ft_containers

C++コンテナ、イージーモード

概要
C++の標準コンテナには、すべて特定の使い方があります。
それを理解するために、再実装してみましょう。

バージョン：5.2

内容

第一章 目的

このプロジェクトでは、C++標準テンプレートライブラリのいくつかのコンテナ型を実装します。

各標準コンテナの構造を参考にする必要があります。その中で正統派のカノン形式の一部が欠落している場合は、実装してはいけない。

注意点として、C++98標準に準拠しなければならないので、それ以降のコンテナの機能は実装しては**いけません**が、すべてのC++98の機能（非推奨のものも含む）は期待できます。

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。
- ソースファイルをコンパイルするMakefileを提出する必要があります。再リンクはしてはいけません。
- Makefileには少なくともルールが含まれている必要があります。
  $(NAME)、all、clean、fclean、re。

フォーマットと命名規則

- 各コンテナについて、適切な名前のクラスファイルを提出する。
- *さよならノルミネット!*コーディングスタイルの強制はありません。あなたの好きなスタイル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、彼らが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、ベスト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。

- 標準ライブラリにあるものはすべて使ってよいことになっています。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。

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。メモリを確保する際には
  メモリリーク

- 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

- 必須ファイルを提出する限り、必要に応じていくつかのファイルを追加したり（コードを分割するためなど）、作品を自由に構成することができます。
- オーディンによって、トールによって!頭を使え!!!

STLコンテナを再コード化するのが目的なので、もちろん
は、自分の**ものを**実装するために**使用することはできません。**

第三章 必須項目

以下のコンテナを実装し、必要な<container>.hppファイルをMakefileで回してください。

- ベクトル
  vector<bool>の特殊化をする必要はない。
- 地図
- 山盛り
  ベクタークラスをデフォルトの基礎コンテナとして使用します。しかし、STLを含む他のコンテナとの互換性は保たなければなりません。

この課題はスタックなしでも合格できます（80/100）。
しかし、ボーナスパートを行う場合は、3つの必須コンテナを実装する必要があります。　　ベクトル、マップ、スタックです。

また、実装もしなければならない。

- std::iterator_traits
- std::reverse_iterator
- std::enable_if
  はい、C++11ですが、C++98の要領で実装することができるようになります。これは、SFINAEを発見してもらうために聞いているのです。
- std::is_integral
- std::equal および/または std::lexicographical_compare
- std::ペア
- std::make_pair

III.1　必要条件

- 名前空間は ft でなければならない。
- コンテナで使用する各内部データ構造は、論理的で正当なものでなければなりません（つまり、mapに単純な配列を使用するのはダメということです）。
- 標準コンテナで提供されるもの以上のパブリック関数を実装することはできません。それ以外のものはすべてプライベートかプロテクトでなければなりません。パブリックな関数や変数は、それぞれ**正当化さ**れなければなりません。
- 標準コンテナのすべてのメンバー関数、非メンバー関数、オーバーロードが想定されています。
- オリジナルのネーミングを踏襲すること。細部にまで気を配ること。
- コンテナがイテレータシステムを持つ場合、それを実装する必要があります。
- std::allocatorを使用する必要があります。
- 非メンバーのオーバーロードでは、キーワードfriendを使用することができます。friendの各使用は
  は正当でなければならず、評価時に確認される。
- もちろん、std::map::value_compareの実装のために、キーワード
  を使用することができます。

https://www.cplusplus.com/ を使用することができます。
およびhttps://cppreference.com/ を参照してください。

III.2　テスト

- また、あなたの防衛のためにテスト、少なくともmain.cppを提供する必要があります。例としてあげられた main よりもさらに踏み込んだものでなければなりません!
- 同じテストを実行する2つのバイナリを作成する必要があります。1つは自分のコンテナのみで、もう1つはSTLコンテナで作成します。
- 出力と**パフォーマンス/タイミングを**比較する（あなたのコンテナは最大20倍遅くなる可能性があります）。
- ft::<container> を使ってコンテナをテストします。

main.cpp ファイルは、イントラネットのプロジェクトページでダウンロードできます。

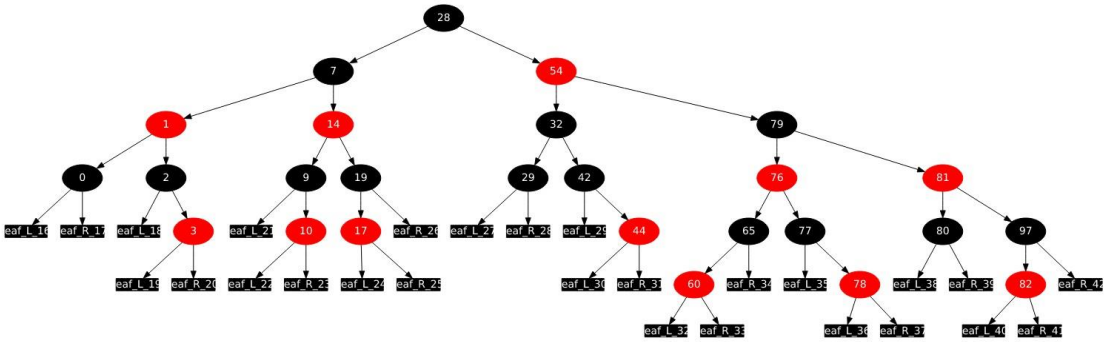
第四章 ボーナス

パート

最後の1つのコンテナを実装すると、さらにポイントが加算されます。

- セット

しかし、今回は**赤黒い木が必須です**。



ボーナスパーツは、必須パーツがPERFECTである場合にのみ査定されます。パーフェクトとは、必須パートが統合的に行われ、誤動作することなく動作することを意味します。　　必須項目をすべてクリアしていない場合、ボーナスパーツの評価は一切行われません。

第五章

提出と相互評価

通常通り、Gitリポジトリに課題を提出する。防衛戦では、あなたのリポジトリ内の作品だけが評価されます。ファイル名が正しいかどうか、遠慮なく再確認してください。