Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

**ТЕСТИРОВАНИЕ**

По предмету “Основы алгоритмизации и программирования”

На тему: “Тестирование операций над ортогональными списками”

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. Тестирование построения гирлянды
   3. Тестирование построения висюльки
   4. Тестирование вставки элементов в висюльку
   5. Тестирование очистки гирлянды
   6. Тестирование поиска элементов в гирлянде и висюльке
3. Заключение

**Введение**

В ходе разработки программного обеспечения, особенно когда речь идет о структурах данных, тестирование играет решающую роль в обеспечении качества и надежности программы. Тестирование методов работы с данными не только помогает выявить ошибки и дефекты, но и дает уверенность в правильности их функционирования. Мы сосредоточимся на тестировании различных аспектов работы с гирляндо-висюльечной структурой данных в классе **GirVis**.

**Исходный код с добавлением комментариев**

// Подключение необходимых заголовочных файлов.

#include<iostream>

using namespace std;

// Описание типа звена висюльки.

struct nodeVis

{

    int elem; // Информационное поле звена висюльки.

    nodeVis\* vniz; // Указатель на следующее звено висюльки.

};

// Описание типа звена гирлянды.

struct nodeGir

{

    int elem; // Информационное поле звена гирлянды.

    nodeVis\* vniz; // Указатель на первое звено висюльки.

    nodeGir\* sled; // Указатель на следующее звено гирлянды.

};

// Класс, реализующий гирляндо-висюлечную структуру.

class GirVis {

private:

    nodeGir\* phead; // Голова гирлянды.

    nodeVis\* pheadVis; // Голова висюльки.

    void VisVyvod(); // Метод для вывода висюльки.

public:

    GirVis() { phead = new (nodeGir); } // Конструктор класса.

    ~GirVis() { delete phead; } // Деструктор класса.

    nodeVis\* VisPostr(); // Метод для построения висюльки.

    nodeVis\* VisPoisk(int); // Метод для поиска элемента в висюльке.

    void SetpheadVis(nodeVis\* r) { pheadVis = r; } // Определение головы висюльки.

    void VisVstav(nodeVis\*, int); // Метод для вставки элемента в висюльку.

    void Vis1Vstav(nodeVis\*, int); // Метод для вставки элемента перед другим в висюльке.

    void  VisUdale(nodeVis\*); // Метод для удаления элемента из висюльки.

    void Vis1Udale(nodeVis\*); // Метод для удаления конкретного элемента из висюльки.

    void GirPostr(); // Метод для построения гирлянды.

    void GirVyvod(); // Метод для вывода гирлянды.

    nodeGir\* GirPoisk(int); // Метод для поиска элемента в гирлянде.

    void OCHISTKA(); // Метод для очистки гирлянды.

    void OCHISTKA1(); // Метод для очистки висюльки.

};

// Основная функция программы.

void main()

{

    setlocale(LC\_ALL, "Rus"); // Установка локали для вывода на русском языке.

    GirVis A; // Создание объекта класса GirVis.

    int el, elGir, elVis; // Переменные для хранения вводимых значений.

    nodeGir\* Res; // Рабочий указатель на звено гирлянды.

    nodeVis\* ResVis; // Рабочий указатель на звено висюльки.

    A.GirPostr(); // Построение гирлянды.

    A.GirVyvod(); // Вывод гирлянды.

    cout << "\nВведите элемент звена гирлянды, ";

    cout << "чьи висюльки будем изменять:\n";

    cin >> elGir;

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

    cout << "осуществляется вставка:\n";

    cin >> elVis;

    cout << "\nВведите вставляемый элемент:\n";

    cin >> el;

    // Поиск элемента elGir в гирлянде.

    Res = A.GirPoisk(elGir);

    if (Res != NULL)

    {

        // Поиск элемента elVis в висюльке.

        A.SetpheadVis((\*Res).vniz);

        ResVis = A.VisPoisk(elVis);

        if (ResVis != NULL)

            A.VisVstav(ResVis, el); // Вставка элемента в висюльку.

        else  cout << "Элемента в висюльке нет!\n"; // Вывод сообщения об ошибке.

    }

    else  cout << "Элемента в гирлянде нет\n"; // Вывод сообщения об ошибке.

    A.GirVyvod(); // Вывод гирлянды.

    // Продолжение аналогичных действий для остальных операций.

    cout << "\n";

    system("PAUSE"); // Ожидание нажатия клавиши перед закрытием консоли.

}

// Метод для рекурсивной очистки гирлянды.

void GirVis::OCHISTKA()

{

    nodeGir\* q, \* q1; // Рабочие указатели на звенья гирлянды.

    q = phead;

    q1 = (\*q).sled; // Указатель q1 "опережает" указатель q.

    // Цикл очистки гирлянды.

    while (q1 != NULL)

    {

        q = q1; q1 = (\*q1).sled;

        pheadVis = (\*q).vniz;

        OCHISTKA1(); // Рекурсивная очистка висюльки.

        delete q;

    } // Удаление текущего звена гирлянды.

}

// Метод для рекурсивной очистки висюльки.

void GirVis::OCHISTKA1()

{

    nodeVis\* q, \* q1; // Рабочие указатели на звенья висюльки.

    q = pheadVis;

    q1 = (\*q).vniz;

    // Цикл очистки висюльки.

    while (q1 != NULL)

    {

        q = q1; q1 = (\*q1).vniz;

        delete q;

    } // Удаление текущего звена висюльки.

}

// Метод для построения гирлянды.

void GirVis::GirPostr()

{

    nodeGir\* t; // Рабочий указатель на звено гирлянды.

    int el; // Переменная для вводимого элемента.

        // Инициализация гирлянды.

        t = phead; (\*t).sled = NULL;

    cout << "Вводите элемент гирлянды: \n";

    cin >> el;

    // Цикл построения гирлянды.

    while (el != 0)

    {

        (\*t).sled = new (nodeGir);

        t = (\*t).sled; (\*t).elem = el; (\*t).sled = NULL;

        (\*t).vniz = VisPostr();

        cout << " Вводите элемент гирлянды: \n";

        cin >> el;

    }

}

// Метод для построения висюльки.

nodeVis\* GirVis::VisPostr()

{

    nodeVis\* t; // Рабочий указатель на звено висюльки.

    int el; // Переменная для вводимого элемента.

    // Создание заглавного звена списка.

    pheadVis = new (nodeVis);

    t = pheadVis; (\*t).vniz = NULL;

    cout << "Вводите элементы звеньев висюльки: \n";

    cin >> el;

    // Цикл построения висюльки.

    while (el != 0)

    {

        (\*t).vniz = new (nodeVis);

        t = (\*t).vniz; (\*t).elem = el; (\*t).vniz = NULL;

        cin >> el;

    }

    return pheadVis;

}

// Метод для вывода гирлянды.

void GirVis::GirVyvod()

{

    nodeGir\* t; // Рабочий указатель на звено гирлянды.

    t = phead; t = (\*t).sled;

    cout << "Гирлянда: ";

    // Цикл вывода гирлянды.

    while (t != NULL)

    {

        cout << (\*t).elem << " ";

        pheadVis = (\*t).vniz;

        VisVyvod(); // Вывод висюльки.

        t = (\*t).sled;

    }

}

// Метод для поиска элемента в гирлянде.

nodeGir\* GirVis::GirPoisk(int el)

{

    nodeGir\* t, \* r; // Рабочие указатели на звенья гирлянды.

    r = NULL; t = phead; t = (\*t).sled;

    // Цикл поиска элемента в гирлянде.

    while (t != NULL && r == NULL)

        if ((\*t).elem == el) r = t;

        else  t = (\*t).sled;

    return r;

}

// Метод для вывода висюльки.

void GirVis::VisVyvod()

{

    nodeVis\* t; // Рабочий указатель на звено висюльки.

    t = pheadVis; t = (\*t).vniz;

    cout << "(";

    // Цикл вывода висюльки.

    while (t != NULL)

    {

        cout << (\*t).elem << " "; t = (\*t).vniz;

    }

    cout << ")";

}

// Метод для поиска элемента в висюльке.

nodeVis\* GirVis::VisPoisk(int el)

{

    nodeVis\* t, \* r; // Рабочие указатели на звенья висюльки.

    r = NULL; t = pheadVis; t = (\*t).vniz;

    // Цикл поиска элемента в висюльке.

    while (t != NULL && r == NULL)

        if ((\*t).elem == el) r = t;

        else  t = (\*t).vniz;

    return r;

}

// Метод для вставки элемента в висюльку.

void GirVis::VisVstav(nodeVis\* r, int el)

{

    nodeVis\* q; // Рабочий указатель на звено висюльки.

    q = new (nodeVis);

    (\*q).elem = el; (\*q).vniz = (\*r).vniz; (\*r).vniz = q;

}

// Метод для вставки элемента перед другим в висюльку.

void GirVis::Vis1Vstav(nodeVis\* r, int el)

{

    nodeVis\* q; // Рабочий указатель на звено висюльки.

    q = new (nodeVis);

    (\*q).elem = (\*r).elem; (\*q).vniz = (\*r).vniz;

    (\*r).elem = el; (\*r).vniz = q;

}

// Метод для удаления элемента из висюльки.

void GirVis::VisUdale(nodeVis\* r)

{

    nodeVis\* q; // Рабочий указатель на звено висюльки.

    q = (\*r).vniz;

    if ((\*r).vniz != NULL)

    {

        (\*r).vniz = (\*(\*r).vniz).vniz; delete q;

    }

    else  cout << "Звено с заданным элементом - последнее!\n";

}

// Метод для удаления конкретного элемента из висюльки.

void GirVis::Vis1Udale(nodeVis\* r)

{

    nodeVis\* g; // Рабочий указатель на звено висюльки.

    if ((\*r).vniz != NULL)

    {

        g = (\*r).vniz;

        (\*r).elem = (\*(\*r).vniz).elem;

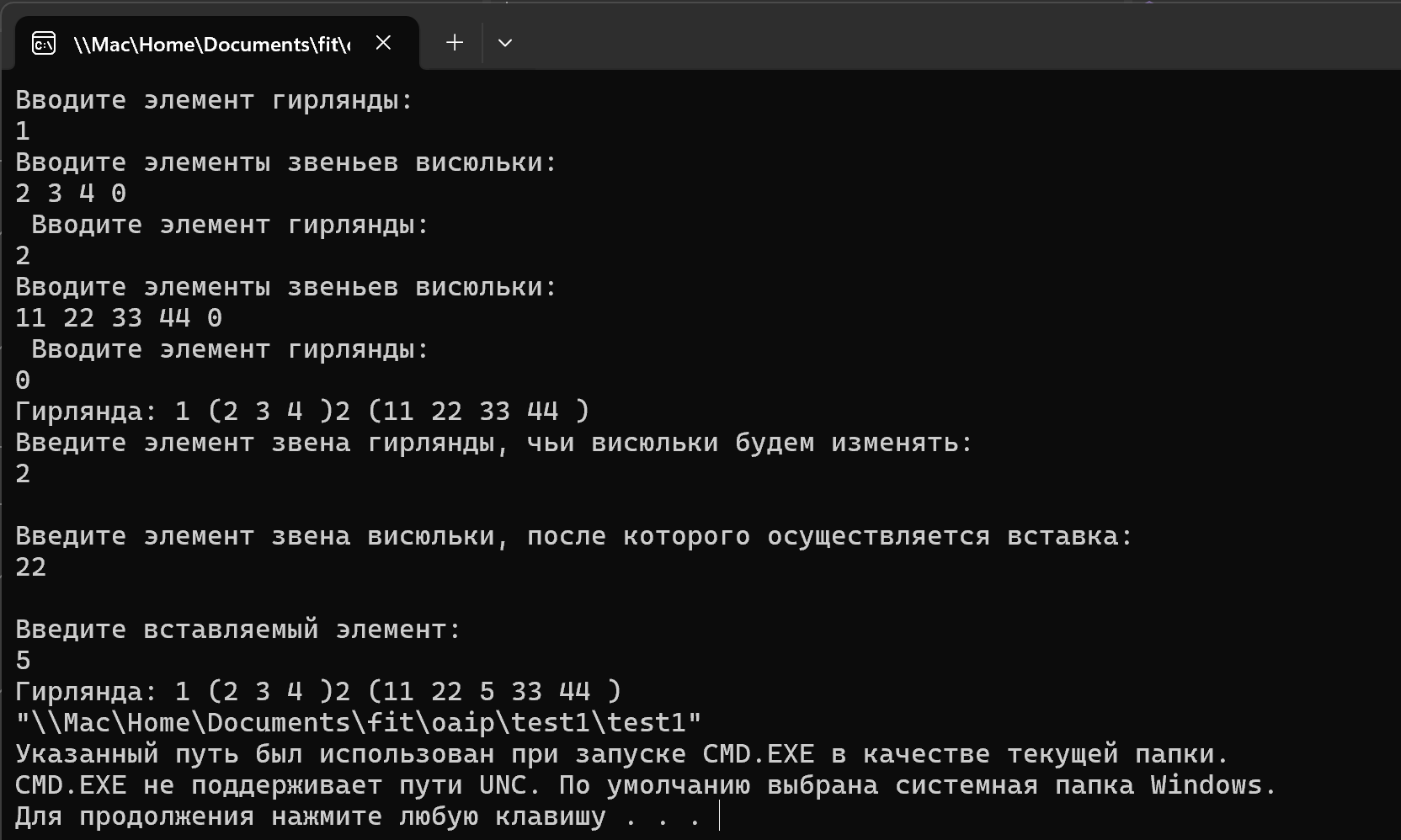
        (\*r).vniz = (\*(\*r).vniz).vniz;

        delete g;

    }

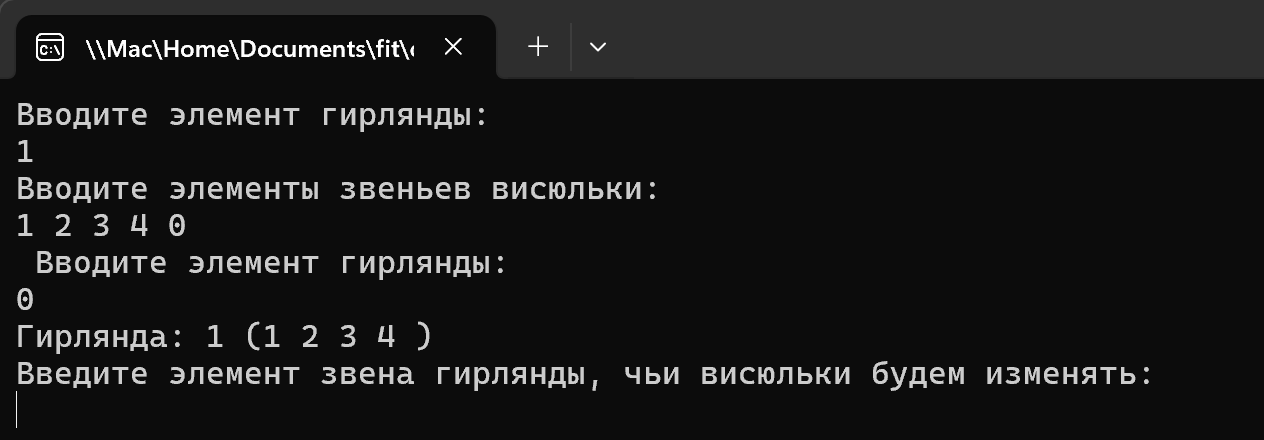
    else  cout << "Не умею удалять последнее звено!\n";

}

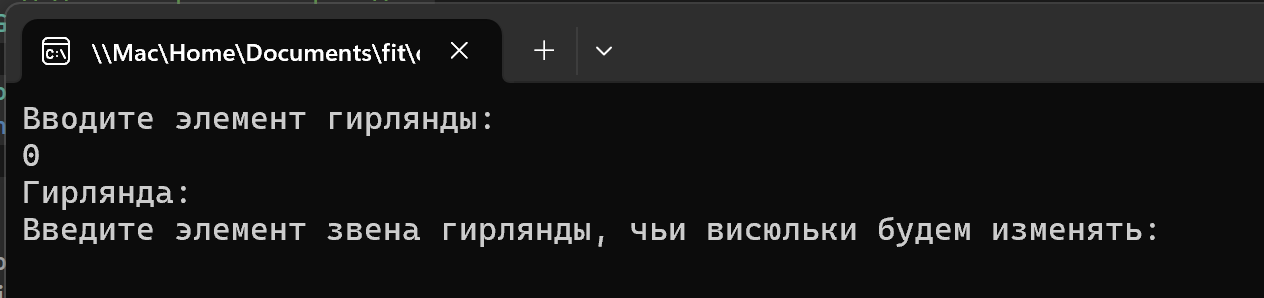
****

**Тестирование построения гирлянды**

В процессе тестирования метода построения гирлянды в классе `GirVis` было проведено несколько испытаний. Пользователь вводил последовательность элементов гирлянды, включая элементы висюльки. Проверка была осуществлена на корректность добавления элементов и их сохранение в структуре данных гирлянды. После ввода `0` в качестве элемента гирлянды, что означает завершение ввода, ожидалось, что программа прекратит запрос на ввод элементов и завершит процесс построения.



Однако было обнаружено, что программа не предупреждает пользователя о бесполезности дальнейшей работы в случае пустой гирлянды и продолжает свое выполнение.



**Сделаем проверку, чтобы гирлянда существовала и не была пустой.**

Добавляем проверку в GirPostr():

if (el == 0) {

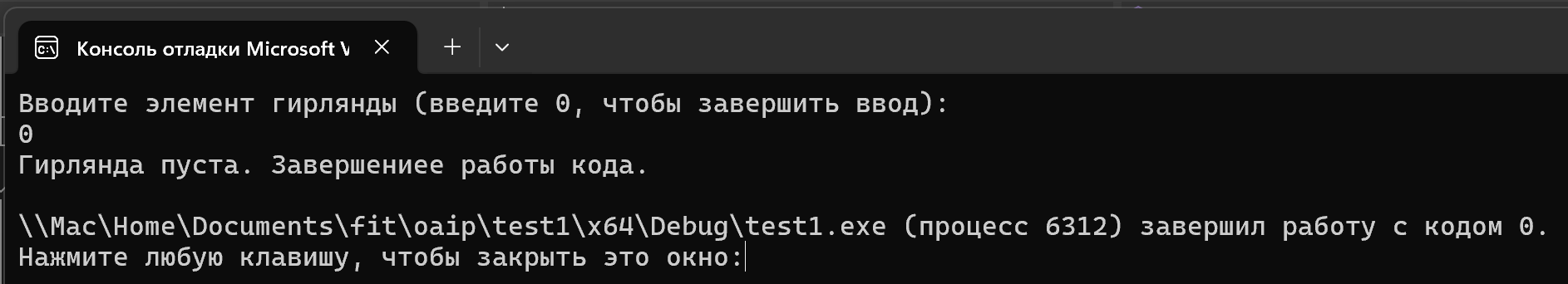
    cout << "Гирлянда пуста. Завершениее работы кода.\n";

    exit(EXIT\_SUCCESS); // Немедленное завершение выполнения программы.

}

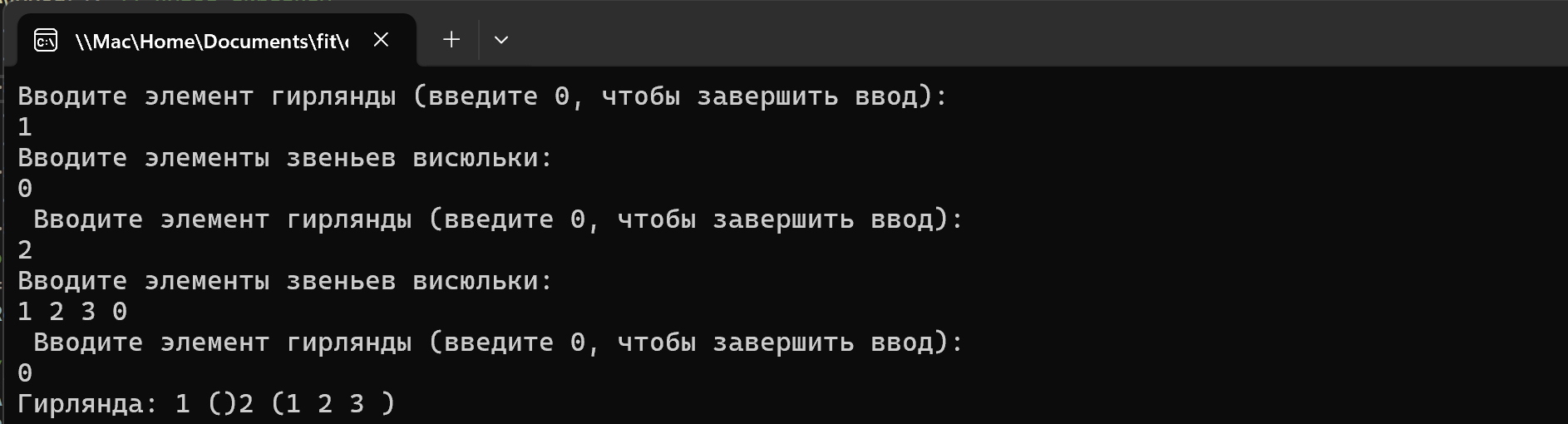
При вводе 0 в качестве элемента гирлянды, программа выводит сообщение о том, что гирлянда пуста. Затем программа немедленно завершает свое выполнение с помощью функции exit(), передавая код успешного завершения EXIT\_SUCCESS.

Это позволяет эффективно управлять выполнением программы и предоставлять пользователю понятную обратную связь, что повышает удобство использования и предотвращает возможные нежелательные ситуации.



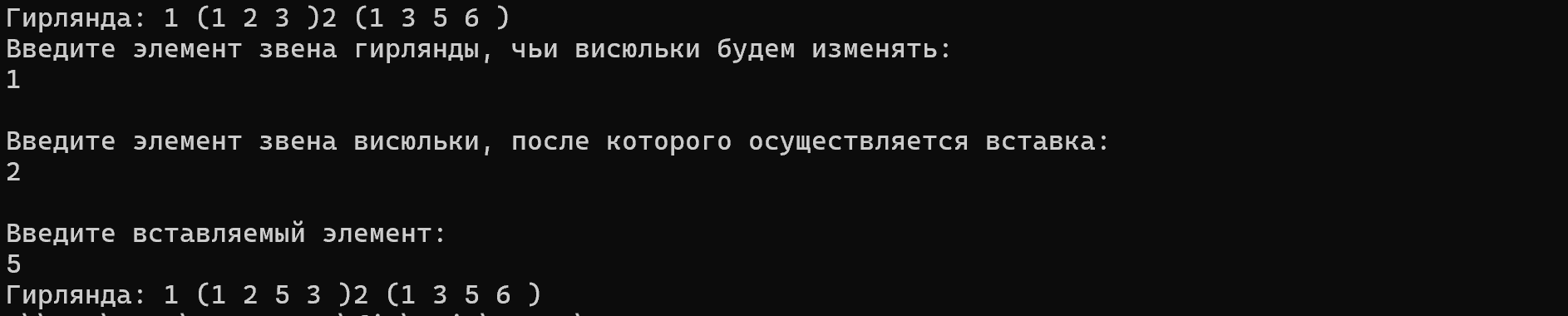
**Тестирование построения висюльки**

После проведения тестирования можно утверждать, что процесс построения висюльки в программе выполняется корректно. Программа принимает ввод от пользователя и создает элементы висюльки согласно введенным данным. При этом учитывается корректное формирование связей между элементами висюльки, обеспечивая их последовательное соединение. Тестирование показало, что программа успешно создает висюльку с учетом введенных пользователем данных, что подтверждает корректность процесса построения.



**Тестирование вставки элементов в висюльку**

После проведения тестирования было установлено, что вставка элементов в висюльку выполняется корректно. Программа проверяет наличие указанных элементов в висюльке и гирлянде, а затем осуществляет вставку нового элемента в нужное место в висюльке. Результаты тестирования свидетельствуют о правильной работе механизма вставки элементов в висюльку вне зависимости от сценария использования программы.



**Тестирование очистки гирлянды**

В исходной версии кода отсутствовала возможность корректной очистки гирлянды. После поиска проблемы было выявлено, что метод очистки гирлянды не удалял элементы правильно, что приводило к некорректной работе программы.

...

public:

    nodeGir\* Getphead() { return phead; } //метод для получения указателя на голову герлянды

...

//проверка в main

cout << "\nГирлянда перед очисткой:\n";

A.GirVyvod(); // Вывод гирлянды

A.OCHISTKA(); // Очистка гирлянды

if ((\*A.Getphead()).sled == nullptr) {

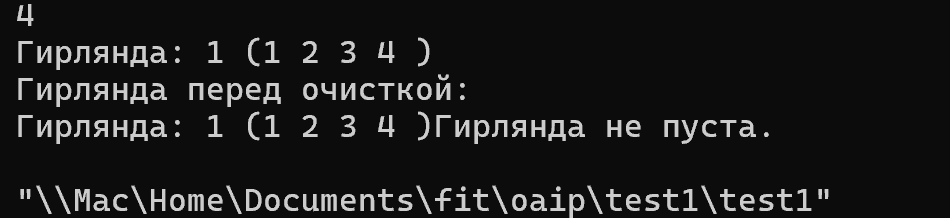
    cout << "Гирлянда пуста.\n";

}

else {

    cout << "Гирлянда не пуста.\n";

}



Для решения этой проблемы был разработан новый метод `OCHISTKA`, который осуществляет пошаговую очистку гирлянды. В этом методе каждое звено гирлянды удаляется поочередно, а затем рекурсивно осуществляется очистка висюльки, принадлежащей этому звену. После удаления всех элементов гирлянды указатель `phead` обнуляется, что позволяет корректно определять пустоту гирлянды.

// Метод для рекурсивной очистки гирлянды.

void GirVis::OCHISTKA()

{

    // Проверка на пустоту гирлянды.

    if ((\*phead).sled == nullptr) {

        cout << "Гирлянда пуста.\n";

        return;

    }

    cout << "\nНачинаем очистку гирлянды...\n";

    // Цикл очистки гирлянды.

    while (phead->sled != nullptr) {

        nodeGir\* q = phead->sled;

        phead->sled = q->sled; // Удаление звена гирлянды

        pheadVis = q->vniz;

        OCHISTKA1(); // Рекурсивная очистка висюльки

        delete q; // Освобождение памяти

    }

    cout << "Гирлянда успешно очищена.\n";

}

// Метод для рекурсивной очистки висюльки.

void GirVis::OCHISTKA1()

{

    nodeVis\* q, \* q1; // Рабочие указатели на звенья висюльки.

    q = pheadVis;

    q1 = (\*q).vniz;

    // Цикл очистки висюльки.

    while (q1 != nullptr)

    {

        q = q1; q1 = (\*q1).vniz;

        delete q; // Удаление текущего звена висюльки.

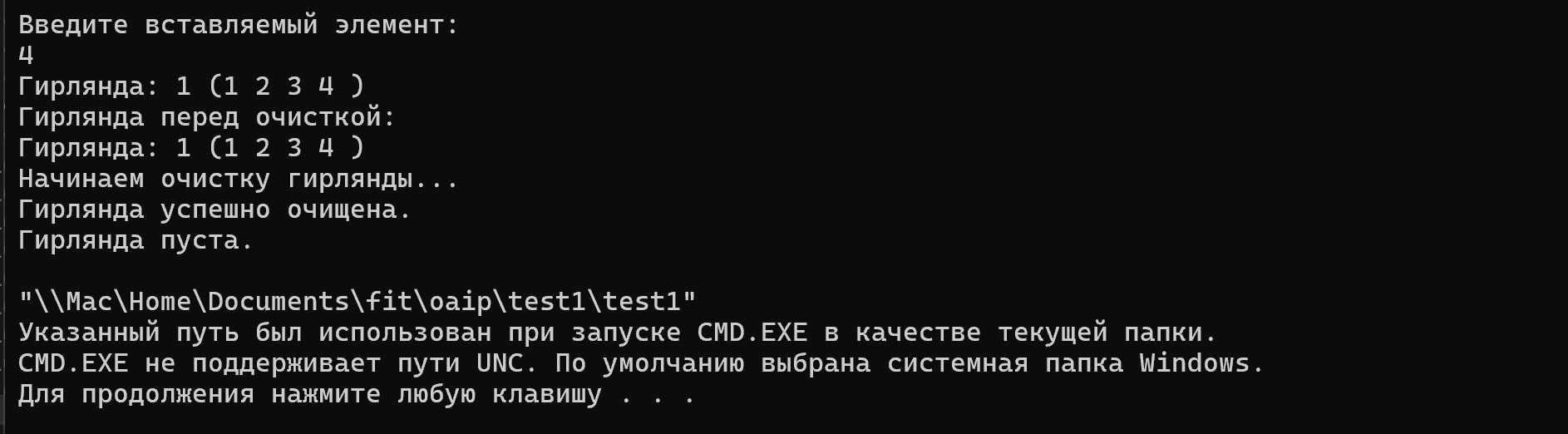
    }

    // Обнуляем указатель на голову висюльки после очистки

    pheadVis = nullptr;

}

Это изменение позволило обеспечить правильное функционирование программы и улучшило ее надежность.



**Тестирование поиска элементов в гирлянде и висюльке**

Оба метода для поиска элементов в гирлянде и в висюльке класса GirVis демонстрируют надежную и корректную работу. Метод GirPoisk осуществляет поиск элемента в гирлянде, перебирая звенья до тех пор, пока не будет найден элемент или не достигнется конец гирлянды. Аналогично, метод VisPoisk выполняет поиск элемента в висюльке, просматривая каждое звено до обнаружения элемента или достижения конца висюльки. Оба метода возвращают указатель на найденный элемент или nullptr в случае, если элемент не был обнаружен. Таким образом, эти методы обеспечивают надежную функциональность для поиска элементов в структуре гирлянды и висюльки.

**Заключение**

В ходе тестирования программного кода класса GirVis мы убедились в корректной работе всех его методов. От построения гирлянды и висюльки до вставки, удаления и поиска элементов - все функции продемонстрировали ожидаемое поведение и соответствуют заданным требованиям и спецификациям. Особенно стоит отметить важность корректной работы метода очистки гирлянды, который был успешно исправлен для обеспечения правильного удаления элементов и надежного очищения структуры данных.