Задание 7

Тестирование контейнеров

Попрактикуемся в написании unit-тестов с использованием библиотеки Google Test.

Unit-тестирование – очень полезная практика при разработке программного обеспечения. Полезна она по следующим причинам:

1. Помогает выявить недостатки реализации. В ходе написания unit-тестов зачастую становится понятно, что в тестируемом коде есть какие-нибудь недочёты по логике работы, удобству и т.д. Фактически, unit-тест выступает в качестве первого пользователя тестируемого кода (конечно, если речь идёт о своевременном написании тестов – сразу после реализации кода). А пользовательский опыт всегда полезен при оценке реализованного решения.

2. Даёт возможность пояснить некоторые тонкие моменты для пользователя. Довольно популярной практикой является написание тестов по аналогии с написанием документации. То есть тестовые сценарии строятся таким образом, чтобы продемонстрировать все предполагаемые варианты использования тестируемого кода. Читая тесты, можно понять логику работы тестируемого кода, его инварианты, как им пользоваться в разных случаях.

3. Контроль регрессий (лат. *regressio* «движение назад, возврат, отход»). В ходе тестирования программных продуктов важно проверять на качество не только новую функциональностью, но и факт того, что старые возможности (ранее протестированные, например, в предыдущей версии) не пострадали. Это называется «регрессионное тестирование». Unit-тесты – это прекрасный инструмент для такого рода тестирования. После реализации какого-то модуля (класса, функции) программист спустя некоторое время может вносить изменения в этот модуль (исправляя ошибки или добавляя новые функциональности). При внесении изменений довольно часто вносятся и ошибки (такова уж судьба всех программистов ☹). И unit-тесты дают возможность проверить, что ранее реализованный код не сломался при добавлении нового. Имея unit-тест с достаточной полнотой тестовых сценариев на какой-то тестируемый модуль (класс, функцию), можно автоматически (запустив тест) проверить корректность старых возможностей кода. Конечно, для нового кода придётся добавить новые unit-тесты. А при внесении изменений в логику работы – возможно, и обновить старые тесты.

Если первые два пункта из вышеперечисленных не накладывают никаких ограничений на unit-тесты, а даже наоборот подсказывают, что тестов должно быть много и подробных, то третий пункт вносит ещё одну важную мысль – unit-тесты должны быть быстрыми. Это требуется для более комфортной каждодневной работы программистов. Чтобы после внесения небольших изменений не приходилось ждать долгое время прохождения всех тестов.

В этой работе мы практикуемся в написании unit-тестов и их использования в повседневной работе.

Формулировка задания

В задании требуется реализовать unit-тесты на два контейнера – последовательный и спискового типа.

За основу для unit-тестирования можно взять либо реализованные в предыдущей самостоятельной работе контейнеры (предпочтительный вариант), либо уже готовые контейнеры из стандартной библиотеки:

- std::vector (требует подключения заголовочного файла <vector>)

- std::list (требует подключения заголовочного файла <list>)

Реализовать не менее 10-ти тестовых сценариев для каждого из контейнеров. Конкретные сценарии придумать самостоятельно, но проверять они должны основную функциональность контейнеров:
- создание контейнера

- вставка элементов в конец

- вставка элементов в начало

- вставка элементов в середину

- удаление элементов из конца

- удаление элементов из начала

- удаление элементов из середины

- получение элементов из контейнера

- получение размера контейнера (фактическое количество элементов)

Дополнительное задание 1.

Реализовать unit-тест на копирование контейнера. Тест должен проверять, что после копирования одного объекта контейнера в другой содержимое обоих контейнеров одинаково

Дополнительное задание 2.

Реализовать unit-тест на удаление контейнера. Тест должен проверять, что для всех элементов внутри контейнера был вызван деструктор во время уничтожения объекта контейнера.

Дополнительное задание 3. Добавить unit-тест на перемещение контейнера (вызов перемещающего оператора присваивания).

Уделить внимание структуре и именованию тестовых сценариев. Предлагаемый вариант оформления следующий.

- Тесты на каждый контейнер должны располагаться в отдельном .cpp-файле с именем **«[Имя_контейнера]_tests.cpp»** (пример – vector_tests.cpp). Такой подход даст наглядную структуру файлов.

- Все тесты (оба .cpp-файла) должны собираться в один исполняемый файл **«containers_tests»**. Это позволит запускать все тесты сразу без дополнительных действий.

- Логика именования тестовых сценариев следующая: **«[Имя_контейнера].[Сценарий]»** (пример – vector.push_back).

- Названия тестовых сценариев лучше делать наглядным. Например, тест броска исключения std::out_of_range из метода std::vector::at в случае передачи невалидного индекса может носить название: **«vector.at_out_of_range»** или **«vector.at_invalid_index»**. Впрочем, делать слишком многословные названия не стоит.

Самостоятельно попробовать запустить получившееся тестовое приложение с разными аргументами командной строки, разобраться в логике их работы. Рекомендуемые для ознакомления и имеющие довольно значительную практическую ценность параметры:

- **gtest_filter** – позволяет путём передачи фильтра по названию запустить только конкретные тесты. Например, вот такой запуск:

```
./containers_tests --gtest_filter=vector.*
```

приведёт к запуску только тестов контейнера vector (если следовать логике именования тестовых сценариев, предложенной выше).

- **gtest_repeat** – позволяет повторить запуск тестов указанное количество раз, что очень полезно при исследовании плавающих проблем. Например, вот такой запуск:

```
./containers_tests --gtest_repeat=10
```

приведёт к тому, что все тесты запустятся 10 раз.

- **gtest_output** – позволяет сгенерировать xml-отчёт по результату запуска тестов. Пример запуска:

```
./containers_tests --gtest_output=xml:report.xml
```

Итоговые требования

1. реализовать два контейнера и код их использования
2. для сборки использовать CMake
3. выгрузить результат на github.com в свой аккаунт

В Чат с преподавателем в Личном кабинете отправить:

1. ссылку на репозиторий с исходниками приложения