«Итератор»

Разработать программу на языке Си++ (ISO/IEC 14882:2014), демонстрирующую решение поставленной задачи. Весь код разбить на заголовочные файлы *.h и соответствующие им файлы реализации *.cpp (а также main.cpp, файл конфигурации системы сборки CMake **и отчет о покрытии кода**). По выполненной работе составить отчет согласно требованиям ГОСТ 7.32-2017 содержащий диаграмму классов согласно спецификации UML 2.0 и предоставить его копию в формате PDF. Все необходимые файлы разместить в git-репозитории (ссылка будет указана отдельно), ***взаимодействие выполнять с использованием платформы GitHub***. Решения, предоставленные иными способами, не принимаются. Выполненные работы подлежат процедуре защиты. ***Необходимым условием является защита лабораторной работы 2.*** Защита проводится во время срока выполнения работы. Срок выполнения работы 19 июня 2021 года. После истечения срока работа считается не сданной и оценивается в 0 баллов. Все присланные работы проверяются на отсутствие ошибок сборки в автоматическом режиме, в случае отсутствия ошибок выполняется ручная проверка кода, в противном случае работа оценивается в 0 баллов. Ручная проверка работ выполняется преподавателем – оценивается полнота выполнения задания, а также качество решения. Оценка выставляется в 10 балльной шкале. Все работы проверяются в системе Антиплагиат. Компилятор и операционная система, используемые при проверке: GNU Compiler Collection 10.2.0 цель x86_64-slackware-linux система Slackware Linux current 2020-10-26.

Количество баллов, начисляемых за выполнение пункта задания, указано в квадратных скобках.

Постановка задачи

Переработать классы, разработанные в рамках лабораторной работы 2.

Разработать шаблоны классов, объекты которых реализуют типы данных, указанные ниже. Для этих шаблонов классов разработать необходимые конструкторы, деструктор, конструктор копирования. Разработать операции: добавления/удаления элемента (уточнено в задаче); получения количества элементов; доступа к элементу (перегрузить оператор `[]`). При ошибках запускать исключение.

Разработать два вида итераторов (обычный и константный) для указанных шаблонов классов.

В главной функции разместить тесты, разработанные с использованием библиотеки GoogleTest.

При разработке тестов, добиться полного покрытия. Отчет о покрытии приложить к работе.

1. [**2 балла, обязательное задание**] Шаблон «динамический массив объектов». Размерность массива не изменяется в момент его переполнения. Начальная размерность задается как параметр конструктора, значение по умолчанию 0. Метод изменения размера. Добавление/удаление элемента в произвольное место.
2. [**2 балла, обязательное задание**] Шаблон «стек» (внутреннее представление – динамический массив хранимых объектов). Размерность стека увеличивается в момент его переполнения. Начальная размерность задается как параметр конструктора, значение по умолчанию 0. Добавление/удаление элемента в начало и в конец.
3. *[2 балла, дополнительное задание]* Шаблон «односвязный список объектов». Добавление/удаление элемента в произвольное место.
4. *[2 балла, дополнительное задание]* Шаблон «циклическая очередь» (внутреннее представление – динамический массив хранимых объектов). Добавление/удаление элемента в произвольное место.
5. *[2 балла, дополнительное задание]* Шаблон «двоичное дерево объектов». Добавление/удаление элемента в произвольное место.

Взаимодействию с использованием платформы GitHub

1. Создаем новую ветку ***ИМЯ*** в локальном репозитории. Перед созданием новой ветки необходимо убедиться, что локальный код синхронизирован с кодом из репозитория на ветке main. Для этого:

1.1 git checkout main

1.2 git pull

1.3 git checkout -b ***ИМЯ***

2. В новой ветке создаем новую директорию (mkdir <Ваше название директории>)
3. Пишем код в новой директории
4. В процессе написания кода можно (нужно) делать коммиты. Для этого:
   4.1 git status - посмотреть отслеживаемые/неотслеживаемые файлы
   4.2. git add <Название файла не в индексе для коммита> (можно писать git add -A, тогда добавятся все неотслеживаемые файлы)
   4.3. git commit -a -m 'Комментарий к коммиту'
5. Перед сдачей кода желательно сделать squash, чтобы избежать в конечном итоге длинной истории коммитов. Для этого
   5.1 git log --oneline - посмотреть историю коммитов
   5.2 git reset --soft HEAD~N, где N - последние N коммитов
   5.3 git commit -m 'Комментарий к общему коммиту'
6. Отправляем код на проверку:
   6.1 git push origin ***ИМЯ***
7. Открываем pull request (PR). Для этого, после выполнения пункта 6, открываем страницу вашего репозитория, нажимаем Compare & pull request. В теме пишем название/номер лабораторной, опционально пишем то, что сделано. Добавляем преподавателя и ассистентов в поля reviewers и assignee. Нажимаем на кнопку Create pull request.
8. Ждем, пока ваш pull request будет просмотрен. Если все выполнено корректно - происходит апрув PR и выполняется мерж. В противном случае будут оставлены замечания, которые нужно будет исправить, после чего нужно будет повторить пункт 6, при этом новый PR открывать не нужно. После исправлений нужно написать комментарий "Исправлено" и запросить реревью.

Обсуждение запрашиваемых изменений происходит в PR, для этого специально есть комментарии.

Справочная литература

Романов Е.Л. *Практикум по программированию на C++: Уч. пособие.* СПб: БХВ-Петербург;

Новосибирск: Изд-во НГТУ, 2004. – 432 с.

Шилдт Г. *Полный справочник по языку C++.* – 4-е изд. М.: Вильямс. 2006. – 800 с.

Буч Г., Рамбо Д., Джекобсон А. *Язык UML. Руководство пользователя.* – 2-е изд. – М.: ДМК Пресс; СПб.: Питер, 2004. – 432 с.

Scott Chacon, Ben Straub *Pro Git.* (https://git-scm.com/book/ru/v2/)

Шпаргалка по Git от GitHub. (https://github.github.com/training-kit/downloads/ru/github-git-cheat-sheet/)

GoogleTest User's Guide (https://github.com/google/googletest/blob/master/docs/index.md)

Modern C++ Programming with Test-Driven Development (https://github.com/dave00galloway/shiny-octo-archer/blob/master/Modern%20C%2B%2B%20Programming%20with%20Test-Driven%20Development.pdf)

Александр Алексеев *Определение степени покрытия кода на C/C++ тестами* (https://eax.me/c-code-coverage/)

iterator - C++ Reference (https://www.cplusplus.com/reference/iterator/iterator/)

std::iterator - cppreference.com (https://en.cppreference.com/w/cpp/iterator/iterator)

Writing a custom iterator in modern C++ (https://internalpointers.com/post/writing-custom-iterators-modern-cpp)