CSC C++

Домашнее задание №3
Умный указатель `linked_ptr`

29 ноября 2015

Постановка задачи. Разработайте шаблонный класс `linked_ptr<T>`, реализующий RAII обертку над указателем. Шаблонным параметром является тип, указатель на который моделирует `linked_ptr`.

Описание. `linked_ptr` — умный указатель (т. е. обёртка над обычным указателем), позволяющий совместно использовать хранимый указатель из нескольких экземпляров класса `linked_ptr`. Память по указателю, хранимому в `linked_ptr`, будет освобождена только тогда, когда все экземпляры `linked_ptr`, хранящие этот указатель, будут уничтожены, либо из них этот указатель будет явно высвобожден.

При копировании умного указателя `a` в `b` (или при инициализации `b(a)`), в `a` добавляется указатель на `b`, а в `b` — указатель на `a`. Таким образом, все экземпляры класса `linked_ptr`, которые хранят один и тот же указатель, оказываются в двусвязном списке — отсюда и название данного умного указателя.

По своей функциональности `linked_ptr` практически эквивалентен `std::shared_ptr`. По сравнению с последним `linked_ptr` обладает меньшим временем инициализации, т. к. не выделяет дополнительных блоков памяти (как `shared_ptr` для счетчика ссылок), а хранит все указатели, работающие с одним объектом, в списке. Впрочем, такая реализация требует от `linked_ptr` большего времени на копирование и удаление, нежели `shared_ptr`.

Требования.

1. Ваша реализация должна соответствовать описанной выше идее поддержания связанных указателей в общем списке.
2. В реализации не должно быть выделений динамической памяти, которые можно избежать. Таким образом, например, нельзя использовать `std::list` для хранения элементов в списке.
3. Связывание и удаление умного указателя из общего списка должно выполняться за $O(1)$.
4. Реализация должна обеспечивать требование определения типа параметра шаблона к моменту его удаления из деструктора `linked_ptr`.
5. Интерфейс `linked_ptr` должен содержать следующие методы, аналогичные методам `std::shared_ptr`:
   (a) Конструкторы `linked_ptr<T>`:
      i. без параметров;
      ii. от `T *`, константного `linked_ptr<T>`;
      iii. а также от `U *` и `linked_ptr<U>`, если `U *` неявно приводим к `T *`.
   (b) Конструкторы от одного аргумента не должны задавать дополнительное приведение типа.

(c) Операторы присваивания от `linked_ptr<T>` и `linked_ptr<U>`. Условия аналогичны конструкторам.

(d) Семейство функций `reset`:

i. без параметров;

ii. от `T *`;

iii. от `U *` (см. выше).

(e) методы `swap()`, `get()`, `unique()` аналогичные методам из `std::shared_ptr`.

(f) Операторы `->` и разыменования.

(g) Операторы проверки на равенство/неравенство.

(h) Операторы сравнения (например, для использования экземпляров `linked_ptr<T>` в качестве элементов `std::set`).

(i) Безопасное приведение к логическому выражению.

6. Реализация должна соответствовать требованиям из Правил сдачи, т. е., например, в ней не должно быть утечек памяти, она должна быть оформлена в соответствии с указанными принципами кодирования и т. п.

Дополнительное задание. При использовании вашего класса с неким шаблонным параметром `C` требуйте полностью определённый тип параметра шаблона `C` только в момент инициализации переменной типа `linked_ptr<C>`, но не, например, в момент объявления переменной типа `linked_ptr<C>`.

Примечания.

- Т. к. интерфейс `linked_ptr` является подмножеством `std::shared_ptr`, настоятельно рекомендуется использовать интерфейс `std::shared_ptr` в качестве образца. Явно не специфицированная в данном задании функциональность должна работать также, как в `std::shared_ptr`, например, см. как работает `unique()` на неинициализированном `std::shared_ptr`.

Формат сдачи. Решение необходимо оформить в виде библиотеки, состоящей из одного заголовочного файла. Заголовочный файл должен называться `linked_ptr.h`.

Класс `linked_ptr` должен находиться в пространстве имён `smart_ptr`. Все необходимые вспомогательные классы и функции должны быть либо скрыты внутри класса `linked_ptr`, либо находиться в пространстве имён `smart_ptr::details`.

Допускается нахождение в репозитории рядом с файлом `linked_ptr.h` вспомогательных файлов, которые вы можете использовать для тестирования вашей библиотеки, а также `Makefile/CMakeLists.txt`, они не будут участвовать в проверке.

Таким образом, ваша директория в Subversion должна выглядеть примерно следующим образом:

```
ha3
├── Makefile
├── test.cpp
└── linked_ptr.h
```

Сроки сдачи. Будет три срока, к которым можно будет сдавать домашнее задание:

- 8:00 6 декабря (воскресенье),
- 8:00 13 декабря (воскресенье),
- 8:00 20 декабря (воскресенье).

Если домашнее задание не принимается с первой попытки, его можно попробовать сдать со следующей попытки.