МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра систем автоматизированного проектирования

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Программирование»

Студентка гр. 3353 _____ Карпенко А.Ю.

Преподаватель _____ Калмычков В.А.

Санкт-Петербург

2024

Оглавление

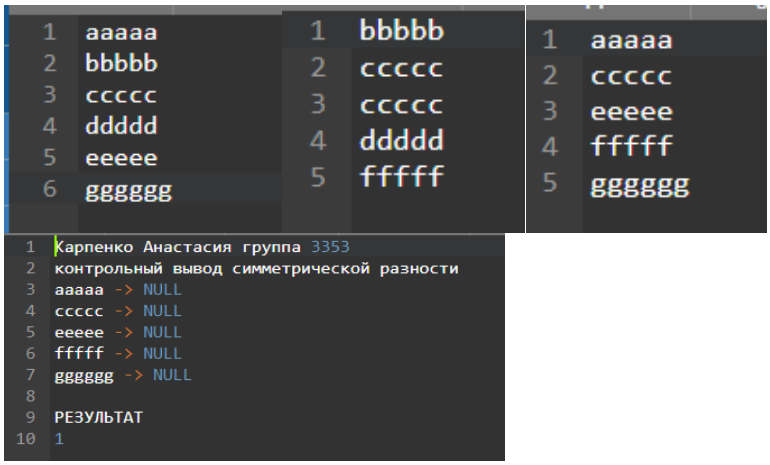
Исходная формулировка задания

Дано 3 файла с символами. Необходимо найти симметрическую разность строк первого и второго файла, далее проверить вложенность списков третьего файла в эту симметрическую разность.

Математическая постановка задания

Дано: 3 файла с набором строк в отсортированном по возрастанию порядке
Необходимо: считать каждый файл, при этом каждую строку разбить на списки заданной длины, получив двумерный массив – текст. Проверить симметрическую разность и вложенность.

Контрольный пример



Разработка интерфейса пользователя

Во входных файлах пользователь заполняет строки словами.
Sssssss…ssss
Sssssss…ssss
…
Sssssss…ssss
Первая строка в выходном файле содержит данные об авторе:
Карпенко Анастасия группа 3353
Со второй строки выводится «контрольный вывод симметрической разности»
Ss sss--> sssss -->sss ss-->s sss -->s s-->NULL
Ss sss--> sssss -->sss ss-->s sss -->s s-->NULL
…
Ss sss--> sssss -->sss ss-->s sss -->s s-->NULL
После выводится поле РЕЗУЛЬТАТ
1-True
0-False

В файле контрольного вывода считанных строк текст принимает вид
Head
|
\|/
Ss sss--> sssss -->sss ss-->s sss -->s s-->NULL
……
|
\|/
Ss sss--> sssss -->sss ss-->s sss -->s s-->NULL
|
\|/
nullptr

Организация ввода-вывода

Для ввода из файла используем библиотеку fstream

(название потока).get(имя переменной);

Для вывода в файл используем библиотеку fstream

(название потока)<<«Текст»<<(имя переменной);

Так же стоит упомянуть, что для работы с файлами потребуется о открывать потоки вывода и открыть файлы, а так же:

fstream (название потока) ;

(название потока).open(«(Имя файла)», ios::(in для ввода, out для ввывода и app для дополнения));

(название потока)..close();

Организация хранения данных

```
// Определение класса для списка узлов
struct StringList {
   StringNode* head{}; // Указатель на начало списка
   StringNode* tail{}; // Указатель на конец списка
   // Конструктор по умолчанию
   StringList();
   // Деструктор для освобождения памяти
   ~StringList();
```

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
fill	Заполнение блока	char part[]	-	void	-
appendNode	Добавление узла в конец списка	const char (*value)	-	void	-
operator<	Определение оператора <	const StringList& lhs, const StringList& rhs	leftNode == nullptr && rightNode != nullptr	bool	
operator==	Оператор сравнения равенства списка узлов	const StringList &lhs, const StringList &rhs	True / false	bool	
ostream& operator<<	Оператор вывода узла в поток	std::ostream& os, const StringList& list	-	-	Вывод узла в поток

```
Определение класса для узла списка
struct StringNode {
   static const int PART_SIZE = 5;
   char data[PART_SIZE]{}; // Строка из 5 символов
   StringNode *next; // Указатель на следующий узел
   // Конструктор для инициализации узла с заданной строкой и указателем на следующий узел
   explicit StringNode(const char (*value));
   // Деструктор для освобождения ресурсов
   ~StringNode();
```

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
operator<	Определение оператора <	const StringNode &lhs, const StringNode &rhs	leftNode == nullptr && rightNode != nullptr	bool	
operator==	Оператор сравнения равенства списка узлов	const StringNode &lhs, const StringNode &rhs	True / false	bool	
ostream& operator<<	Оператор вывода узла в поток	std::ostream &os, const StringNode &node	-	-	Вывод узла в поток

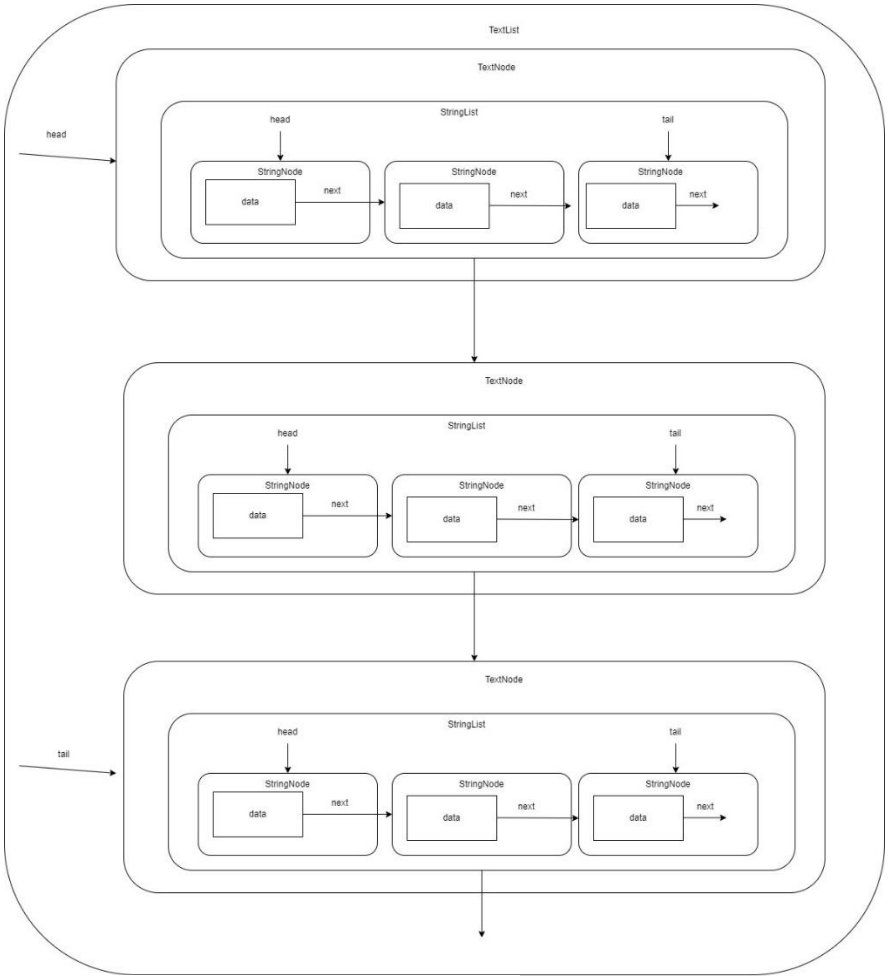
```
// Определение класса для списка узлов
```

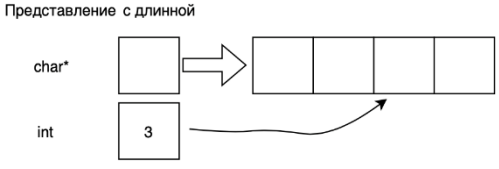
```
struct TextList {
    TextNode* head{}; // Указатель на начало списка
    TextNode* tail{}; // Указатель на конец списка
    // Конструктор по умолчанию
    TextList();

    explicit TextList(const std::string& filename);
    // Деструктор для освобождения памяти
    ~TextList();
```

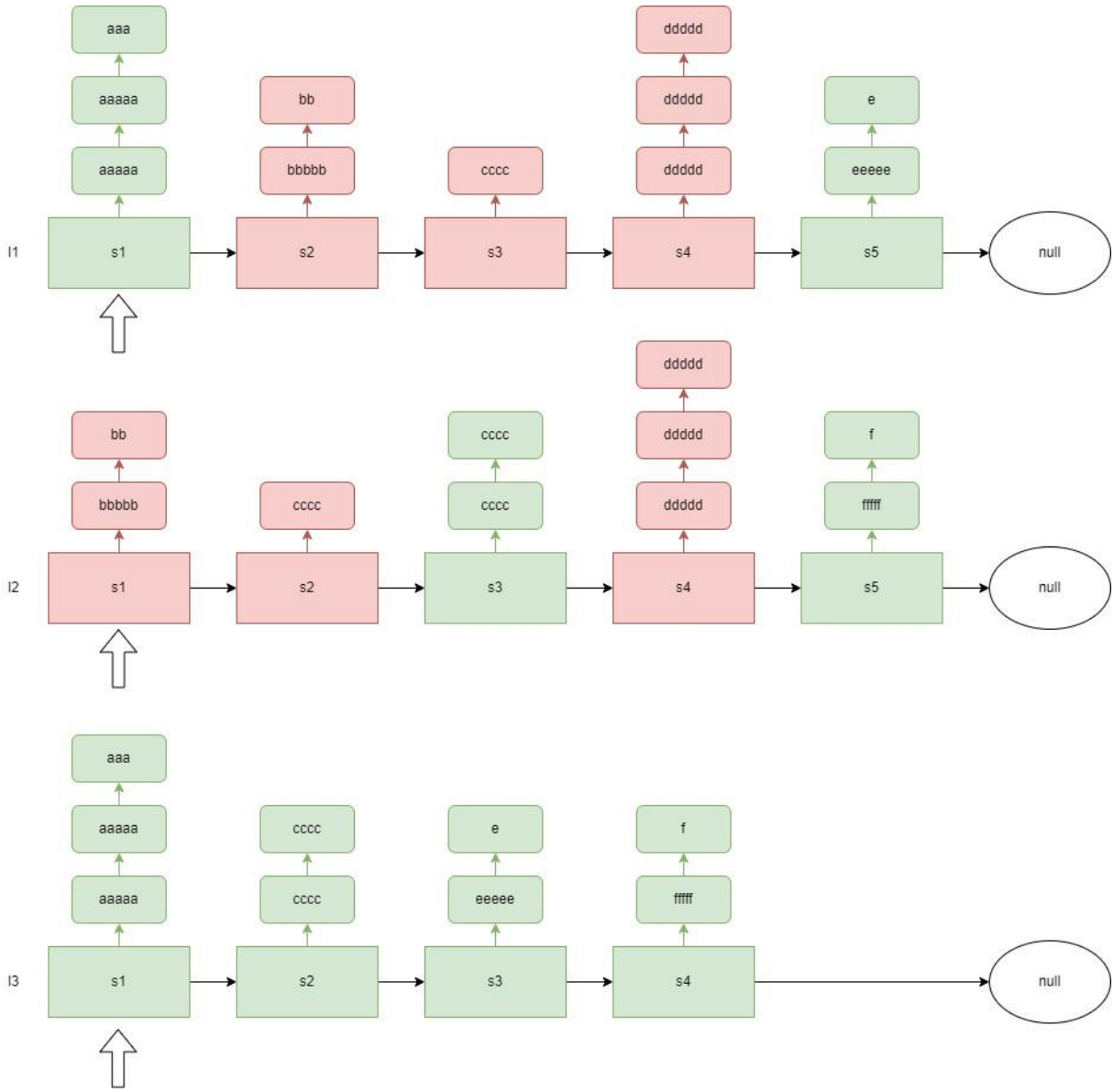
Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
appendNode	Метод добавления узла в конец списка	const StringList& data	-	void	-
ostream& operator<<	Оператор вывода узла в поток	std::ostream& os, const TextList& list	-	-	Вывод узла в поток

```
// Определение класса для узла списка
struct TextNode {
    StringList data; // Указатель на StringList
    TextNode* next; // Указатель на следующий узел
    // Конструктор с параметрами
    explicit TextNode(const StringList& list);
    // Деструктор для освобождения ресурсов
    ~TextNode();
```

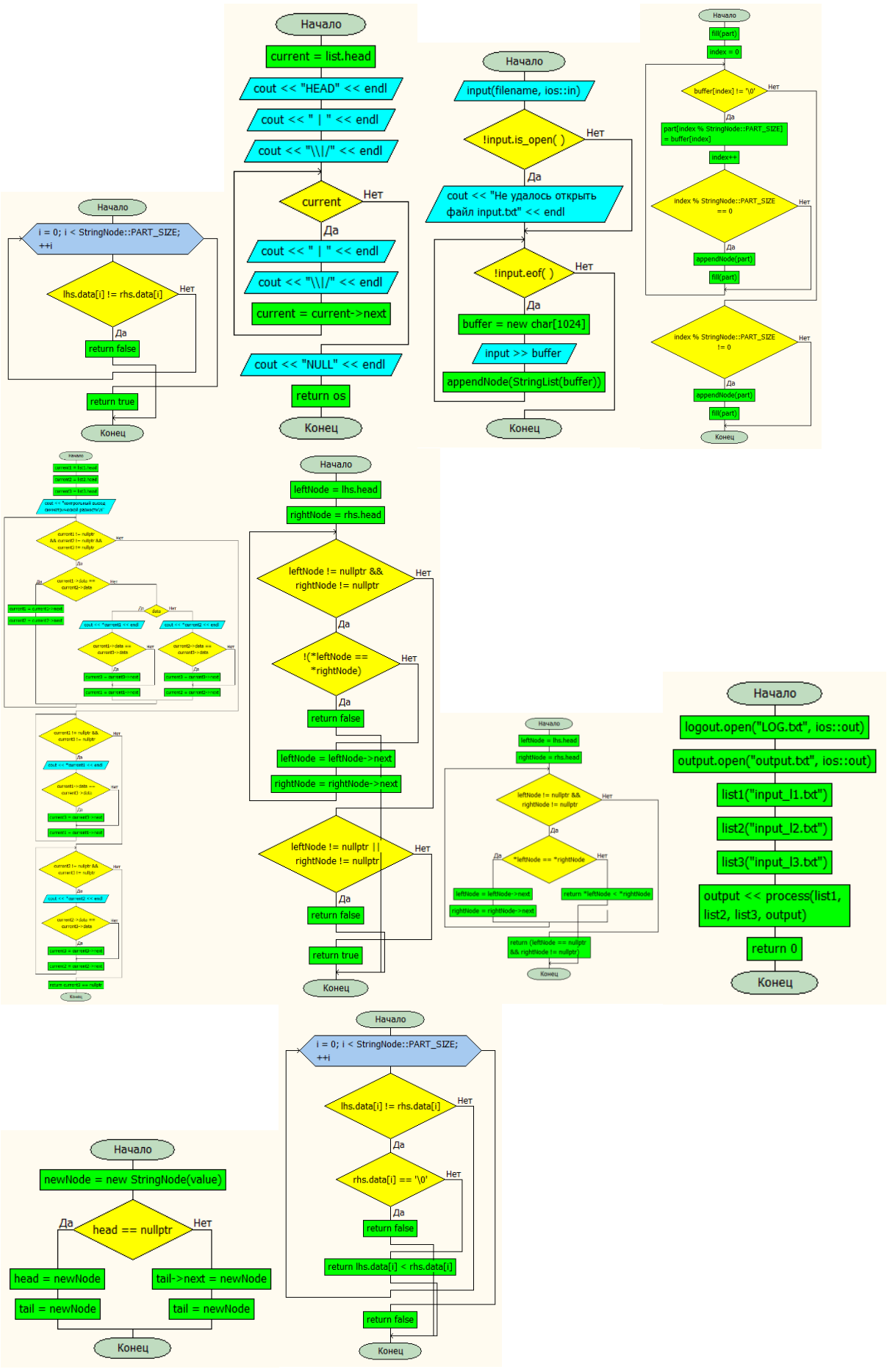
Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
ostream& operator<<	Оператор вывода узла в поток	std::ostream& os, const TextNode& node	-	-	Вывод узла в поток






aaa
aaaaa
aaaaa
bb
bbbbb
cccc
ddddd
ddddd
ddddd
e
eeeee
l1
s1
s2
s3
s4
s5
null
ddddd
bb
bbbbb
cccc
cccc
ddddd
ddddd
f
fffff
l2
s1
s2
s3
s4
s5
null
aaa
aaaaa
aaaaa
cccc
cccc
e
eeeee
f
fffff
l3
s1
s2
s3
s4
null

Описание алгоритма



Текст программы

Main.cpp

```
#include <iostream>
#include <fstream>

#include "TextList.h"

using namespace std;

bool process(const TextList& list1, const TextList& list2, const TextList& list3, fstream& output){
    TextNode* current1 = list1.head;
    TextNode* current2 = list2.head;
    TextNode* current3 = list3.head;
    output<<"контрольный вывод симметрической разности\n";
    while (current1 != nullptr && current2 != nullptr && current3 != nullptr) {
        if(current1->data == current2->data){
            current1= current1->next;
            current2= current2->next;
        }else if (current1->data < current2->data){
            output<<*current1<<endl;
            if(current1->data == current3->data){current3 = current3->next;}
            current1=current1->next;
        }else {
            output<<*current2<<endl;
            if(current2->data == current3->data){current3 = current3->next;}
            current2=current2->next;
        }
    }
    while(current1 != nullptr && current3!= nullptr){
        output<<*current1<<endl;
        if(current1->data == current3->data){current3 = current3->next;}
        current1=current1->next;
    }
    while(current2 != nullptr && current3!= nullptr){
        output<<*current2<<endl;
        if(current2->data == current3->data){current3 = current3->next;}
        current2=current2->next;
    }
    output<<"\nРЕЗУЛЬТАТ\n";
    return current3 == nullptr;
}
int main() {
    fstream logout;
    fstream output;
    logout.open("LOG.txt", ios::out);
    output.open("output.txt", ios::out);
    output<<"Карпенко Анастасия группа 3353\n";

    TextList list1("input_l1.txt");
    logout<<list1<<endl;

    TextList list2("input_l2.txt");
    logout<<list2<<endl;

    TextList list3("input_l3.txt");
    logout<<list3<<endl;

    output<<process(list1, list2, list3, output);
    return 0;
}
```

StringList.h

```
#ifndef STRING_LIST_H
#define STRING_LIST_H

#include "StringNode.h"

using namespace std;

// Определение класса для списка узлов
struct StringList {
    StringNode* head{}; // Указатель на начало списка
    StringNode* tail{}; // Указатель на конец списка
    // Конструктор по умолчанию
    StringList();
    void fill(char part[]);
    StringList(const StringList& other);

    explicit StringList(const char (*buffer));

    // Деструктор для освобождения памяти
    ~StringList();

    // Добавление узла в конец списка
    void appendNode(const char (*value));

    // Оператор для сравнения StringNode
    friend bool operator<(const StringList& lhs, const StringList& rhs);

    // Оператор сравнения равенства StringNode
```

	friend bool operator==(const StringList& lhs, const StringList& rhs); // Вывод StringList в поток friend std::ostream& operator<<(std::ostream& os, const StringList& node); }; #endif // STRING_LIST_H
StringList.cpp	#include "StringList.h" #include "StringNode.h" // Конструктор по умолчанию StringList::StringList() : head(nullptr), tail(nullptr) { } void StringList::fill(char part[]) { for (int i = 0; i < StringNode::PART_SIZE; ++i) { part[i] = '\0'; } } StringList::StringList(const char (*buffer)) { char part[StringNode::PART_SIZE]; fill(part); // Используем функцию fill() для заполнения нулями int index = 0; while (buffer[index] != '\0') { part[index % StringNode::PART_SIZE] = buffer[index]; index++; if (index % StringNode::PART_SIZE == 0) { appendNode(part); fill(part); // Используем функцию fill() для сброса массива part в нули } } if (index % StringNode::PART_SIZE != 0) { appendNode(part); fill(part); // Используем функцию fill() для сброса массива part в нули } } // Конструктор копирования StringList::StringList(const StringList& other) : head(nullptr), tail(nullptr) { // Если список other пуст, просто выходим из конструктора if (other.head == nullptr) return; // Копируем данные из списка other StringNode* current = other.head; while (current) { appendNode(current->data); current = current->next; } } // Деструктор для освобождения памяти StringList::~StringList() { StringNode* current = head; // Указатель на текущий узел while (current != nullptr) { StringNode* next = current->next; // Сохраняем указатель на следующий узел delete current; // Освобождаем память для текущего узла current = next; // Переходим к следующему узлу } head = nullptr; // Устанавливаем указатель на начало списка в nullptr tail = nullptr; // Устанавливаем указатель на конец списка в nullptr } // Добавление узла в конец списка void StringList::appendNode(const char (*value)) { // Создаем новый узел с заданным значением auto* newNode = new StringNode(value); // Если список пуст, устанавливаем новый узел как начало и конец списка if (head == nullptr) { head = newNode; tail = newNode; } else { // Добавляем новый узел в конец списка и обновляем указатель на хвост tail->next = newNode; tail = newNode; } } // Определение оператора < bool operator<(const StringList& lhs, const StringList& rhs) { StringNode* leftNode = lhs.head; // Указатель на текущий узел списка lhs StringNode* rightNode = rhs.head; // Указатель на текущий узел списка rhs // Перебираем узлы обоих списков и сравниваем содержимое их узлов while (leftNode != nullptr && rightNode != nullptr) { // Если содержимое узлов отличается, списки не равны

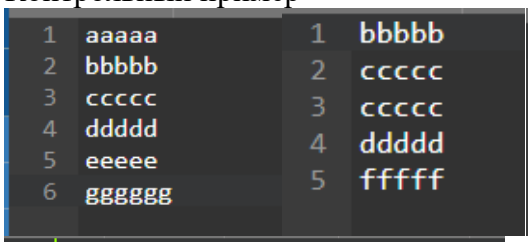
	`` if (*leftNode == *rightNode) { leftNode = leftNode->next; // Переходим к следующему узлу списка lhs rightNode = rightNode->next; // Переходим к следующему узлу списка rhs } else return *leftNode < *rightNode; } return (leftNode == nullptr && rightNode != nullptr); } // Оператор сравнения равенства списка узлов bool operator==(const StringList &lhs, const StringList &rhs) { StringNode *leftNode = lhs.head; // Указатель на текущий узел списка lhs StringNode *rightNode = rhs.head; // Указатель на текущий узел списка rhs // Перебираем узлы обоих списков и сравниваем содержимое их узлов while (leftNode != nullptr && rightNode != nullptr) { // Если содержимое узлов отличается, списки не равны if (!(*leftNode == *rightNode)) return false; leftNode = leftNode->next; // Переходим к следующему узлу списка lhs rightNode = rightNode->next; // Переходим к следующему узлу списка rhs } // Если один список закончился раньше другого, они не равны if (leftNode != nullptr || rightNode != nullptr) return false; // Все узлы совпадают, списки равны return true; } // Оператор вывода узла в поток std::ostream& operator<<(std::ostream& os, const StringList& list) { StringNode* current = list.head; while (current != nullptr) { os << current->data << " -> "; current = current->next; } os << "NULL"; return os; } ```
StringNode.h	``` #ifndef STRING_NODE_H #define STRING_NODE_H #include <iostream> // Для использования std::ostream // Определение класса для узла списка struct StringNode { static const int PART_SIZE = 5; char data[PART_SIZE]{}; // Строка из 5 символов StringNode *next; // Указатель на следующий узел // Конструктор для инициализации узла с заданной строкой и указателем на следующий узел explicit StringNode(const char (*value)); // Деструктор для освобождения ресурсов ~StringNode(); // Оператор для сравнения StringNode friend bool operator<(const StringNode &lhs, const StringNode &rhs); // Оператор сравнения равенства StringNode friend bool operator==(const StringNode &lhs, const StringNode &rhs); // Вывод узла в поток friend std::ostream &operator<<(std::ostream &os, const StringNode &node); }; #endif // STRING_NODE_H ```
StringNode.cpp	``` #include "StringNode.h" using namespace std; // Конструктор StringNode::StringNode(const char (*value)) : next(nullptr) { // Копируем строку value в поле data, обрезая ее до 5 символов for (int i = 0; i < PART_SIZE; ++i) data[i] = value[i]; } // Деструктор StringNode::~StringNode() = default; // Оператор сравнения равенства StringNode bool operator==(const StringNode &lhs, const StringNode &rhs) { // Сравниваем содержимое строк узлов for (int i = 0; i < StringNode::PART_SIZE; ++i) if (lhs.data[i] != rhs.data[i]) return false; return true; } ```

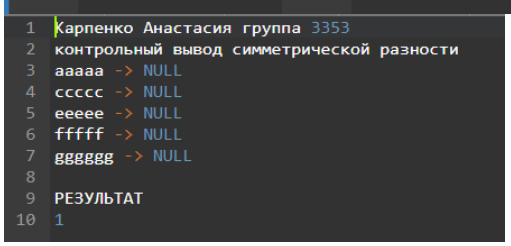
	// Определение оператора < bool operator<(const StringNode &lhs, const StringNode &rhs) { for (int i = 0; i < StringNode::PART_SIZE; ++i) { if (lhs.data[i] != rhs.data[i]) { if (rhs.data[i] == '\0') return false; return lhs.data[i] < rhs.data[i]; } } return false; } // Оператор вывода узла в поток std::ostream &operator<<(std::ostream &os, const StringNode &node) { os << node.data; return os; }
TextList.h	#ifndef TEXT_LIST_H #define TEXT_LIST_H #include "TextNode.h" #include "TextList.h" using namespace std; // Определение класса для списка узлов struct TextList { TextNode* head{}; // Указатель на начало списка TextNode* tail{}; // Указатель на конец списка // Конструктор по умолчанию TextList(); explicit TextList(const std::string& filename); // Деструктор для освобождения памяти ~TextList(); // Добавление узла в конец списка void appendNode(const StringList& data); // Вывод StringList в поток friend std::ostream& operator<<(std::ostream& os, const TextList& node); }; #endif // TEXT_LIST_H
TextList.cpp	#include <fstream> #include "TextList.h" #include "TextNode.h" // Конструктор по умолчанию TextList::TextList() : head(nullptr), tail(nullptr) { } // Деструктор для освобождения памяти TextList::~TextList() { TextNode* current = head; // Указатель на текущий узел while (current != nullptr) { TextNode* next = current->next; // Сохраняем указатель на следующий узел delete current; // Освобождаем память для текущего узла current = next; // Переходим к следующему узлу } head = nullptr; // Устанавливаем указатель на начало списка в nullptr tail = nullptr; // Устанавливаем указатель на конец списка в nullptr } TextList::TextList(const std::string& filename) { fstream input(filename, ios::in); if (!input.is_open()) { cout << "Не удалось открыть файл input.txt" << endl; throw std::runtime_error("Ошибка открытия файла"); } while (!input.eof()) { char *buffer = new char[1024]; // Буфер для хранения считанной строки input >> buffer; appendNode(StringList(buffer)); } } // Метод добавления узла в конец списка void TextList::appendNode(const StringList& data) { auto* newNode = new TextNode(data); // Создание нового узла if (head == nullptr) { // Если список пуст head = newNode; tail = newNode; } else {

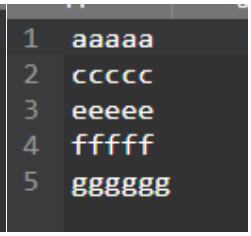
	tail->next = newNode; tail = newNode; } } // Оператор вывода узла в поток std::ostream& operator<<(std::ostream& os, const TextList& list) { TextNode* current = list.head; os << "HEAD" << endl; os << " | " << endl; os << "\\\|/" << endl; while (current) { os << current->data << endl; os << " | " << endl; os << "\\\|/" << endl; current = current->next; } os << "NULL" << endl; return os; }
TextNode.h	#ifndef TEXT_NODE_H #define TEXT_NODE_H #include "StringList.h" #include <iostream> // Для использования std::ostream // Определение класса для узла списка struct TextNode { StringList data; // Указатель на StringList TextNode* next; // Указатель на следующий узел // Конструктор с параметрами explicit TextNode(const StringList& list); // Деструктор для освобождения ресурсов ~TextNode(); // Вывод узла в поток friend std::ostream& operator<<(std::ostream& os, const TextNode& node); }; #endif // TEXT_NODE_H
TextNode.cpp	#include "TextNode.h" using namespace std; // Конструктор с параметрами TextNode::TextNode(const StringList& list) : data(list), next(nullptr) {} // Деструктор для освобождения ресурсов TextNode::~TextNode() = default; // Оператор вывода узла в поток std::ostream& operator<<(std::ostream& os, const TextNode& node) { os << node.data; return os; }

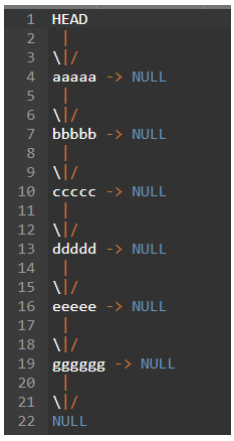
Результаты работы программы

Контрольный пример

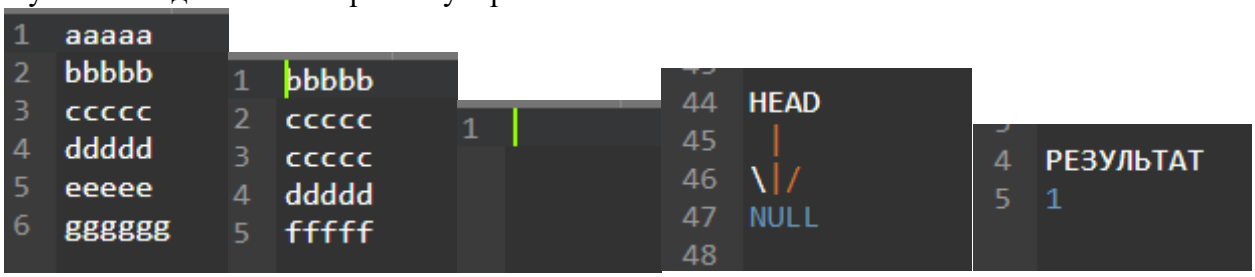




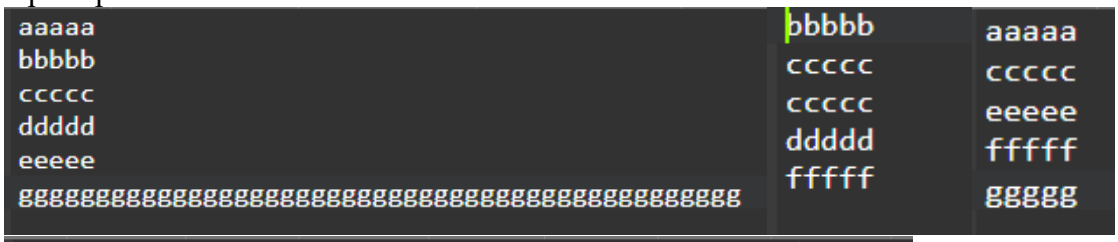


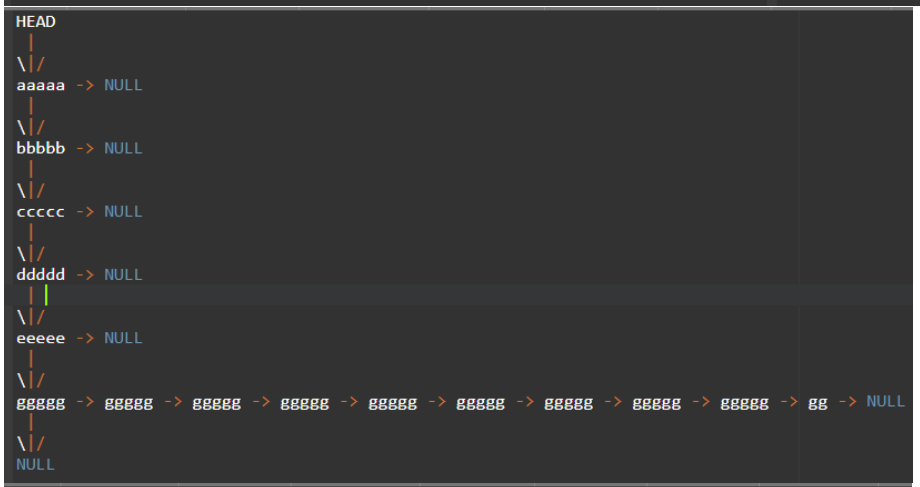


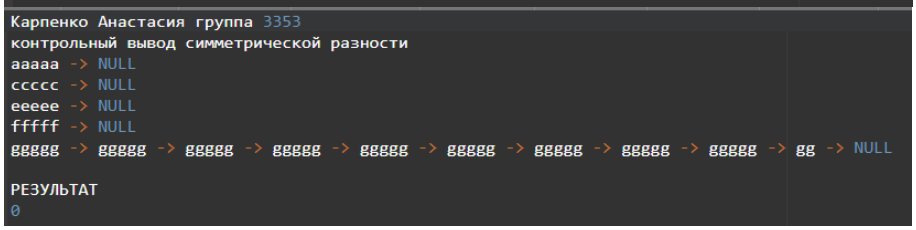
Пустота входит в симметрическую разность



Пример с большим количеством символов







Вывод

В ходе работы было освоено разбиение на несколько файлов: .cpp и .h. Была проведена работа с функциями, освоена работа со списками списков.