



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、オーソドックスな正統派クラス形式

概要

本書は、C++モジュールのうち、Module 02の演習を収録したものです。

バージョン : 7

内容

I	はじめに	2
II	一般規定	3
III	新規則	5
IV	演習00：正統派カノン式で初めての授業	6
V	練習問題01：より便利な固定小数点数クラスを目指して	8
VI	練習曲02：今、私たちは話している	10
VII	練習問題03：BSP	12

第一章 はじめに

C++は、Bjarne Stroustrupが「C with Classes」（出典：[Wikipedia](#)）というプログラミング言語の元祖として作った汎用プログラミング言語である。

これらのモジュールの目標は、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。オブジェクト指向を学ぶには、多くの言語が推奨されます。この言語は複雑な言語であり、物事を単純化するために、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++が多くの側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、もしあなたが熟練したC++開発者になりたいのであれば、42のコモンコアの後にさらに進むのはあなた次第なのです

第II章 総則

コンパイル

- `c++`と`-Wall` `-Wextra` `-Werror`フラグでコードをコンパイルします。
- フラグ `-std=c++98` を追加しても、あなたのコードはコンパイルできるはずです。

フォーマットと命名規則

- 演習用のディレクトリは、`ex00`, `ex01`, ...のように命名されます。 `exn`
- ファイル、クラス、関数、メンバ関数、属性にガイドラインに必要な名前を付ける。
- クラス名は、**UpperCamelCase** 形式で記述します。クラスコードを含むファイルには、常にクラス名に従った名前が付けられます。例えば例えば、`ClassName.hpp/ClassName.h`, `ClassName.cpp`, または `ClassName.tpp`.例えば、レンガの壁を表すクラス "`BrickWall` "の定義を含むヘッダーファイルがあれば、そのファイル名は`BrickWall.hpp`となります。
- 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。あなたの好きなものに従えばいいのです。しかし、同僚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ために、最善を尽くしてください。

許可/禁止

あなたはもうCでコーディングしていない。C++の時代だ!というわけで。

- 標準ライブラリのほとんどすべてを使用することが許されています。したがって、すでに知っているものに固執するのではなく、使い慣れたC関数のC++的なバージョンをできるだけ使うのが賢い方法でしょう。
- ただし、それ以外の外部ライブラリは使用できません。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ます。以下の関数も禁止されています。`*printf()`、`*alloc()`、`free()`。これらを使用した場合、成績は0点、それで終わりです。

- 明示的に指定しない限り、`using`名前空間`<ns_name>`と友人キーワードは禁止です。そうでない場合、あなたの成績は-42となります。
- **STL** の使用はモジュール 08 のみ許可されています。つまり、コンテナ (`vector/list/map/` など) とアルゴリズム (`<algorithm>` ヘッダを含む必要があるもの) はそれまで使わないでください、ということです。さもなければ、あなたの成績は -42 になります。

いくつかのデザイン要件

- C++でもメモリリークは発生します。メモリを確保するときに (`new` キーワード)を使用する場合は、メモリリークを回避する必要があります。
- モジュール02からモジュール08までは、特に明記されている場合を除き、正教会の正書法で授業を設計する必要があります。
- 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）、演習では0を意味します。
- それぞれのヘッダーは、他のヘッダーと独立して使用できるようにする必要があります。従って、必要な依存関係はすべてインクルードしなければなりません。しかし、インクルードガードを追加することによって、二重インクルードの問題を回避しなければなりません。そうでなければ、あなたの成績は0点となります。

リードミー

- 必要であれば、いくつかのファイルを追加することができます（コードを分割するためなど）。これらの課題は、プログラムによる検証を行わないので、必須ファイルを提出する限り、自由に行ってください。
- 演習のガイドラインは短く見えても、例題には明示的に書かれていない要件が示されていることがあります。
- 始める前に、各モジュールを完全に読んでください本当に、そうしてください。
- オーディンによって、ツールによって!頭を使い!!!



多くのクラスを実装する必要があります。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る人でなければ、退屈に思えるかもしれません。



演習をこなすには、ある程度の自由度が与えられています。ただし、必須のルールは守り、怠慢は禁物です。多くの有益な情報を見逃すことになりますよ。理論的な概念については、ためらわずに読んでください。

第III章 新規則


今後、あなたが作成するクラスは、特に断りがない限り、すべて**正統派の正典形式**で設計されなければなりません。そして、以下の4つの必須メンバー関数を実装することになります。

- 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
- デストラクタ

クラス・コードを2つのファイル（分割してください。（.hpp/.h）にはクラスの定義が、ソースファイル（.cpp）には実装が記述されます。

第四章

演習00：正統派カノン式で初めての授業

	演習：00
正統派のカノン式で初めての授業	
ターンインディレクトリ： <code>ex00/</code>	
提出するファイル： <code>Makefile</code> 、 <code>main.cpp</code> 、 <code>Fixed.{h, hpp}</code> 、 <code>Fixed.cpp</code> 。	
禁止されている機能：なし	

整数と浮動小数点数を知っているつもりなんですね。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ください。さあ、読んでみてください。

今日まで、あなたがコードで使うすべての数は、基本的に整数か浮動小数点数、またはその変種（`short`、`char`、`long`、`double`など）のいずれかでした。上の記事を読むと、整数と浮動小数点数は逆の性質を持っていると考えてよいでしょう。

しかし、今日、状況は変わります。固定小数点数という新しい素晴らしい数値型を発見することになるでしょう。固定小数点数は、ほとんどの言語のスカラー型から姿を消してしまいましたが、性能、精度、範囲、正確さのバランスがとれた貴重なものです。そのため、固定小数点数はコンピュータグラフィックス、サウンドプロセッシング、科学的プログラミングなどに特に適しています。

C++には固定小数点数がないため、足し算をすることになります。バークレー校の[この記事](#)は良いスタートです。バークレー大学が何なのかわからない人は、Wikipediaのページの[この部分を読んでみてください](#)。

固定小数点数を表すクラスをOrthodox Canonical Formで作成する。

- プライベートメンバー
 - 固定小数点数の値を格納するための**整数**。
 - 小数点のビット数を格納する**静的な定数**です。この値は常に整数リテラルである 8 となります。
- 一般会員です。
 - 固定小数点数値の値を 0 に初期化するデフォルトコンストラクタです。
 - コピーコンストラクタです。
 - 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;` は、固定小数点値の生の値を返すものである。
 - メンバ関数 `void setRawBits(int const raw);` 固定小数点数の生の値を設定するものです。

このコードを実行する。

```
#include <iostream>

イン      main( void ) {
ン
トaを固定、 b(
a )を固定、
cを固定。

c=bである。

std::cout << a. getRawBits() << std::endl;
std::cout << b. getRawBits() << std::endl;
std::cout << c. getRawBits() << std::endl
となります。


} 0を返します。
```

のようなものが出力されるはずです。

```
$> ./a.out
というデフォルトコンス
トラクタと というコピー
コンストラクタ
コピー代入演算子が呼ばれる // <-- この行は実装によって欠落する可能性がある getRawBits メンバ関数が呼ばれる
という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 を呼び
出す。
getRawBitsメンバ関数が呼ばれた 0
getRawBitsメンバ関数が呼ばれた 0
という名のデス
トラクタ。
$>
```


第五章

練習問題01：より便利な固定小数点数クラスを目指して

	演習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>	
提出するファイル: <i>Makefile</i> 、 <i>main.cpp</i> 、 <i>Fixed.{h, hpp}</i> 、 <i>Fixed.cpp</i> 。	
使用可能な関数 : <i>roundf</i> (<i><cmath></i> から)	

前の練習は良いスタートでしたが、私たちのクラスはかなり役に立ちません。0.0という値しか表現できないのです。

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てください。

- 定数 **integer** をパラメータとするコンストラクタです。
対応する固定小数点値に変換しています。分数ビット値は、演習00のように8で初期化されます。
- 浮動小数点数の定数をパラメータとして受け取るコンストラクタです。
対応する固定小数点値に変換しています。分数ビット値は、演習00のように8で初期化されます。
- メンバ関数 **float toFloat(void) const;**
であり、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;**
であり、固定小数点値を整数値に変換する。

そして、**Fixed**クラスファイルに以下の関数を追加してください。

- 挿入 (") 演算子のオーバーロードで、固定小数点数の浮動小数点表現を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挿入します。

コードを実行する

。

```
#include <iostream>

int main( void ) {

    固定      a;
    const b( 10 ); const c(
    42.42f ); const d( b ) を修
    正しました。

    a = Fixed( 1234.4321f );

    std::cout << "a is " << a << std::endl;
    std::cout << "b is " << b <<
    std::endl; std::cout << "c is " << c <<
    std::endl; std::cout << "d is " << 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() << " as integer" << std::endl;
    std::cout << "b. toInt() << " as integer" << std::endl; std::cout << "d. toInt() << "
    as integer" << std::endl; std::cout << "d is " << d. toInt() << "
    as integer" << std::endl;
}


    02 返します
```

のようなものが出力されるはずです。

```
$> ./a.out
というデフォルトのコン
ストラクタ、Intというコ
ンストラクタ、Floatとい
うコンストラクタ、Copy
というコンストラクタが
あります。
というコピー代入演算子、 という
フロートコンストラクタ。
というコピー代入演算子、 という
デストラクタ
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
Destructorと呼ば
れる破壊者
Destructorと呼ば
れる破壊者
$>
```


第六章

Exercise 02: Now we're talking

	演習02
今、私たちが話しているのは	
ターンインディレクトリ : <code>ex02/</code>	
提出するファイル: <code>Makefile</code> 、 <code>main.cpp</code> 、 <code>Fixed.{h, hpp}</code> 、 <code>Fixed.cpp</code> 。	
使用可能な関数 : <code>roundf</code> (<code><cmath></code> 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してください。

- 6つの比較演算子 : `>`, `<`, `>=`, `<=`, `==`, `!=`。
- 四則演算の演算子。 `+`、`-`、`*`、`/`。
- 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）演算子で、`1 + E > 1`のように表現可能な最小の E から固定小数点値を増加または減少させることができます。

これらの4つのパブリック・オーバーロード・メンバー関数をクラスに追加します。

- 静的メンバ関数 `min` は、固定小数点数への参照を 2 つパラメータとして受け取り、最小のものへの参照を返します。
- への2つの参照をパラメータとする静的メンバ関数`min`。
の固定小数点数であり、最も小さいものへの参照を返す。
- 静的メンバ関数 `max` は、固定小数点数への参照を 2 つパラメータとして受け取り、最も大きいものへの参照を返します。
- への2つの参照をパラメータとする静的メンバ関数`max`。
の固定小数点数であり、最も大きいものへの参照を返す。

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ことで

```
#include <iostream>

int main( void ) {
    固定          a;
    固定された    b( 固定( 5.05f ) * 固定( 2 ) );
    定数
    std::cout << a << std::endl; std::cout
    << ++a << std::endl; std::cout << a <<
    std::endl; std::cout << ++a <<
    std::endl; std::cout << a << std::endl;
    std::cout << a++ << std::endl;
    std::cout << a << std::endl;

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

    std::cout << Fixed::max( a, b ) << std::endl;

} 0を返します。
```

のように出力されるはずです（読みやすくするため、以下の例ではコンストラクタ/デストラクタのメッセージを削除しています）。

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


第VII章 演習03 :

BSP

	演習03
BSP	
ターンインディレクトリ : <i>ex03/</i>	
提出するファイル: Makefile、main.cpp、Fixed.{h, hpp}、Fixed.cpp、Point.{h, hpp}、Point.cpp、bsp.cpp	
使用可能な関数 : roundf (<cmath> から)	

せっかく機能的な**Fixed**クラスができたのだから、それを使ってみたいですね。

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。
とても便利ですね。



BSPとは、Binary space partitioningの略です。

どういたしまして。
:)



このモジュールは、演習03を行わず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Orthodox Canonical Formで作成しましょう。

- プライベートメンバー
 - A 固定された定数属性x。
 - A 固定された定数属性y。
 - その他、役に立つことなら何でも。
- 一般会員です。
 - x と y を 0 に初期化するデフォルトコンストラクタです。
 - 2つの定数浮動小数点数をパラメータとして受け取るコンストラクタです。x とyはこれらのパラメータで初期化されます。
 - コピーコンストラクタです。
 - コピー代入演算子のオーバーロード。
 - デストラクタ。
 - その他、役に立つことなら何でも。

最後に、以下の関数を適当なファイルに実装してください。

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

- a、b、c。私たちの愛する三角形の頂点。
- ポイントです。チェックするポイント。
- を返す。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、その点が頂点または辺である場合は、偽を返す。
。

自分のクラスが期待通りに動作することを確認するために、自分でテストを実装して提出する。

