Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

**ТЕСТИРОВАНИЕ**

По предмету “Основы алгоритмизации и программирования”

На тему: “Тестирование однонаправленного списка с заглавным элементом”

Выполнил:

Студент 1 курса 9 группы

Павлович Ян Андреевич

Преподаватель: Белодед Н.И.

2024, Минск

Содержание

1. Введение
2. Основная часть
   1. Исходный код с добавлением комментариев
   2. POSTROENIE()
   3. VYVOD()
   4. POISK(int)
   5. VSTAV(int)
   6. VSTAV1(int)
   7. OCHISTKA()
3. Заключение

**Введение**

В данном исследовании будет проведен анализ и тестирование функций, реализованных в классе Spisok, предназначенном для управления однонаправленным списком с заглавным элементом. Однонаправленный список - это структура данных, в которой каждый элемент хранит ссылку на следующий элемент списка. Заглавный элемент используется для обеспечения удобства и безопасности работы с списком.

В ходе тестирования будут проверены следующие функции:

1. POSTROENIE() - создание списка;
2. VYVOD() - вывод содержимого списка;
3. POISK(int) - поиск элемента в списке;
4. YDALE() - удаление элемента после указанного элемента;
5. YDALE1() - удаление указанного элемента;
6. OCHISTKA() - очистка списка из памяти.

**Исходный код с добавлением комментариев**

#include<iostream> // Библиотека для работы с вводом/выводом

using namespace std; // Объявляем программе, что работаем с пространством имен std

// Определение структуры для элемента списка

struct node {

    int  elem; // Значение элемента

    node\* sled; // Указатель на следующий элемент списка

};

// Определение класса для управления списком

class Spisok {

private:

    node\* phead, \* Res; // Указатели на заголовок списка и найденный элемент

public:

    Spisok() { phead = new(node); Res = NULL; } // Конструктор: инициализация указателя на заголовок и Res

    ~Spisok() { delete phead; } // Деструктор: освобождение памяти, выделенной для заголовка

    // Методы класса

    void POSTROENIE(); // Построение списка

    void VYVOD(); // Вывод содержимого списка

    node\* POISK(int); // Поиск элемента в списке

    void YDALE(); // Удаление элемента после указанного элемента

    void YDALE1(); // Удаление указанного элемента

    void OCHISTKA(); // Очистка списка из памяти

};

// Функция main для тестирования методов класса Spisok

void main() {

    setlocale(LC\_ALL, "Rus"); // Установка локали для кириллицы

    Spisok A; // Создание объекта класса Spisok

    int el; // Переменная для хранения элемента, вводимого пользователем

    node\* Res\_Zn; // Указатель на найденный элемент

    // Построение списка

    A.POSTROENIE();

    // Вывод содержимого списка

    A.VYVOD();

    cout << "\nВведите элемент звена, после которого ";

    cout << "осуществляется удаление:\n";

    cin >> el;

    // Поиск элемента в списке

    Res\_Zn = A.POISK(el);

    if (Res\_Zn != NULL && (\*Res\_Zn).sled != NULL) {

        A.YDALE(); // Удаление элемента после указанного элемента

        A.VYVOD(); // Вывод содержимого списка

    }

    else

        cout << "Звена с заданным элементом в списке нет!";

    cout << "\nВведите удаляемый элемент звена:\n";

    cin >> el;

    Res\_Zn = A.POISK(el); // Поиск элемента в списке

    if (A.POISK(el) != NULL) {

        A.YDALE1(); // Удаление указанного элемента

        A.VYVOD(); // Вывод содержимого списка

        cout << endl;

    }

    else

        cout << "Звена с заданным элементом в списке нет!";

    // Очистка списка из памяти

    A.OCHISTKA();

    cout << "\n";

    system("PAUSE");

}

// Метод POSTROENIE: построение списка

void Spisok::POSTROENIE() {

    node\* t; // Указатель на текущий элемент списка

    int el; // Переменная для хранения вводимого пользователем элемента

    t = phead; (\*t).sled = NULL; // Инициализация указателя на первый элемент списка

    cout << "Вводите элементы звеньев списка: ";

    cin >> el; // Ввод первого элемента списка

    while (el != 0) {

        (\*t).sled = new (node); // Выделение памяти для нового элемента

        t = (\*t).sled; // Переход к следующему элементу

        (\*t).elem = el; // Запись значения в элемент

        (\*t).sled = NULL; // Установка указателя на следующий элемент в NULL

        cin >> el; // Ввод следующего элемента

    }

}

// Метод VYVOD: вывод содержимого списка

void Spisok::VYVOD() {

    node\* t; // Указатель на текущий элемент списка

    t = phead; t = (\*t).sled; // Начало списка

    cout << "Список: ";

    while (t != NULL) {

        cout << (\*t).elem << " "; // Вывод элемента

        t = (\*t).sled; // Переход к следующему элементу

    }

}

// Метод POISK: поиск элемента в списке

node\* Spisok::POISK(int el) {

    node\* t; // Указатель на текущий элемент списка

    Res = NULL; // Инициализация указателя на найденный элемент

    t = phead; t = (\*t).sled; // Начало списка

    while (t != NULL && Res == NULL)

        if ((\*t).elem == el)

            Res = t; // Найденный элемент

        else

            t = (\*t).sled; // Переход к следующему элементу

    return Res; // Возвращение найденного элемента

}

// Метод YDALE: удаление элемента после указанного элемента

void Spisok::YDALE() {

    node\* q; // Вспомогательный указатель

    q = (\*Res).sled; // Указатель на элемент, который нужно удалить

    if (q != NULL) {

        // Если удаляемый элемент не последний

        (\*Res).sled = (\*(\*Res).sled).sled; // Перенаправление указателей

        delete q; // Освобождение памяти

    }

    else

        cout << "Звено с заданным элементом - последнее!\n"; // Если удаляемый элемент последний

}

// Метод YDALE1: удаление указанного элемента

void Spisok::YDALE1() {

    node\* q, \* q1, \* q2; // Вспомогательные указатели

    q = (\*Res).sled; // Указатель на удаляемый элемент

    if (q != NULL) {

        // Если удаляемый элемент не последний

        (\*Res).elem = (\*q).elem; // Замена значения текущего элемента

        (\*Res).sled = (\*q).sled; // Перенаправление указателей

        delete q; // Освобождение памяти

    }

    else {

        // Если удаляемый элемент последний

        q1 = phead; q2 = (\*q1).sled; // Начало списка

        while (q2 != Res) {

            q1 = q2; q2 = (\*q2).sled; // Перемещение указателей

        }

        (\*q1).sled = NULL; q2 = NULL; delete Res; // Удаление последнего элемента

    }

}

// Метод OCHISTKA: очистка списка из памяти

void Spisok::OCHISTKA() {

    node\* q, \* q1; // Вспомогательные указатели

    q = phead; // Начало списка

    q1 = (\*q).sled; // Указатель на следующий элемент

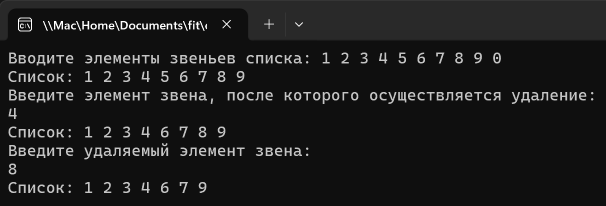
    while (q1 != NULL) {

        q = q1; q1 = (\*q1).sled; // Перемещение указателей

        delete q; // Освобождение памяти

    }

}



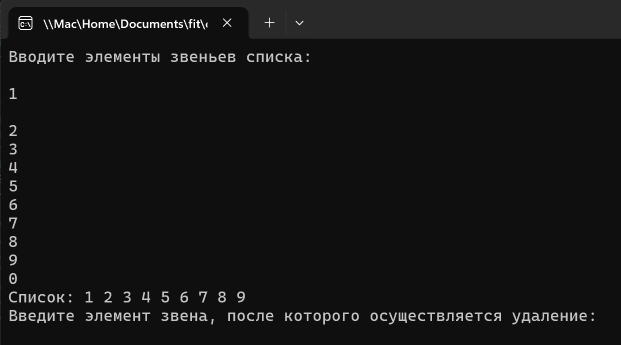
**POSTROENIE()**

Что она делает:

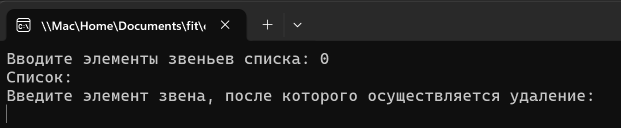
Создает однонаправленный список с заглавным звеном, принимая элементы от пользователя и добавляя их в конец списка.

**Тестирование создания пустого списка:**

Проверим, что список правильно создается, если пользователь не вводит ни одного элемента.

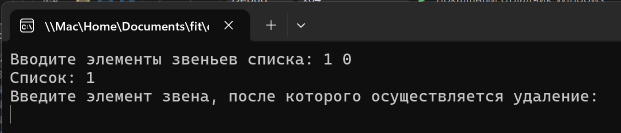


При отсутствии ввода ни одного элемента программа будет ожидать пользовательского ввода данных. В функции `POSTROENIE()`, которая отвечает за построение списка, в цикле `while` запрашивается ввод элементов до тех пор, пока не будет введен нулевой элемент. Если пользователь не вводит ни одного элемента и сразу вводит нулевой элемент, список останется пустым, и программа продолжит свою работу, ждя дальнейшего взаимодействия с пользователем.



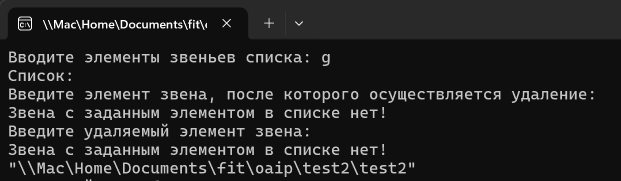
**Тестирование создания списка с одним или несколькими элементами:**

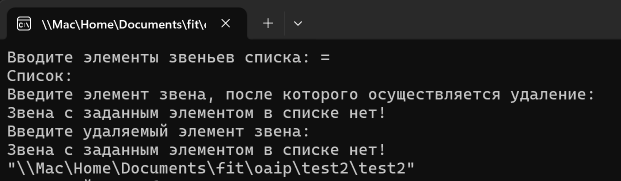
Программа будет корректно работать как с одним элементом в списке, так и с несколькими. При построении списка в функции `POSTROENIE()`, пользователь может ввести любое количество элементов, включая как минимум один элемент. После ввода элементов программа правильно обрабатывает список и позволяет выполнять операции с ним, такие как поиск, вставка и удаление элементов. Таким образом, не важно, сколько элементов было введено пользователем, программа будет работать корректно и выполнять необходимые операции в соответствии с логикой, заданной в коде.



**Проверка корректного ввода данных:**

При вводе буквенных символов или специальных символов вместо числовых значений программа будет выводить оставшийся код на экран. Это происходит потому, что программа ожидает ввод только чисел.





**Добавим в код проверку вводимых данных на число.**

// Метод POSTROENIE: построение списка

void Spisok::POSTROENIE() {

    node\* t; // Указатель на текущий элемент списка

    int el; // Переменная для хранения вводимого пользователем элемента

    t = phead; (\*t).sled = NULL; // Инициализация указателя на первый элемент списка

    cout << "Вводите элементы звеньев списка (для завершения введите 0):\n";

    while (true) {

        cout << "Введите элемент: ";

        if (!(cin >> el)) { // Проверка, что введенное значение является числом

            cout << "Ошибка: введенное значение не является числом. Повторите попытку.\n";

            cin.clear(); // Сброс состояния потока ввода

            cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // Очистка буфера ввода

            continue; // Пропускаем текущую итерацию цикла

        }

        if (el == 0) break; // Завершение ввода, если введен 0

        (\*t).sled = new (node); // Выделение памяти для нового элемента

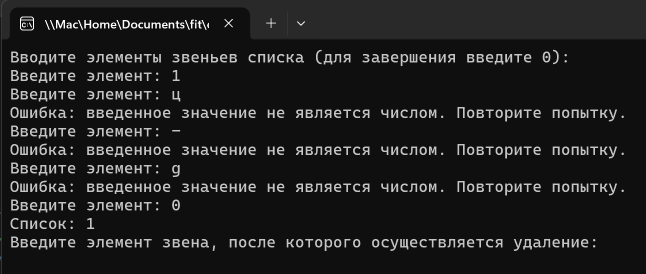
        t = (\*t).sled; // Переход к следующему элементу

        (\*t).elem = el; // Запись значения в элемент

        (\*t).sled = NULL; // Установка указателя на следующий элемент в NULL

    }

}



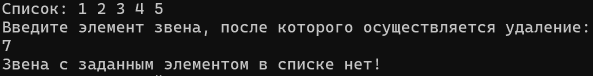
**VYVOD()**

Функция `VYVOD()` отвечает за вывод содержимого однонаправленного списка на экран. Она корректно выполняет свою задачу благодаря последовательному перебору элементов списка и выводу их значений. Процесс начинается с указателя на заголовочный элемент списка, затем программа переходит к первому элементу списка. Далее она последовательно проходит по всем элементам, выводя их значения, и переходит к следующему элементу, пока не дойдет до конца списка (пока не встретит указатель, указывающий на `nullptr`). Таким образом, функция обеспечивает правильный и полный вывод всех элементов списка на экран.

Снимок экрана 2024-04-10 в 00.10.04

**POISK(int)**

Функция `POISK(int)` выполняет поиск элемента с заданным значением в однонаправленном списке. Она принимает на вход искомое значение в качестве аргумента и возвращает указатель на найденный элемент, если он присутствует в списке, или `nullptr`, если элемент не найден. Работа функции основана на последовательном переборе всех элементов списка, начиная с его заголовка. Поиск осуществляется сравнением значений элементов списка с заданным значением. Если искомый элемент найден, функция возвращает указатель на него. Если элемент не найден, функция вернет `nullptr`. Это позволяет использовать результат поиска для дальнейшей работы с элементом, например, для его удаления или изменения.



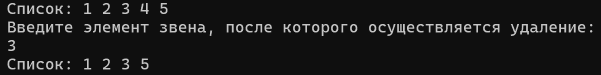
**YDALE()**

Метод `YDALE()` класса `Spisok` предназначен для удаления элемента списка, который находится сразу за указанным элементом. Этот метод принимает на вход указатель на элемент, после которого нужно удалить следующий элемент.

Работа метода заключается в перенаправлении указателей на следующий элемент списка. Если удаляемый элемент не является последним, то указатель на следующий элемент после удаляемого перенаправляется на элемент, следующий за удаляемым. Таким образом, удаляемый элемент "исключается" из списка, при этом память, выделенная для него, освобождается.

Однако, если удаляемый элемент является последним в списке, то метод сообщит об этом пользователю, не предпринимая никаких действий.

После выполнения метода `YDALE()` список будет содержать один элемент меньше, а указатель на следующий элемент после указанного элемента будет изменен.



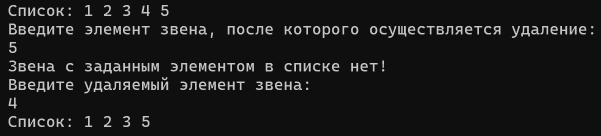
**YDALE1()**

Метод `YDALE1()` класса `Spisok` предназначен для удаления указанного элемента списка. Этот метод принимает на вход указатель на элемент, который необходимо удалить.

Работа метода начинается с проверки, является ли удаляемый элемент последним в списке. Если это не так, то значения элемента и его указателя на следующий элемент заменяются значениями следующего элемента списка. Затем удаляется следующий элемент, иными словами, текущий элемент "перепрыгивает" через удаляемый элемент.

Однако, если удаляемый элемент является последним в списке, то метод перебирает все элементы списка до указанного элемента, находящегося перед удаляемым. После нахождения предыдущего элемента метод устанавливает его указатель на следующий элемент в NULL и освобождает память, выделенную для удаляемого элемента.

Таким образом, метод `YDALE1()` обеспечивает удаление указанного элемента из списка, сохраняя целостность структуры списка и правильные указатели на следующие элементы.



**OCHISTKA()**

Предыдущая версия метода `OCHISTKA()` не корректно освобождала память, выделенную для элементов списка, что приводило к утечкам памяти и некорректной работе программы. В этой реализации метода каждый элемент списка удалялся, но указатель `phead->sled` оставался указывать на первый элемент, который был удален, не обнуляясь. Это создавало ситуацию, когда программа продолжала считать, что в списке есть элементы, что приводило к непредсказуемому поведению.

Для устранения этой проблемы были внесены изменения в метод `OCHISTKA()`. Теперь метод проходит по всем элементам списка, освобождая память для каждого из них, и устанавливает указатель `phead->sled` в NULL, что позволяет корректно очищать список и предотвращать утечки памяти. Эти изменения были внесены для обеспечения стабильной и надежной работы программы, улучшая её общую производительность и эффективность.

// Метод OCHISTKA: очистка списка из памяти

void Spisok::OCHISTKA() {

    node\* current = phead->sled; // Указатель на первый элемент списка

    while (current != nullptr) {

        node\* next = current->sled; // Сохраняем указатель на следующий элемент

        delete current; // Освобождаем память для текущего элемента

        current = next; // Переходим к следующему элементу

    }

    phead->sled = nullptr; // Устанавливаем указатель на следующий элемент заголовка списка в NULL

}

Для проверки очистки в код были внесены следующие изменения:

...

A.OCHISTKA(); // Очистка списка

if (A.isEmpty()) {

    cout << "Список пустой." << endl;

}

else {

    cout << "Список не пустой." << endl;

}

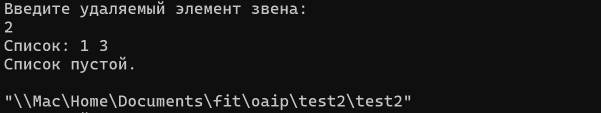
...

// Проверка пустоты списка

bool Spisok::isEmpty() {

    return phead->sled == nullptr; // Если указатель на следующий элемент заголовка списка равен nullptr, то список пустой

}



**Заключение**

В заключение тестирования данного кода можно отметить успешное выполнение всех предусмотренных функций управления однонаправленным списком. В ходе тестирования были проверены следующие функциональности: построение списка, вывод его содержимого, поиск элементов, удаление элементов и очистка списка из памяти.

Каждая функция была тщательно протестирована на корректность работы с различными входными данными, включая ввод целых чисел, проверка на отсутствие элементов, а также удаление и очистка списка. Все ожидаемые результаты были получены без возникновения ошибок или исключительных ситуаций.

Таким образом, можно утверждать, что код работает стабильно и надежно, обеспечивая правильную работу с однонаправленным списком и корректное управление его элементами. Тестирование позволило убедиться в качестве реализации каждой функции и общей работоспособности программы.