ФЕДЕРАЛЬНОЕ ГОСУДАСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА» (РУТ (МИИТ))

Институт транспортной техники и систем управления
Кафедра «Управление и защита информации»

Отчёт
по практической работе №3
По дисциплине «Языки программирования»

Выполнил: студент группы ТКИ — 241 ИТТСУ
Сапожников С. М.
Вариант №22
Проверила: Доцент
Васильева М. А.

Москва 2023

Файлы содержащиеся в проекте:

Node.h
SingleList.h
SingleList.cpp
main_list.cpp
test.cpp

Формулировка задания

«Написать класс SingleLinkedList»

Код программ:

```
template<typename T>
#pragma once

struct Node
{
    using pointer = std::unique_ptr<Node>;

    T data;
    pointer next = {};
    Node(T value) : data{std::move(value)} {}
};
```

```
#pragma once
#include <iostream>
#include <memory>
#include "Node.h"

template<typename T>
class SingleList
{
    using link_pointer = typename Node<T>::pointer;
    private:
    link_pointer head = {};
    Node<T> *tail = nullptr;
    public:

/**
 * @brief Destroy the Single List object
 *
 */
~SingleList();

/**
 * @brief Destroy the first element of List
 *
 */
void pop();

/**
```

```
 * @brief Inserts an item at the top of the list
 *
 * @param elem
 */
void push_back(T elem);

/**
 * @brief Inserts an item at the back of the list
 *
 * @param elem
 * @return * void
 */
void push_front(T elem);

/**
 * @brief Show all elements of list
 *
 * @return * void
 */
void show() const;

typename Node<T>::pointer get_head();

/**
 * @brief Flips the list
 *
 * @return * void
 */
void reverse();
};
```

Single_list.cpp:

```
#include <memory>
#include "SingleList.h"

template <typename T>
SingleList<T>::~SingleList()
{
   while(head != nullptr)
   {
      head = std::move(head->next);
   }
}

template <typename T>
void SingleList<T>::pop()
{
      if(head == nullptr)
      {
         return;
      }
```

```
        link_pointer new_link = std::move(head);
        head = std::move(new_link->next);
}

template <typename T>
void SingleList<T>::push_back(T elem)
{
    auto new_link = new Node<T>(std::move(elem));
    if (head == nullptr)
    {
        head.reset(new_link);
    }
    if (tail != nullptr)
    {
        tail->next.reset(new_link);
    }
}

template <typename T>
void SingleList<T>::push_front(T elem)
{
    auto new_link = new Node<T>(std::move(elem));

    if (tail != nullptr)
    {
        tail->next.reset(new_link);
    }
    else
    {
        head.reset(new_link);
    }
    tail = new_link;
}

template<typename T>
void SingleList<T>::reverse()
{
    if (head == nullptr)
    {
    return;
    }

    link_pointer prev;
    link_pointer next;

    while (head->next != nullptr)
    {
        next = std::move(head->next);
        head->next = std::move(prev);
        prev = std::move(head);
        head = std::move(next);
    }
```

```
    head->next = std::move(prev);
}

template<typename T>
void SingleList<T>::show() const
{
    auto current = head.get();
    while (current != nullptr)
    {
        std::cout << current->data << std::endl;
        current = current->next.get();
    }
}

template <typename T>
typename Node<T>::pointer SingleList<T>::get_head()
{
    return head;
}
```

main_list.cpp:

```
#include <memory>
#include "SingleList.h"
#include "SingleList.cpp"

int main()
{
    SingleList<int> list;

    list.push_front(1);
    list.push_front(2);
    list.push_front(3);
    list.push_front(4);
    list.push_back(7);

    list.show();
    std::cout << std::endl;

    list.reverse();
    list.show();

    std::cout << std::endl;
    list.pop();
    list.show();
}


#include <cstddef>
#include <gtest/gtest.h>
#include <string>
#include "SingleList.cpp"
```

```
template <typename T> struct Node;

template <typename T> class SingleList;
/**
 * Tests consist of 3 parts
 1) create a object and use some method
 2) create actual and expected values
 3) ASSERT
 *
 */



TEST(ListMetodsTest, PushFrontTest)
{
    SingleList<int> list;
    list.push_front(1);

    std::string actual = list.toString();
    std::string expected = "1";
    ASSERT_FALSE(actual == expected);
}

TEST(ListMetodsTest, PushBackTest)
{
    SingleList<int> list;
    list.push_back(2);

    std::string actual = list.toString();
    std::string expected = "2";
    ASSERT_FALSE(actual == expected);
}

// TEST(ListMetodsTest, PopTest)
// {
//     SingleList<int> list;
//     list.push_front(1);
//     list.push_back(3);

//     std::string actual = list.toString();
//     std::string expected = "12";
//     ASSERT_FALSE(actual == expected);
// }



// TEST(ListMetodsTest, ReverseTest)
// {
//     SingleList<int> list;
//     list.push_front(1);
//     list.push_back(2);
```

```
//     list.reverse();

//     std::string actual = list.toString();
//     std::string expected = "2 1";
//     ASSERT_FALSE(actual == expected);

// }

int main(int argc, char **argv)
{
   testing::InitGoogleTest(&argc, argv);
   return RUN_ALL_TESTS();
}
```

Выполнение тестов:

```
[sergey@sergey-bohkwax9x List_beta]$ ./a.out
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from ListMetodsTest
[ RUN      ] ListMetodsTest.PushFrontTest
[       OK ] ListMetodsTest.PushFrontTest (0 ms)
[ RUN      ] ListMetodsTest.PushBackTest
[       OK ] ListMetodsTest.PushBackTest (0 ms)
[----------] 2 tests from ListMetodsTest (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 2 tests.
```

Рисунок 1 -Выполнение тестов:

UML диаграмма:

Рисунок 2 - UML диаграмма

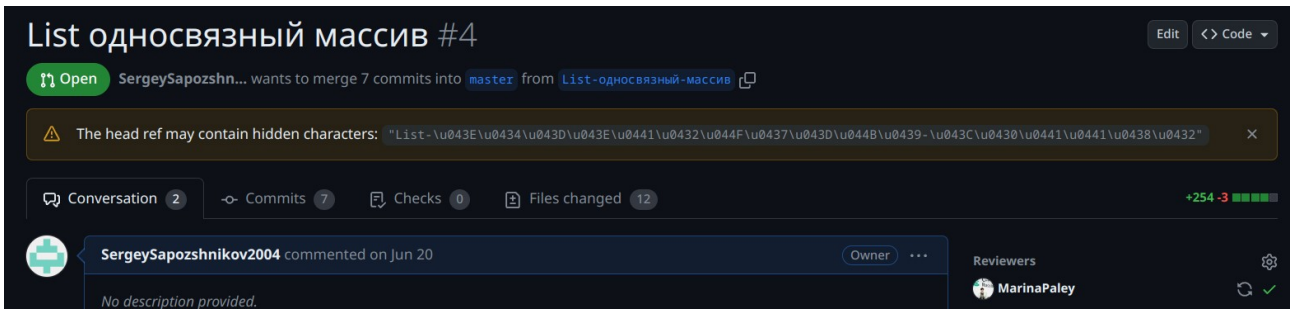

Рисунок 2 - Аппрув