

Оглавление

Модульное тестирование	2
Библиотека Catch2 C++	2
Подключение библиотеки	2
Настройка автозапуска тестов	7
Задание на лабораторную работу	9
Проверка публичных функций библиотек	9
Проверка работы с памятью	11

Модульное тестирование

Библиотека Catch2 C++

Catch2 — это в основном среда модульного тестирования для C++, но она также предоставляет базовые функции микротестирования и простые макросы BDD.

Главное преимущество Catch2 заключается в том, что его использование просто и естественно. Имена тестов не обязательно должны быть допустимыми идентификаторами, утверждения выглядят как обычные логические выражения C++, а разделы предоставляют удобный и локальный способ совместного использования кода настройки и демонтажа в тестах.

GitHub: <https://github.com/catchorg/Catch2/tree/v2.x>

Подключение библиотеки

Шаг 1. Перейти по [ссылке](#) и скачать последнюю версию библиотеки (см. рис. 1).



Рис. 1. Ссылка для скачивания

Шаг 2. Создать папку для хранения файлов библиотеки тестирования с названием «catch2» (см. рис 2).

Имя	Дата изменения	Тип
catch2	22.11.2023 4:54	Папка с файлами
DataStructuresAndAlgorithms	01.11.2023 4:56	Папка с файлами
x64	01.11.2023 4:42	Папка с файлами
.gitattributes	01.11.2023 4:43	Исходный файл Git ...
.gitignore	01.11.2023 4:47	Исходный файл Git I...
DataStructuresAndAlgorithms.sln	01.11.2023 4:40	Visual Studio Solution

Рис. 2. Создание папки для хранения библиотеки

Шаг 3. Скопировать скаченные файлы библиотеки тестирования в созданную папку «catch2» (см. рис. 3).

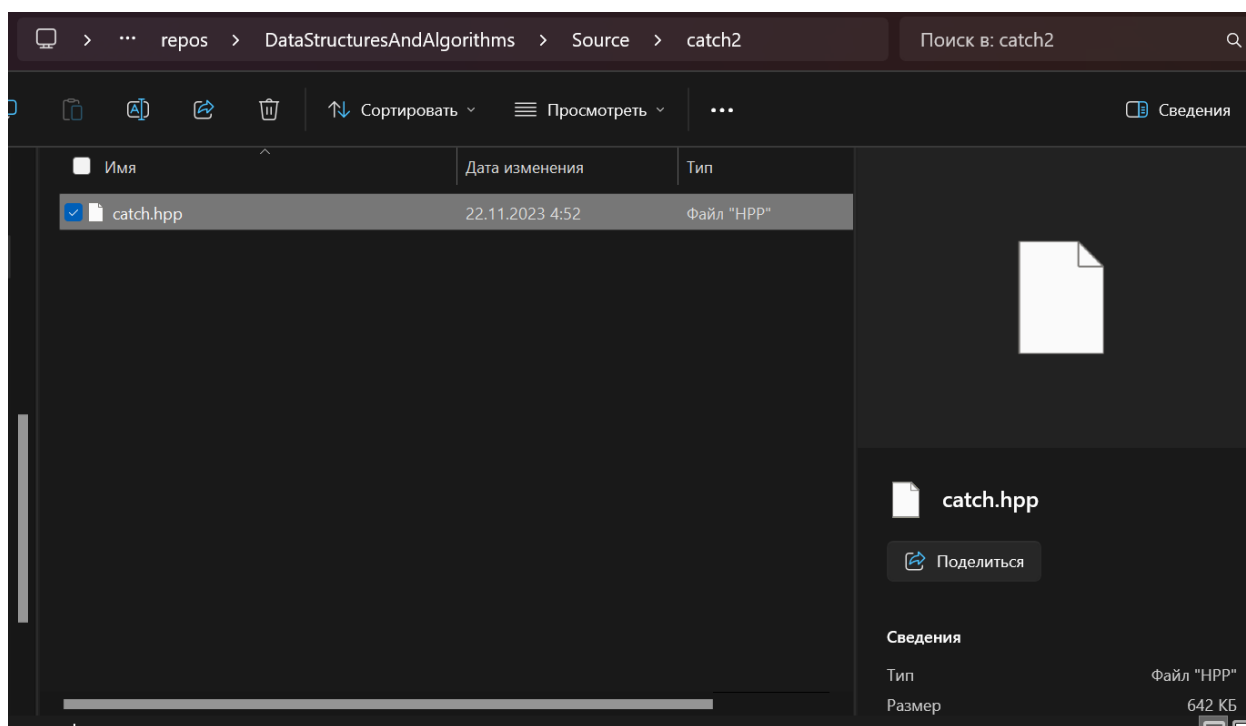


Рис. 3. Копирование файлов библиотеки

Шаг 4. Добавление заголовочного файла библиотеки «catch2» в проект – открытие страницы свойств проекта. Тестовый программный модуль будет находится в новом проекте, для этого необходимо создать новый консольный проект C++ с названием «Test». ПКМ кликнуть по названию проекта (см. рис 4), в открывшемся меню выбрать «Свойства» (см. рис. 5), откроется окно «Страницы свойств ...» (см. рис 6).

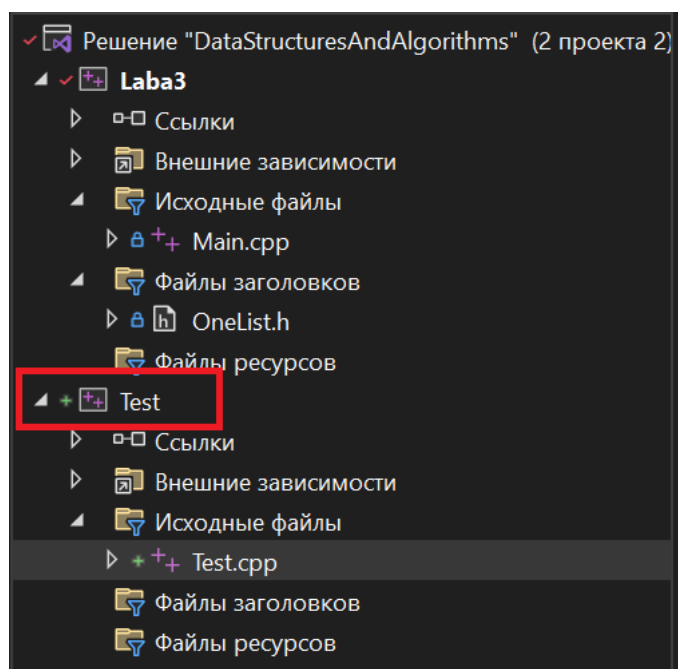


Рис. 4. Свойства проекта

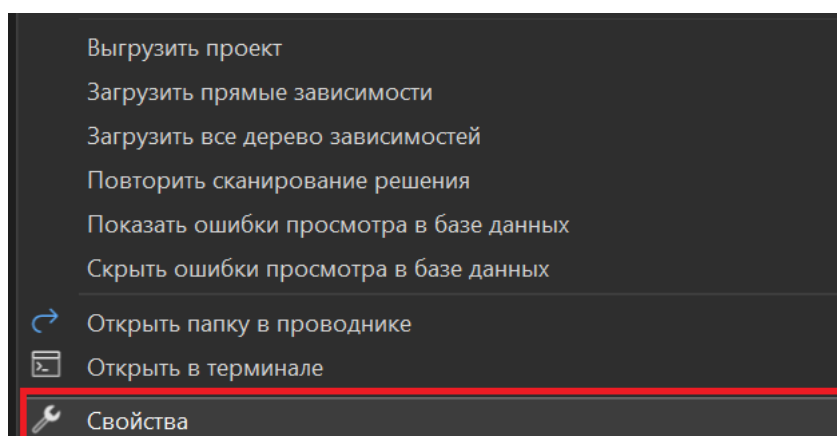


Рис. 5. Свойства проекта

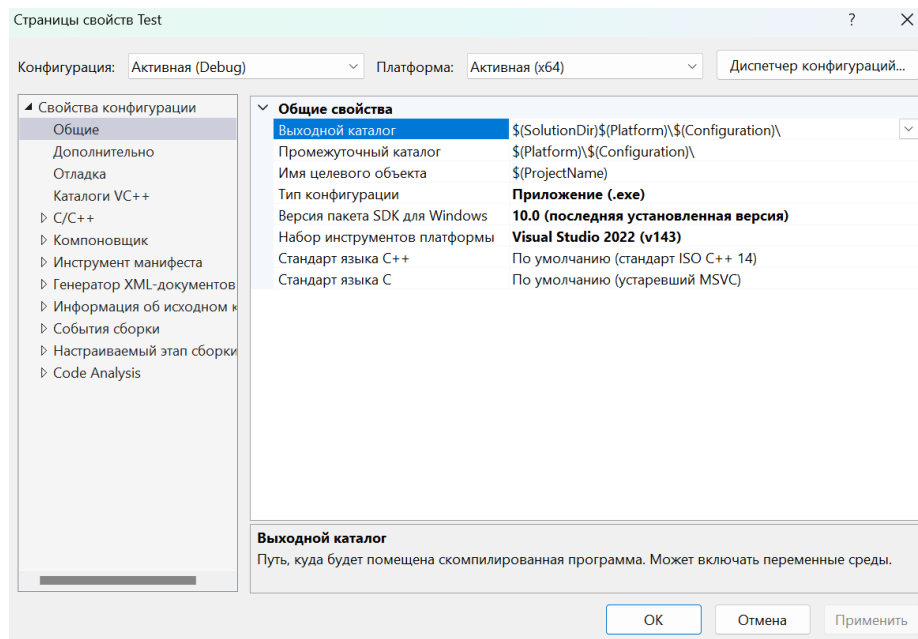


Рис. 6. Страницы свойств проекта

Шаг 5. Добавление заголовочного файла библиотеки «catch2» в проект – добавление библиотеки. В «Свойства конфигурации» открыть выпадающий список «C/C++», выбрать пункт «Общие». В графе «Дополнительные каталоги включаемых файлов» указать файл с библиотекой тестирования (см. рис. 7).

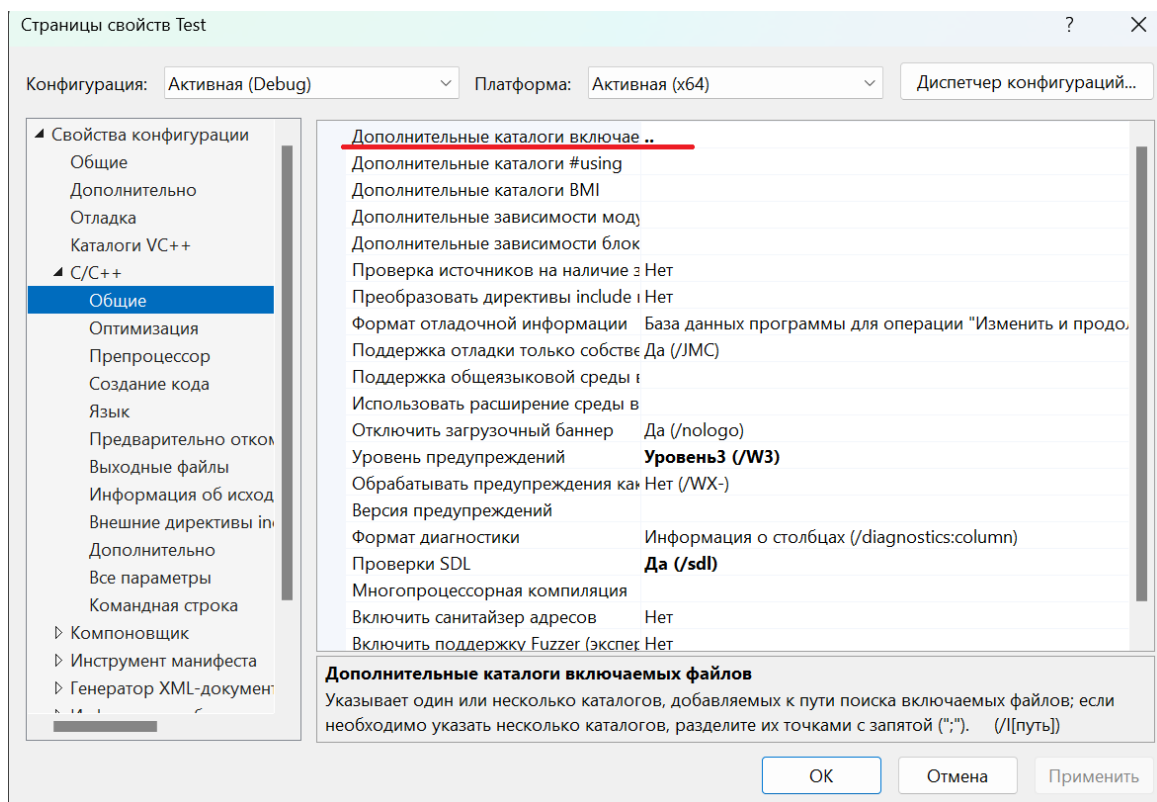


Рис. 7. Указание дополнительных каталогов включаемых файлов

Шаг 6. Подключение библиотеки в файле «test.cpp».

Руководство по использованию библиотеки, а также ее подключению можно найти по ссылке: <https://github.com/catchorg/Catch2/blob/v2.x/docs/tutorial.md>

Для подключения библиотеки тестирования в файл «main.cpp» необходимо добавить следующие строки кода:

```
#define CATCH_CONFIG_MAIN // This tells Catch to provide a main() - only do this in
one cpp file
#include <catch2\catch.hpp>
```

Для запуска проекта с тестами необходимо указать его в качестве запускаемого проекта.

Шаг 7. Пример теста (см. рис. 8).

```
#define CATCH_CONFIG_MAIN // This tells Catch to provide a main() - only do this in
one cpp file
#include <catch2\catch.hpp>

unsigned int Factorial(unsigned int number) {
    return number <= 1 ? number : Factorial(number - 1) * number;
}

TEST_CASE("Factorials are computed", "[factorial]") {
    REQUIRE(Factorial(1) == 1);
    REQUIRE(Factorial(2) == 2);
    REQUIRE(Factorial(3) == 6);
    REQUIRE(Factorial(10) == 3628800);
}
```

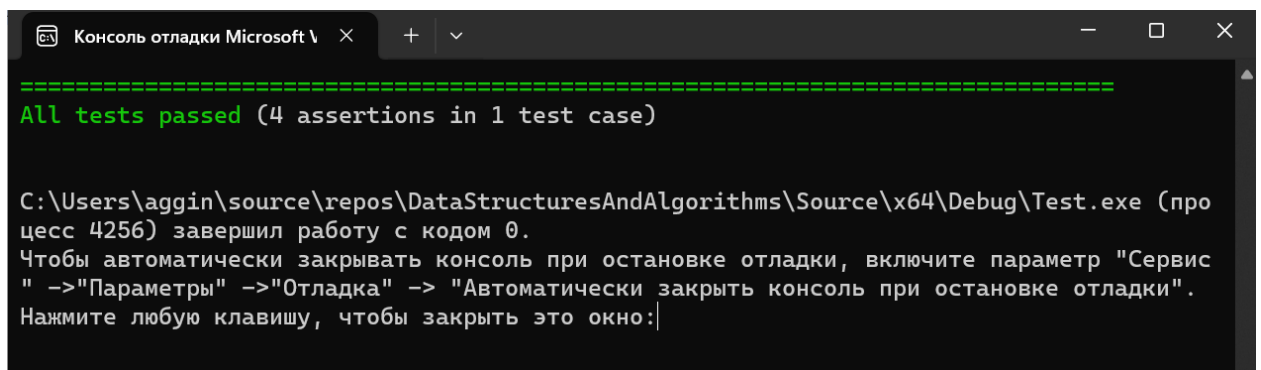


Рис. 8. Запуск первого теста

Настройка автозапуска тестов

Открыть свойства проекта «Test». Выбрать в пункте «Конфигурации» - «Все конфигурации», в пункте «Платформа» - «Все платформы». Далее переходим «Свойства конфигурации» - «События сборки» - «События после сборки» и в строке «командная строка» прописать «\$(TargetPath)», то есть по окончании сборки приложения запуститься проект Test (см. рис. 9).

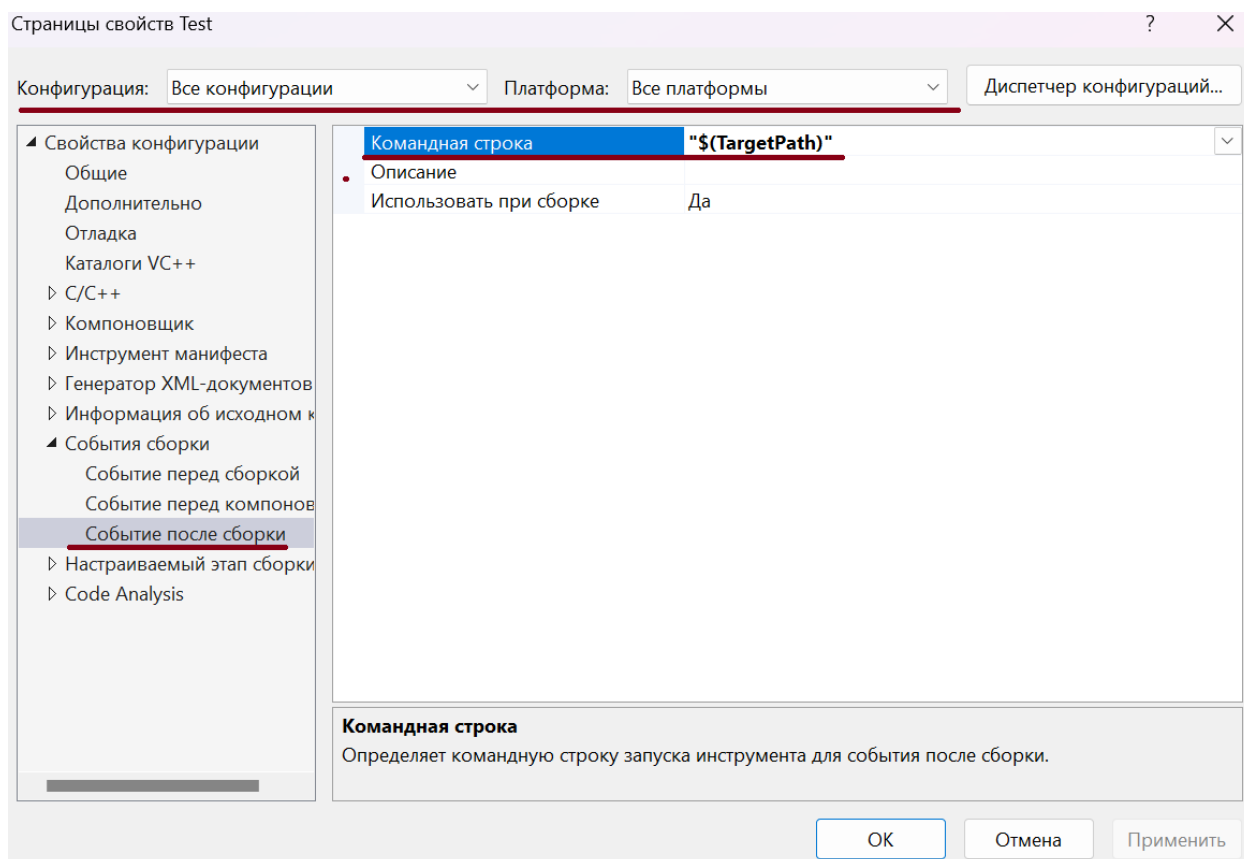
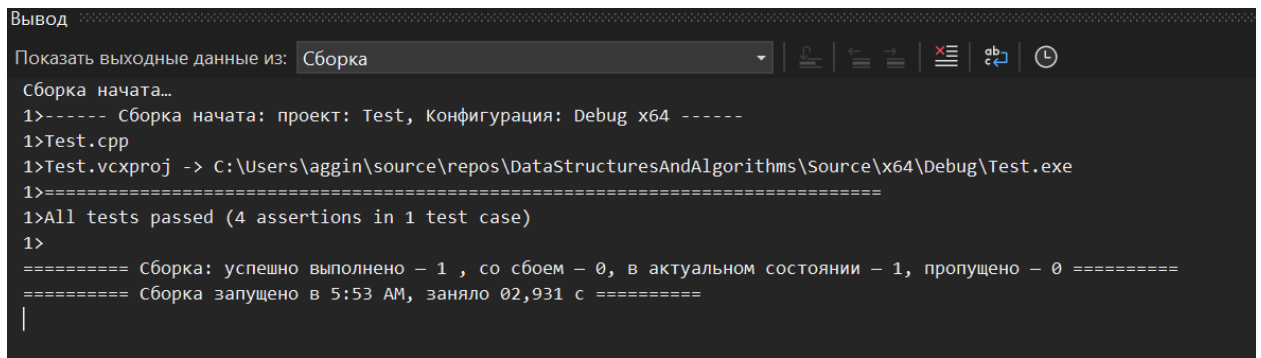


Рис. 9. Настройка автозапуска тестов

После настройки автозапуска тестов можно уже не указывать данные проект в качестве запускаемого.

При сборке приложение тесты пройдут в автоматическом режиме и результаты отобразятся в окне «Вывод» (см. рис. 10).

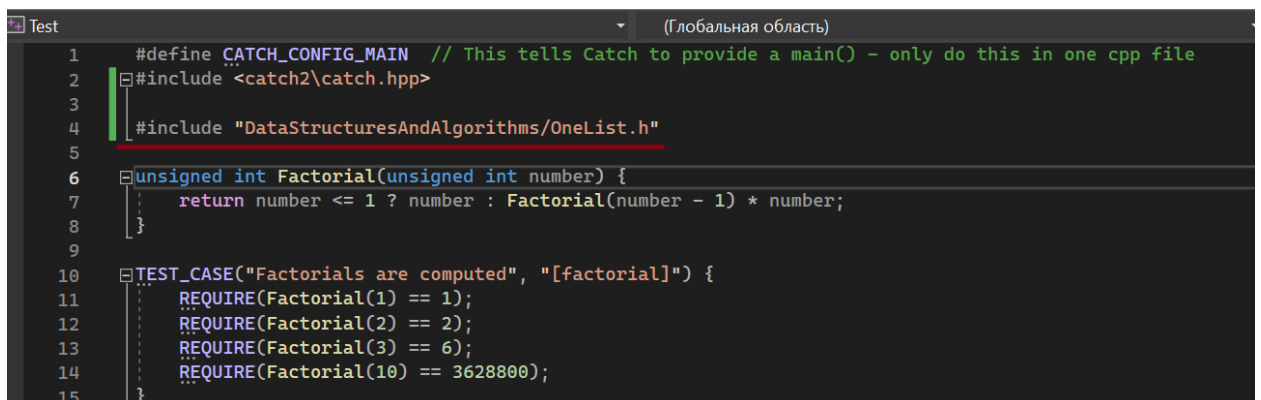


```
Вывод
Показать выходные данные из: Сборка
Сборка начата...
1>----- Сборка начата: проект: Test, Конфигурация: Debug x64 -----
1>Test.cpp
1>Test.vcxproj -> C:\Users\aggin\source\repos\DataStructuresAndAlgorithms\Source\x64\Debug\Test.exe
1>=====
1>All tests passed (4 assertions in 1 test case)
1>
===== Сборка: успешно выполнено – 1 , со сбоями – 0, в актуальном состоянии – 1, пропущено – 0 =====
===== Сборка запущено в 5:53 AM, заняло 02,931 с =====
|
```

Рис. 10. Сборка проекта и автозапуск тестов

В случае ошибки при выполнении теста в окне «Вывод» можно кликнуть по сообщению об ошибке и перейти к тесту, в который не удалось пройти.

Для проверки кода из главного проекта необходимо подключить его библиотечные файлы через include.



```
Test (Глобальная область)
1  #define CATCH_CONFIG_MAIN // This tells Catch to provide a main() - only do this in one cpp file
2  #include <catch2/catch.hpp>
3
4  #include "DataStructuresAndAlgorithms/OneList.h"
5
6  unsigned int Factorial(unsigned int number) {
7      return number <= 1 ? number : Factorial(number - 1) * number;
8  }
9
10 TEST_CASE("Factorials are computed", "[factorial]") {
11     REQUIRE(Factorial(1) == 1);
12     REQUIRE(Factorial(2) == 2);
13     REQUIRE(Factorial(3) == 6);
14     REQUIRE(Factorial(10) == 3628800);
15 }
```


Задание на лабораторную работу

Всю необходимую документацию, а также примеры по использованию библиотеки тестирования «catch2» можно найти по ссылке:

<https://github.com/catchorg/Catch2/tree/v2.x/docs>

В качестве подопытного используем библиотеки классов односвязного и двухсвязного списков, реализованных на лабораторной работе №3.

Чтоб убедиться в корректности библиотек, следует писать тесты, охватывающие различные сценарии для каждой его публичной функции. Ниже приведен набор тестов, которые необходимо реализовать.

Проверка публичных функций библиотек

1. Конструктор (OneList()):

- Проверьте, что вновь созданный список пуст.
- Убедитесь, что указатели **begin** и **end** имеют значение **nullptr**.
- Убедитесь, что счетчик равен нулю.

2. Конструктор копирования (OneList(const OneList& obj)):

- Создайте исходный список с некоторыми элементами.
- Используйте конструктор копирования, чтобы создать новый список.
- Убедитесь, что новый список является полной копией оригинала.
- Убедитесь, что изменение одного списка не влияет на другой.

3. Деструктор (~OneList()):

- Создайте список с некоторыми элементами.
- Удалить список с помощью деструктора.
- Убедитесь, что память правильно освобождена.
- Убедитесь, что список пуст после уничтожения.

4. push_front(void push_front(T data)):

- Добавьте элемент в пустой список и проверьте его **begin** и **end** обновите.
- Добавьте элемент в непустой список и убедитесь, что новый элемент находится в начале.

5. push_back(void push_back(T data)):

- Добавьте элемент в пустой список и проверьте его **begin** и **end** обновите.

- Добавьте элемент в непустой список и убедитесь, что новый элемент находится в конце.
6. **insert (void insert(int index, T item)):**
- Вставьте элемент в начало пустого списка.
 - Вставьте элемент в начало непустого списка.
 - Вставьте элемент в середину списка.
 - Вставьте элемент в конец списка.
 - Убедитесь, что счетчик обновляется правильно после каждой вставки.
7. **pop_front(void pop_front()):**
- Удалите элемент из начала пустого списка (убедитесь, что он не аварийно завершает работу).
 - Удалить элемент из начала списка с одним элементом.
 - Удалить элемент из начала списка с несколькими элементами.
 - Убедитесь, **begin** что **end** они обновляются правильно.
8. **pop_back(void pop_back()):**
- Удалите элемент из конца пустого списка (убедитесь, что он не аварийно завершает работу).
 - Удалить элемент из конца списка с одним элементом.
 - Удалить элемент из конца списка с несколькими элементами.
 - Убедитесь, **begin** что **end** они обновляются правильно.
9. **remove(void remove(T data)):**
- Удалить элемент, который находится в середине списка.
 - Удалить элемент, находящийся в начале списка.
 - Удалить элемент, находящийся в конце списка.
 - Удалить все вхождения определенного значения.
 - Убедитесь, **begin** что **end** они обновляются правильно.
10. **clear (void clear()):**
- Звонок **clear** по пустому списку.
 - Вызов **clear** списка с несколькими элементами.
 - Убедитесь, что список пуст после вызова **clear**.
11. **front(T front()):**
- Получите доступ к переднему элементу пустого списка (убедитесь, что он генерирует исключение).
 - Доступ к переднему элементу непустого списка.
12. **back(T back()):**

- Получите доступ к заднему элементу пустого списка (убедитесь, что он генерирует исключение).
- Доступ к заднему элементу непустого списка.

13. **size(int size()):**

- Проверьте размер пустого списка.
- Проверьте размер списка с несколькими элементами.

14. **operator [](T& operator[](int index)):**

- Доступ к элементам по различным индексам в допустимом диапазоне.
- Убедитесь, что он генерирует исключение при доступе к недопустимому индексу.

15. **Increment and Decrement Operators (operator++и operator--):**

- Протестируйте операторы приращения префикса и постфикса в различных сценариях.
- Протестируйте оператор декремента префикса в различных сценариях.

Проверка работы с памятью

1. **Множественные добавления:**

- Добавьте несколько значений в список, используя различные методы (**push_back**, **push_front**, **insert**).
- После каждого добавления проверяйте правильность размера списка.
- Проверьте, что элементы добавлены в правильном порядке.
- Проверяйте правильность функций **front** и **back** после каждого добавления.
- Убедитесь, что память правильно выделена для каждого добавленного элемента.

2. **Множественные удаления:**

- Выполните несколько удалений, используя различные методы (**pop_back**, **pop_front**, **remove**).
- После каждого удаления проверяйте правильность размера списка.
- Проверьте, что элементы снимаются в правильном порядке.
- Проверяйте правильность функций **front** и **back** после каждого удаления.
- Убедитесь, что память для удаленных элементов правильно освобождена.
- Убедитесь, что список обрабатывает удаление последнего элемента и становится пустым.

3. **Высвобождение ресурсов памяти:**

- Создайте список с несколькими элементами.
- Явно вызывайте деструктор или позволяйте ему вызываться, когда список выходит за пределы области видимости.
- Попробуйте получить доступ к элементам или свойствам списка после уничтожения и убедитесь, что создаются исключения или происходит соответствующее поведение.
- Убедитесь, что память, выделенная для списка, освобождена правильно, проверяя наличие потенциальных утечек памяти.

4. Краевые случаи:

- Протестируйте пустой список, чтобы убедиться, что он работает правильно при добавлении и удалении.
- Добавляйте и удаляйте элементы в начале и конце списка, чтобы охватить граничные случаи.
- Смешивайте добавления и удаления в различных последовательностях, чтобы проверить стабильность списка.

5. Обработка исключений:

- Проверьте наличие исключений при выполнении операций с пустым списком.
- Проверьте наличие исключений при доступе к элементам по недопустимым индексам.
- Проверьте, не возникают ли исключения при попытке удалить несуществующие элементы.

6. Комплексная проверка списка:

- После каждого набора операций (добавление или удаление) проверяйте все состояние списка, используя ожидаемое состояние.
- Сравните фактическое состояние списка с ожидаемым, чтобы убедиться в правильности.

Для проверки множественного добавления и удаления необходимо добавлять в список более 10 млн значений и удалять их, такую операцию необходимо производить несколько раз. Добавление и удаление должно производиться различными способами, при добавлении и удалении необходимо производить контроль значений, например: по определенному индексу должен быть доступен конкретный элемент.