、状況は一変する。**固定小数点数**である！ほとんどの言語のスカラー型にはない固定小数点数は、性能、精度、範囲、精度の間で貴重なバランスを提供します。固定小数点数がコンピュータ・グラフィックス、サウンド処理、科学的プログラミングなどに特に適しているのはそのためです。

C++には固定小数点数がないので、足し算をすることになる。バークレー校

の[この記事](#)は良いスタートだ。バークレー大学がどんな大学か知らない人は、ウィキペディアの[このセクション](#)を読んでください。

オーソドックスな正準形式で、固定小数点数を表すクラスを作る：

- プライベートメンバー
 - 固定小数点数の値を格納する**整数**。
 - 小数ビット数を格納する**静的定数**。その値は常に整数リテラル 8 となります。
- 一般会員：
 - 固定小数点数の値を 0 に初期化するデフォルトのコンストラクタ。
 - コピーコンストラクタ。
 - コピー代入演算子のオーバーロード。
 - デストラクタ。
 - メンバ関数 `int getRawBits(void) const`;
これは固定小数点値の生の値を返す。
 - メンバ関数 `void setRawBits(int const raw)`;
これは固定小数点数の生の値を設定する。

このコードを実行する：

```
#インクルード
<iostream>
int main( void ) {
    // aを固定し、b
    // (a) を固定し
    // 、cを固定する
    ;

    c = bである;

    std::cout << a.getRawBits() <<
    std::endl; std::cout << b.getRawBits()
} << std::endl; std::cout <<
c.getRawBits() << std::endl;
```


0を返す;

のようなものが出力されるはずだ：

```
$> ./a.out
というデフォルトコンストラクタ。
コピー代入演算子が呼ばれた // <-- この行は、実装によっては欠けているかもしれない getRawBitsメンバ関数が呼ばれた
と呼ばれるデフォルトのコンストラクタ
コピー代入演算子 getRawBits メンバ関数 getRawBits メンバ関数 0
getRawBitsメンバ関数の呼び出し 0
getRawBitsメンバ関数の呼び出し 0
と呼ばれるデストラクタ。
$>
```

第五章

練習01： より有用な固定小数点数 クラスを目指して

	エクササイズ01
より有用な固定小数点数クラスを目指して	
ターンイン・ディレクトリ : <i>ex01/</i>	
提出ファイル: <i>Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp</i>	
使用できる関数: <i>roundf</i> (<i><cmath></i> から)	

前回の練習は良いスタートだったが、このクラスはかなり使い物にならない。0.0という値しか表すことができないのだ。

以下のパブリック・コンストラクタとパブリック・メンバ関数をクラスに追加します：

- **定数 integer** をパラメータとするコンストラクタ。
これは対応する固定小数点値に変換する。小数ビットの値は、練習問題00のように8で初期化される。
- **定数浮動小数点数**をパラメータとするコンストラクタ。
これは対応する固定小数点値に変換する。小数ビットの値は、練習問題00のように8で初期化される。
- メンバ関数 `float toFloat(void) const;`
固定小数点値を浮動小数点値に変換する。

- メンバ関数 `int toInt(void) const;`

固定小数点値を整数値に変換します。そして、以下

の関数をFixedクラス・ファイルに追加する：

- パラメータとして渡された出力ストリーム・オブジェクトに固定小数点数の浮動小数点数表現を挿入する挿入 (") 演算子のオーバーロード。

のコードを実行

する:

```
#インクルード
<iostream>

int 固定メイン( void )
{ { const b( 10 ); const c(
  42.42f ); const d( b )
  { { を修正 void ) {
  { { メイン( void )
    a = Fixed( 1234.4321f );

    std::cout << "aは" << a << std::endl;
    std::cout << "b は " << b << std:: :
    std::cout << "c は " << c << std::endl;
    std::cout << "d は " << d << std::endl
    ;

    std::cout << "a is " << a.toInt() << " as integer" <<
    std::endl; std::cout << "b is " << b.toInt() << " as integer"
  } << std::endl; std::cout << "c is " << c.toInt() << " 整数として"
    << std::endl; std::cout << "d は " << d.toInt() << " 整数として"
    << std::endl;


    0を返す;
```

のようなものが出力されるはずだ:

```
$> ./a.out
デフォルトのコンストラクタ
Int コンストラクタ Float コ
ンストラクタ Copy コンストラ
クタ
というコピー代入演算子。
コピー代入演算子
aは1234.43
b は10
c は42.4219
d は10
a は整数で1234
b は整数で10
c は整数で42
d は整数で10である
。
$>
```


第六章

練習02：今、話していること

	エクササイズ02
	今、話しているのは
	ターンイン・ディレクトリ : <i>ex02/</i>
	提出ファイル: Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp
	使用できる関数: roundf (<cmath> から)

以下の演算子をオーバーロードするパブリック・メンバー関数をクラスに追加する：

- 6つの比較演算子：>、<、>=、<=、==、そして！
- 四則演算子：+、-、*、/。
- 4つのインクリメント/デクリメント(プリインクリメントとポストインクリメント、プリデクリメントとポストデクリメント)演算子は、 $1 + \epsilon$ のような最小の表現可能な ϵ から固定小数点値を増減させます。

これらの4つのパブリック・オーバーロード・メンバ関数をクラスに追加します：

- 静的メンバ関数 min は、固定小数点数の2つの参照をパラメータとして取り、最小のものへの参照を返す。
- 静的メンバ関数 min は、**定数** 固定小数点数で、最も小さいものへの参照を返す。
- 固定小数点数の2つの参照をパラメータとし、最大値への参照を返す静的メ

ンバ関数 `max`。

- への2つの参照をパラメータとする静的メンバ関数`max`。
の固定小数点数への参照を返す。

クラスのすべての機能をテストするのはあなた次第です。しかし、以下のコードを実行すると

```
#インクルード
<iostream>
メイン( void )
{
    固定 定数    a;
                b( 固定( 5.05f ) * 固定( 2 ) );

    std::cout << a << std::endl;
    std::cout << ++a << std::endl;
    std::cout << a << std::cout <<
    a++ << std::endl; std::cout << a <<
    std::endl;

    std::cout << b << std::endl;

    std::cout << Fixed::max( a, b ) << std::endl
;
}
0を返す;
```

以下のように出力されるはずである（読みやすくするため、以下の例ではコンストラクター／デストラクターのメッセージは削除されている）：


```
$> ./a.out
0
0.00390625
0.00390625
0.00390625
0.0078125
10.1016
10.1016
$>
```



0による除算を行った場合、プログラムがクラッシュしても構わない。

第VII章 練習問題03

： BSP

	エクササイズ03
BSP	
ターンイン・ディレクトリ : <i>ex03/</i>	
提出ファイル: Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp, Point.{h, hpp}, Point.cpp, bsp.cpp	
使用できる関数 : roundf (<cmath> から)	

機能的な**固定**クラスができたのだから、それを使うのもいいだろう。

ある点が三角形の内側にあるかどうかを示す関数を実装する。
とても便利でしょう？



BSPはバイナリー・スペース・パーティショニングの略です。どういたしまして。:)



練習問題03を解かなくても、このモジュールに合格することができます。

まず、2次元の点を表すクラス**Point**をオーソドックスなカノニカル形式で作ってみよう：

- プライベートメンバー
 - 固定された定数属性 x_0 。
 - 固定属性 y_0 。
 - 他に役に立つことは何でも。
- 一般会員：
 - x と y を 0 に初期化するデフォルトコンストラクタ。
 - 2つの定数浮動小数点数をパラメータとするコンストラクタ。 x と y をこれらのパラメータで初期化する。
 - コピーコンストラクタ。
 - コピー代入演算子のオーバーロード。
 - デストラクタ。
 - 他に役に立つことは何でも。

最後に、以下の関数を適切なファイルに実装する：

```
bool bsp( Point const a, Point const b, Point const c, Point const point);
```

- a 、 b 、 c ：三角形の頂点。
- ポイント：チェックするポイント。
- 戻り値点が三角形の内側にあれば真。そうでなければ偽。
したがって、点が頂点または辺上にある場合は偽を返す。

クラスが期待通りに動作することを確認するために、自分でテストを実施し、提出

すること。

第8章

提出と相互評価

通常通り、Gitリポジトリに課題を提出してください。あなたのリポジトリ内の作品だけが、ディフェンス中に評価されます。フォルダ名やファイル名が正しいかどうか、ためらわずに再確認してください。



????????????XXXXXXXXXX = \$3\$d6f957a965f8361750a3ba6c97554e9f

