



C++ - モジュール 08

テンプレート化されたコンテナ、イテレータ
、アルゴリズム

概要

本書は、C++モジュールのうち、Module 08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	モジュール固有のルール	5
IV	練習曲00：イージーファインド	6
V	練習問題01：スパン	7
VI	練習曲02：変異した忌まわしいもの	9

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。さもないと、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となりますよ。理論的な概念については、ためらわずに読んでください。

第三章

モジュール固有のルール

このモジュールでは、標準のコンテナや標準のアルゴリズムがなくても、演習を解くことができることに気づきでしょう。


しかし、それらを使用することこそ、本モジュールの目標なのです。STLを使うことは許されています。そう、コンテナ（vector/list/map/など）とアルゴリズム（ヘッダ<algorithm>で定義）を使うことができるのです。しかも、できる限り使うべきです。したがって、適切な場所であればどこでもそれらを適用するよう、最善を尽くしてください。

そうしないと、たとえコードが期待通りに動いたとしても、非常に悪い評価を受けることになります。怠けてはいけません。

テンプレートは通常通りヘッダーファイルで定義することができます。あるいは、テンプレートの宣言をヘッダーファイルで書き、その実装を.tppファイルで書くこともできます。いずれにせよ、ヘッダーファイルは必須であり、.tppファイルは任意です。

第四章

練習曲00：イージーファインド

	演習：00
イージーファインド	
ターンインディレクトリ：ex00/	
提出するファイル:Makefile, main.cpp, easyfind.{h, hpp}とオプションファイル。easyfind.tpp	
禁止されている機能:なし	

最初の簡単な運動は、正しいスタートを切るための方法です。

型Tを受け入れる関数テンプレートeasyfindを書きなさい。それは2つのパラメータを取ります。

1つ目の型はT、2つ目の型は整数です。

Tが**整数**のコンテナであるとする、この関数は第1パラメータに第2パラメータが最初に出現することを見つけなければならない。


発生しない場合は、例外を投げるか、任意のエラー値を返すことができます。もしインスピレーションが必要なら、標準的なコンテナがどのように動作するかを分析してみてください。

もちろん、すべて想定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。



連想コンテナを扱う必要がない。

第五章 演習01：ス パン

	演習：01
スパン	
ターンインディレクトリ：ex01/	
提出するファイル:Makefile、main.cpp、Span.{h, hpp}、Span.cpp。	
禁止されている機能:なし	

最大N個の整数を格納できる**Span**クラスを開発せよ。Nはunsigned int変数で、コンストラクタに渡される唯一のパラメータとなる。

このクラスは、**Span**に数字を1つ追加するためのaddNumber()というメンバ関数を持つことになります。これは、**Span**を埋めるために使用されます。すでにN個の要素が格納されている場合に新しい要素を追加しようとすると、例外が発生するはずです。

次に、shortestSpan()とlongestSpan()の2つのメンバ関数を実装する。

それぞれ、格納されているすべての数値の間の最短スパンまたは最長スパン（お好みで距離）を求め、それを返します。もし、格納されている数が1つもないか、1つしかない場合は、最短距離は求まらない。したがって、例外を投げる。

もちろん、あなた自身のテストを書き、それは以下のものよりもはるかに徹底的なものになるでしょう。少なくとも1万個以上の数字で**Span**をテストしてください。もっと多い方が良い。

このコードを実行する。

```
int main()
{
    Span sp = Span(5);

    sp.addNumber(6); sp.

$> ./ex01
2
14
$>

std::cout << sp. shortestSpan() << std::endl;
std::cout << sp. longestSpan() << std::endl;

0を返します。
}
```



手がかりがない場合は、コンテナについて勉強してください。いくつかのメンバ関数は、コンテナに一連の要素を追加するために、イテレータの範囲を取ります。

第六章

練習曲02：変異した忌まわしいもの

	演習：02
変異した醜態	
ターンインディレクトリ： <i>ex02/</i>	
提出するファイル: <i>Makefile</i> , <i>main.cpp</i> , <i>MutantStack.{h, hpp}</i> とオプションファイル。 <i>MutantStack.tpp</i>	
禁止されている機能: なし	

さて、そろそろもっと本格的な話に移ろうか。何か変なものを開発しよう。

`std::stack` コンテナは非常に素晴らしいものです。しかし残念なことに、これは反復可能ではない唯一の STL コンテナの一つです。それは残念なことです。

しかし、なぜそれを受け入れなければならないのでしょうか？特に、足りない機能を作るために、オリジナルのスタックを勝手に切り刻むことができるのならなおさらです。

この不正を修復するためには、`std::stack` コンテナをイテラブルにする必要があります。

MutantStack クラスを作成します。`std::stack`で実装する予定です。これは、そのすべてのメンバー関数に加え、イテレータという追加機能を提供するものである。

もちろん、すべてが期待通りに動くことを確認するために、自分でテストを書いて提出することになります。

以下、テスト例をご覧ください。

```
int main()
{
    MutantStack<int> mstack

    mstack.push(5)、
    mstack.push(17);

    std::cout << mstack.top() << std::endl;

    mstack.pop();

    std::cout << mstack.size() << std::endl;

    mstack.push(3)、
    mstack.push(5)、 mstack.
    push(737);
    //[...]
    mstack.push(0);

    MutantStack<int>::iterator it = mstack.begin();
    MutantStack<int>::iterator ite = mstack.end();

    ++it
    --ite;
    while (it != ite)
    {
        std::cout << *it << std::endl;
        ++it
    }
    std::stack<int> s(mstack);
    0を返します。
}
```