



C++ - モジュール 03

継承

概要

このドキュメントには、C++ モジュールのモジュール03の練習問題が含まれています。

バージョン: 7

内容

I	はじめに	2
II	一般規定	3
III	練習00: ああ...オープン	5
IV	練習01: セレーナ、私の愛しい人!	7
V	エクササイズ02: 反復練習	8
VI	練習03: 今のは変だ!	9
VII	提出と相互評価	11

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のコモン・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。さもないければ、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！頭を使い！




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。

第三章

練習00： ああ...オープン

	エクササイズ： 00
そして...オープン	
ターンイン・ディレクトリ： <i>ex00/</i>	
提出するファイル： Makefile, main.cpp, ClapTrap.{h, hpp}, ClapTrap.cpp	
禁止されている関数： なし	

まず、クラスを実装しなければならない！ なんと斬新な！

これはClapTrapと呼ばれ、括弧で指定された値に初期化された以下のプライベート属性を持つ：

- コンストラクタのパラメータとして渡される名前。
- ヒットポイント（10）は、クラップトラップの健康状態を表す。
- エネルギーポイント (10)
- 攻撃ダメージ (0)

ClapTrapがよりリアルに見えるように、以下のパブリック・メンバー関数を追加する：

- `void attack(const std::string& target);`
- `void takeDamage(unsigned int amount);`
- `void beRepaired(unsigned int amount);`

クラップトラックが攻撃したとき、その対象は＜攻撃ダメージ＞のヒットポイントを失う。クラップトラックが自身を修復すると、＜量＞のヒット・ポイントを取り戻す。攻撃と修復にはそれぞれ1エネルギー・ポイントを消費する。もちろん、ヒットポイントもエネルギーポイントも残っていない場合、ClapTrapは何もできない。

これらのメンバ関数のすべてで、何が起きたかを説明するメッセージを表示しなければならない。例えば、attack()関数は次のようなメッセージを表示する（もちろん、角括弧は付けない）：


ClapTrap<名前>は<ターゲット>を攻撃し、<ダメージ>点のダメージを与える！

コンストラクタとデストラクタはメッセージも表示しなければならないので、相互評価者はそれらが呼び出されたことを簡単に知ることができます。

コード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。

第四章

練習01：セリーナ、私の愛しい人！

	エクササイズ：01
セレーナ、私の愛しい人！	
ターンイン・ディレクトリ：ex01/	
提出ファイル：前回の演習のファイル + ScavTrap.{h, hpp}, ScavTrap.cpp	
禁止されている関数：なし	

ClapTrapはいくらあっても足りないので、派生ロボットを作ることにする。これは**ScavTrap**と名付けられ、ClapTrapのコンストラクタとデストラクタを継承する。しかし、コンストラクター、デストラクター、attack()は異なるメッセージを表示する。結局のところ、ClapTrapは自分の個性を認識している。

適切な建設／破壊の連鎖がテストで示されなければならないことに注意してください。ScavTrapが作成されると、プログラムはClapTrapを構築することから始める。破壊は逆の順序である。なぜか？

ScavTrapはClapTrapの属性を使用し（結果的にClapTrapを更新する）、初期化しなければならない：

- コンストラクタのパラメータとして渡される名前。
- ヒットポイント（100）は、クラップトラップの健康状態を表す。
- エネルギーポイント（50）
- 攻撃ダメージ（20）

ScavTrapもまた、独自の特別な能力を持つことになる：


```
void guardGate();
```

このメンバ関数は、ScavTrapがゲートキーパモードになったことを知らせるメッセージを表示します。

プログラムにテストを追加することもお忘れなく。

第五章

エクササイズ02：反復練習

	エクササイズ：02
反復作業	
ターンイン・ディレクトリ：ex02/	
提出ファイル：提出ファイル：前回の練習問題 + FragTrap.{h, hpp}, FragTrap.cpp	
禁止されている関数：なし	

クラップトラップを作るのは、おそらくあなたの神経を逆なでし始めたのだろう。

さて、ClapTrapを継承した**FragTrap**クラスを実装する。これはScavTrapとよく似ている。しかし、その構築と破壊のメッセージは異なっていなければなりません。適切な構築／破壊の連鎖がテストで示されなければなりません。FragTrapが作成されるとき、プログラムはClapTrapを構築することから始まります。破壊は逆の順序である。

アトリビュートについては同じだが、今回は値を変えた：

- コンストラクタのパラメータとして渡される名前。
- ヒットポイント（100）は、クラップトラップの健康状態を表す。
- エネルギーポイント（100）
- 攻撃ダメージ（30）

フラグトラップにも特別な能力がある：


```
void highFivesGuys(void);
```

このメンバ関数は、正のハイタッチ要求を標準出力に表示する。

もう一度、プログラムにテストを追加する。

第六章

練習03：今度は変だ！

	練習：03
今は奇妙だ！	
ターンイン・ディレクトリ：ex03/	
提出ファイル：提出ファイル：前回の練習問題+DiamondTrap.{h, hpp}、DiamondTrap.cpp	
禁止されている関数：なし	

この練習では、半分FragTrapで半分ScavTrapのClapTrapというモンスターを作ります。名前はDiamondTrapで、FragTrapとScavTrapの両方を継承します。これはとてもリスキーだ！

DiamondTrapクラスはname private属性を持つ。この属性に、ClapTrap基底クラスのものと同じ変数名（ここではロボットの名前について話していない）を与える。

より明確にするために、2つの例を挙げよう。

ClapTrapの変数がnameの場合、DiamondTrapの変数にnameを与える。

ClapTrapの変数が_nameの場合、DiamondTrapの変数に_nameを与える。

その属性とメンバ関数は、いずれかの親クラスから選択される：

- コンストラクタのパラメータとして渡される名前。
- ClapTrap::name（コンストラクタのパラメータ+接尾辞"_clap_name"）
- ヒットポイント（フラッグトラップ）

- エネルギーポイント (ScavTrap)
- 攻撃ダメージ (フラッグトラップ)
- アタック() (スカブトラップ)

ダイヤモンド・トラップは、親クラスの特別な機能に加え、独自の特別な能力を持つ:

```
void whoAmI();
```

このメンバ関数は、その名前とClapTrap名の両方を表示する。

もちろん、DiamondTrapのClapTrapサブオブジェクトは一度だけ作成される。そう、トリックがあるのだ。

もう一度、プログラムにテストを追加する。



Wshadowと-Wno-shadowコンパイラ・フラグをご存知ですか?



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。

第七章

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\$cf36316f07f871b4f14926007c37d388

