**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчЁт

**по лабораторной работе № 4**

**по дисциплине «Программирование»**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент гр. 4354 |  |  | И.В. Чучалин |
| Преподаватель |  |  | В.А. Калмычков |

# Оглавление

[Оглавление 2](#_Toc197954174)

[Задание 2](#_Toc197954175)

[Контрольные примеры 2](#_Toc197954176)

[Организация интерфейса пользователя 2](#_Toc197954177)

[Организация входного файла 2](#_Toc197954178)

[Организация выходного файла 3](#_Toc197954179)

[Организация консоли 3](#_Toc197954180)

[Реализация ввода/вывода 3](#_Toc197954181)

[Модульное представление 3](#_Toc197954182)

[Внутреннее представление данных 3](#_Toc197954183)

[Описание внутренних функций 4](#_Toc197954184)

[Определение функций 4](#_Toc197954185)

[Назначение функций 5](#_Toc197954186)

[Описание алгоритма 5](#_Toc197954187)

[Общее описание алгоритма программы 5](#_Toc197954188)

[Принцип работы иерархического списка 5](#_Toc197954189)

[Блок-схема 6](#_Toc197954190)

[Текст программы 9](#_Toc197954191)

[Результат работы программы 13](#_Toc197954192)

[Вывод 13](#_Toc197954193)

# Задание

Задан текст, представленный иерархическим списком. Необходимо проверить истинность утверждения: S3 с (S1 п S2).

# Контрольные примеры

1. Утверждение истинно.

Входные данные: Полученный результат:

okaywow S1 п S2: okaywow

dont read Утверждение S3 с (S1 п S2) истинно.

this text

okaywow

you did it

okay wow

okaywow

2. Утверждение ложно.

Входные данные: Полученный результат:

okay S1 п S2: okay

dont read Утверждение S3 с (S1 п S2) ложно.

this text

okay

you did it

okaywow

ok

# Организация интерфейса пользователя

## Организация входного файла

О1. Текст, представленный в виде трех групп строк, разделенных пустыми строками.

## Организация выходного файла

1. Вывод считанных списков.
2. Вывод пересечения.
3. Вывод результата.

## Организация консоли

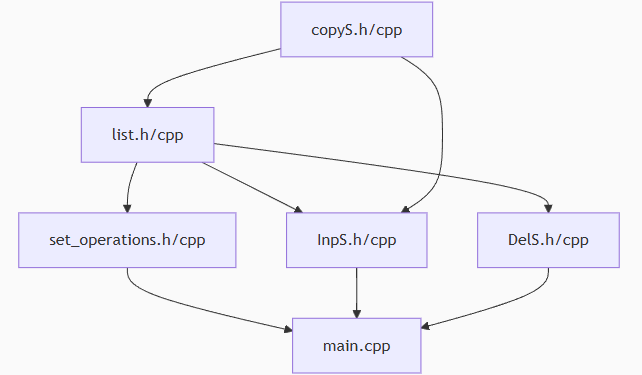
О1. Вывод ошибок.

# Реализация ввода/вывода

Использованы потоки ввода (cin>>) и вывода (cout<<) библиотеки «iostream», а также функции объекта cin: getline, clear, eof.

Использованы потоки чтения из файла (>>) и записи в файл (<<) библиотеки «fstream», а также функции объекта типа fstream: open, is\_open, fail, seekg, eof, close. Для настройки ввода были использованы функции setiosflags, resetiosflags библиотеки «iomanip».

# Модульное представление



# Внутреннее представление данных

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Переменная | Назначение |
| EL\_Stroka | struct | char A[N] | Хранит строку данных |
| EL\_Stroka\* next | Связывает элементы в односвязный список |
| Form\_Stroka | EL\_Stroka\* h, \* l | Указывает на первый и последний элемент списка строк |
| unsigned L | Счётчик элементов в списке |
| EL\_V | Form\_Stroka F | Хранит список строк (горизонтальный уровень) |
| EL\_V\* next | Связывает элементы в вертикальный список (иерархия) |
| Form\_V | EL\_V\* h, \* cur, \* last | Указывает на первый, текущий и последний уровень списка. |

|  |  |  |
| --- | --- | --- |
| Имя переменной | Тип | Назначение |
| N | const unsigned | Максимальная длина строки |
| fin | ifstream | Поток для чтения из файла |
| sets | |  |  | | --- | --- | | Form\_V\*[3] |  | | Массив указателей на множества |
| currentSet | int | Индекс текущего множества |
| temp | char[N] | Буфер для временного хранения строки из файла. |
| curV | EL\_V\* | Текущий уровень списка. |
| pos | int | Позиция в буфере |
| ch | char | Очередной символ из файла |
| level1, level2 | EL\_V\* | Указатели на текущие уровни списков |
| elem1, elem2 | EL\_Stroka\* | Указатели на текущие строки в уровнях |
| current1, current2 | const char\* | Текущие строки для сравнения. |
| resLevel | EL\_V\* | Указатель на текущий уровень результирующего списка |
| newLevel | EL\_V\* | Временный указатель для создания новых уровней |
| cmp | int | Результат сравнения строк |
| subLevel, mainLevel | EL\_V\* | Указатели на уровни списков |
| subElem, mainElem | |  |  | | --- | --- | | EL\_Stroka\* |  | | Указатели на строки в уровнях. |
| currentSub, currentMain | const char\* | Текущие строки для сравнения |
| curV | EL\_V\* | Текущий уровень списка |
| curS | EL\_Stroka\* | Текущая строка в уровне |
| tempV, tempS | EL\_V\*, EL\_Stroka\* | Временные указатели для освобождения памяти |
| S1, S2, S3 | Form\_V | Множества для хранения данных из файла |
| intersection | Form\_V | Результат пересечения |
| fout | ofstream | Поток для записи результатов в файл |
| isSubset | bool | Флаг результата проверки |

# Описание внутренних функций

## Определение функций

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Имя функции | Аргументы | | | | Возвращаемое значение |
| Входные | Выходные | Модифи-цируемые | Транзитные |
| CopyString | const char\* src, unsigned max | char\* dest | dest |  | void |
| CompareStrings | const char\* s1, const char\* s2 |  |  |  | int |
| InitELStroka | const char\* value | EL\_Stroka\*& el | el |  | void |
| InitFormStroka |  | Form\_Stroka& fs | fs.h, fs.l, fs.L |  |
| InitELV |  | EL\_V\*& el | el |  |
| InitFormV |  | Form\_V& fv | fv.h, fv.cur, fv.last |  |
| InsertString | const char\* value | Form\_Stroka& fs | fs.h, fs.l, fs.L |  |
| PrintHierarchy | |  |  | | --- | --- | | const Form\_V& list, const char\* name |  | | std::ostream& out |  |  |
| InpThreeSets | const char\* filename | Form\_V& S1, Form\_V& S2, Form\_V& S3 | S1, S2, S3 | std::ifstream fin, char temp[N] | void |
| |  |  | | --- | --- | | Intersection |  | | const Form\_V& s1, const Form\_V& s2 | Form\_V& result | result | EL\_V\* level1, EL\_V\* level2, EL\_Stroka\* elem1, EL\_Stroka\* elem2 | void |
| IsSubset | const Form\_V& subset, const Form\_V& mainSet |  |  | EL\_V\* subLevel, EL\_V\* mainLevel, EL\_Stroka\* subElem, EL\_Stroka\* mainElem | bool |
| Delete | Form\_V& fv |  | fv.h, fv.cur, fv.last | EL\_V\* curV, EL\_Stroka\* curS, EL\_V\* tempV, EL\_Stroka\* tempS | void |
| main |  |  | S1, S2, S3, intersection | std::ofstream fout | int |

## Назначение функций

* CopyString: Копирует строку src в dest .
* CompareStrings: Сравнивает строки лексикографически.
* InsertString: Добавляет строку value в конец списка fs.
* PrintHierarchy: Рекурсивно выводит список list в поток out.
* Init-функции инициируют.
* InpThreeSets читает данные из файла filename и заполняет множества S1, S2, S3.
* Intersection: Записывает в result пересечение множеств s1 и s2.
* IsSubset: Возвращает true, если subset является подмножеством mainSet.
* Delete рекурсивно удаляет все элементы списка fv и освобождает память.

# Описание алгоритма

## Общее описание алгоритма программы

Программа предназначена для проверки утверждения, является ли множество S3 подмножеством пересечения множеств S1 и S2. Работа программы начинается с чтения трех множеств строк из входного файла, где каждое множество отделено пустой строкой. Данные хранятся в виде иерархического списка, состоящего из двух уровней: вертикального (EL\_V) для групп строк и горизонтального (EL\_Stroka) для самих строк. После загрузки данных программа вычисляет пересечение S1 и S2, последовательно сравнивая элементы обоих множеств в отсортированном порядке. Результат пересечения сохраняется в отдельном списке. Затем осуществляется проверка, все ли элементы S3 присутствуют в полученном пересечении. Если это условие выполняется, программа возвращает истину, в противном случае — ложь. Весь процесс сопровождается выводом промежуточных результатов (исходных множеств, пересечения и итога проверки) в выходной файл. Для работы с памятью используются динамические структуры данных, которые создаются и удаляются вручную, что исключает утечки ресурсов.

## Принцип работы иерархического списка

Иерархический список организован как двухуровневая структура. На верхнем уровне находятся элементы типа EL\_V, которые образуют вертикальный список и содержат ссылки на нижний уровень — односвязные списки строк (Form\_Stroka). Каждый элемент Form\_Stroka (EL\_Stroka) хранит строку фиксированной длины (определенной константой N) и указатель на следующий элемент. Вертикальные уровни (EL\_V) связаны между собой через поле next, что позволяет создавать сложные иерархии данных. Для управления списками используются служебные структуры Form\_Stroka и Form\_V, которые содержат указатели на начало (h), конец (l или last) и текущий элемент (cur), а также счетчик элементов (L). Такая организация позволяет эффективно добавлять новые элементы в конец списка и обходить структуры как по горизонтали (перебор строк в уровне), так и по вертикали (переход между уровнями). Все операции вставки, удаления и сравнения оптимизированы для работы с предварительно отсортированными данными, что обеспечивает высокую производительность.

## Блок-схема

1. copyS.cpp 2. DelS.cpp

CopyString CompareStrings Delete

Изображение выглядит как текст, диаграмма, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, снимок экрана, диаграмма, линия

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

3. List.cpp

InsertString PrintHierarchy

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, снимок экрана, дизайн, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

4. InpS.cpp 5. Main.cpp

InpThreeSets

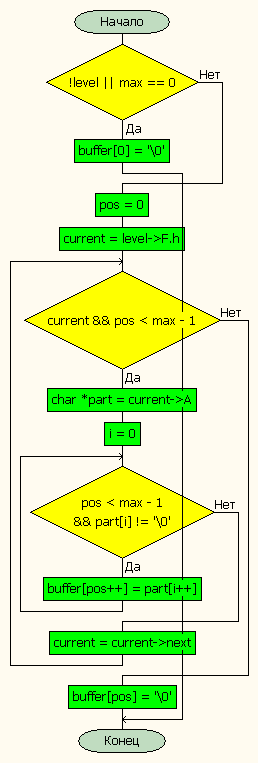
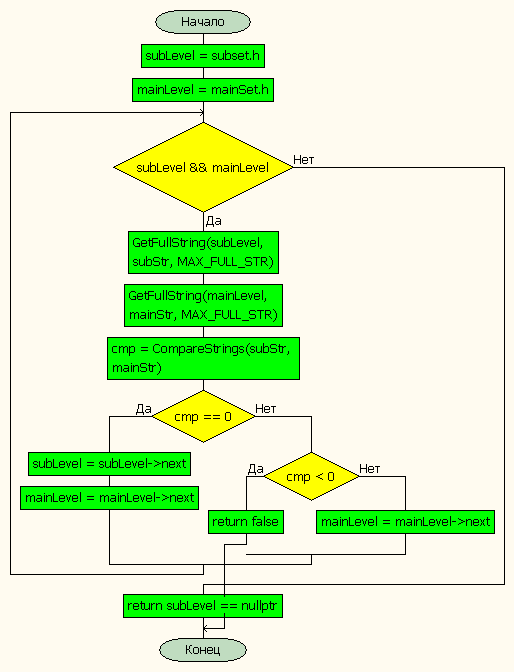
Изображение выглядит как диаграмма, Красочность, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

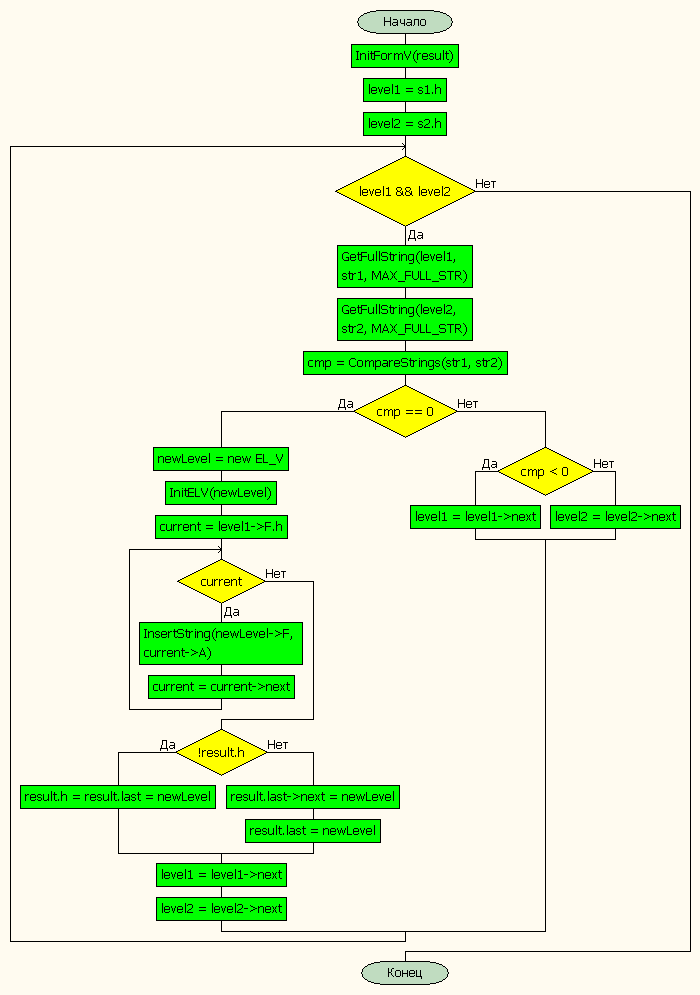
Контент, сгенерированный ИИ, может содержать ошибки.

5. Set\_operations.cpp

GetFullString IsSubset

Intersection



# Текст программы

**copyS.h**

**#pragma once**

**#ifndef COPYS\_H**

**#define COPYS\_H**

**void CopyString(char\* dest, const char\* src, unsigned max);**

**int CompareStrings(const char\* s1, const char\* s2);**

**#endif**

**DelS.h**

**#pragma once**

**#ifndef DELS\_H**

**#define DELS\_H**

**#include "list.h"**

**void Delete(Form\_V& fv);**

**#endif**

**InpS.h**

**#ifndef INPS\_H**

**#define INPS\_H**

**#include "list.h"**

**// Чтение трех множеств из файла**

**void InpThreeSets(Form\_V& S1, Form\_V& S2, Form\_V& S3, const char\* filename);**

**#endif**

**List.h**

**#pragma once**

**#ifndef LIST\_H**

**#define LIST\_H**

**#include <fstream>**

**const unsigned N = 6;**

**struct EL\_Stroka {**

**char A[N];**

**EL\_Stroka\* next;**

**};**

**struct Form\_Stroka {**

**EL\_Stroka\* h, \* l;**

**unsigned L;**

**};**

**struct EL\_V {**

**Form\_Stroka F;**

**EL\_V\* next;**

**};**

**struct Form\_V {**

**EL\_V\* h, \* cur, \* last;**

**};**

**void InitELStroka(EL\_Stroka\*& el, const char\* value);**

**void InitFormStroka(Form\_Stroka& fs);**

**void InitELV(EL\_V\*& el);**

**void InitFormV(Form\_V& fv);**

**void InsertString(Form\_Stroka& fs, const char\* value);**

**void PrintHierarchy(const Form\_V& list, std::ostream& out, const char\* name);**

**#endif**

**Set\_operations.h**

**#pragma once**

**#ifndef LIST\_H**

**#define LIST\_H**

**#include <fstream>**

**const unsigned N = 6;**

**struct EL\_Stroka {**

**char A[N];**

**EL\_Stroka\* next;**

**};**

**struct Form\_Stroka {**

**EL\_Stroka\* h, \* l;**

**unsigned L;**

**};**

**struct EL\_V {**

**Form\_Stroka F;**

**EL\_V\* next;**

**};**

**struct Form\_V {**

**EL\_V\* h, \* cur, \* last;**

**};**

**void InitELStroka(EL\_Stroka\*& el, const char\* value);**

**void InitFormStroka(Form\_Stroka& fs);**

**void InitELV(EL\_V\*& el);**

**void InitFormV(Form\_V& fv);**

**void InsertString(Form\_Stroka& fs, const char\* value);**

**void PrintHierarchy(const Form\_V& list, std::ostream& out, const char\* name);**

**#endif**

**copyS.cpp**

**#include "copyS.h"**

**void CopyString(char\* dest, const char\* src, unsigned max) {**

**unsigned i = 0;**

**while (i < max - 1 && src[i] != '\0') {**

**dest[i] = src[i];**

**i++;**

**}**

**dest[i] = '\0';**

**}**

**int CompareStrings(const char\* s1, const char\* s2) {**

**int i = 0;**

**while (s1[i] != '\0' && s2[i] != '\0') {**

**if (s1[i] < s2[i]) return -1;**

**if (s1[i] > s2[i]) return 1;**

**i++;**

**}**

**if (s1[i] == '\0' && s2[i] != '\0') return -1;**

**if (s1[i] != '\0' && s2[i] == '\0') return 1;**

**return 0;**

**}**

**DelS.cpp**

**#include "DelS.h"**

**#include "list.h"**

**void Delete(Form\_V& fv) {**

**EL\_V\* curV = fv.h;**

**while (curV) {**

**EL\_Stroka\* curS = curV->F.h;**

**while (curS) {**

**EL\_Stroka\* tempS = curS;**

**curS = curS->next;**

**delete tempS;**

**}**

**EL\_V\* tempV = curV;**

**curV = curV->next;**

**delete tempV;**

**}**

**InitFormV(fv);**

**}**

**InpS.cpp**

**#include "InpS.h"**

**#include "list.h"**

**#include "copyS.h"**

**#include <fstream>**

**void InpThreeSets(Form\_V& S1, Form\_V& S2, Form\_V& S3, const char\* filename) {**

**InitFormV(S1);**

**InitFormV(S2);**

**InitFormV(S3);**

**std::ifstream fin(filename);**

**if (!fin.is\_open()) return;**

**const int MAX\_SETS = 3;**

**Form\_V\* sets[MAX\_SETS] = { &S1, &S2, &S3 };**

**int currentSet = 0;**

**char temp[N];**

**EL\_V\* curV = nullptr;**

**int pos = 0;**

**char ch;**

**while (fin.get(ch)) {**

**if (ch == '\n') {**

**if (pos > 0) {**

**temp[pos] = '\0';**

**if (curV) InsertString(curV->F, temp);**

**pos = 0;**

**}**

**if (fin.peek() == '\n') {**

**fin.get();**

**currentSet++;**

**if (currentSet >= MAX\_SETS) currentSet = MAX\_SETS - 1;**

**curV = nullptr;**

**}**

**else {**

**curV = nullptr;**

**}**

**continue;**

**}**

**if (!curV) {**

**curV = new EL\_V;**

**InitELV(curV);**

**if (!sets[currentSet]->h) {**

**sets[currentSet]->h = sets[currentSet]->last = curV;**

**}**

**else {**

**sets[currentSet]->last->next = curV;**

**sets[currentSet]->last = curV;**

**}**

**}**

**temp[pos++] = ch;**

**if (pos >= N - 1) {**

**temp[pos] = '\0';**

**InsertString(curV->F, temp);**

**pos = 0;**

**}**

**}**

**if (pos > 0 && curV) {**

**temp[pos] = '\0';**

**InsertString(curV->F, temp);**

**}**

**fin.close();**

**}**

**List.cpp**

**#include "list.h"**

**#include "copyS.h"**

**#include <iostream>**

**void InitELStroka(EL\_Stroka\*& el, const char\* value) {**

**el = new EL\_Stroka;**

**CopyString(el->A, value, N);**

**el->next = nullptr;**

**}**

**void InitFormStroka(Form\_Stroka& fs) {**

**fs.h = fs.l = nullptr;**

**fs.L = 0;**

**}**

**void InitELV(EL\_V\*& el) {**

**el = new EL\_V;**

**InitFormStroka(el->F);**

**el->next = nullptr;**

**}**

**void InitFormV(Form\_V& fv) {**

**fv.h = fv.cur = fv.last = nullptr;**

**}**

**void InsertString(Form\_Stroka& fs, const char\* value) {**

**EL\_Stroka\* newEl;**

**InitELStroka(newEl, value);**

**if (!fs.h) {**

**fs.h = fs.l = newEl;**

**fs.L = 1;**

**return;**

**}**

**// Вставка в конец (без сортировки)**

**fs.l->next = newEl;**

**fs.l = newEl;**

**fs.L++;**

**}**

**void PrintHierarchy(const Form\_V& list, std::ostream& out, const char\* name) {**

**out << "--- " << name << " ---\n";**

**EL\_V\* level1 = list.h;**

**while (level1) {**

**EL\_Stroka\* level2 = level1->F.h;**

**bool firstElement = true;**

**while (level2) {**

**if (!firstElement) out << " -> ";**

**out << "[" << level2->A << "]";**

**firstElement = false;**

**level2 = level2->next;**

**}**

**level1 = level1->next;**

**if (level1) out << "\n\\/";**

**out << "\n";**

**}**

**out << "----------------\n\n";**

**}**

**Set\_operations.cpp**

**#include "set\_operations.h"**

**#include "list.h"**

**#include "copyS.h"**

**#define MAX\_FULL\_STR 1024**

**static void GetFullString(const EL\_V\* level, char\* buffer, unsigned max) {**

**if (!level || max == 0) {**

**buffer[0] = '\0';**

**return;**

**}**

**unsigned pos = 0;**

**EL\_Stroka\* current = level->F.h;**

**while (current && pos < max - 1) {**

**const char\* part = current->A;**

**unsigned i = 0;**

**while (pos < max - 1 && part[i] != '\0') {**

**buffer[pos++] = part[i++];**

**}**

**current = current->next;**

**}**

**buffer[pos] = '\0';**

**}**

**void Intersection(const Form\_V& s1, const Form\_V& s2, Form\_V& result) {**

**InitFormV(result);**

**EL\_V\* level1 = s1.h;**

**EL\_V\* level2 = s2.h;**

**while (level1 && level2) {**

**char str1[MAX\_FULL\_STR];**

**char str2[MAX\_FULL\_STR];**

**GetFullString(level1, str1, MAX\_FULL\_STR);**

**GetFullString(level2, str2, MAX\_FULL\_STR);**

**int cmp = CompareStrings(str1, str2);**

**if (cmp == 0) {**

**EL\_V\* newLevel = new EL\_V;**

**InitELV(newLevel);**

**EL\_Stroka\* current = level1->F.h;**

**while (current) {**

**InsertString(newLevel->F, current->A);**

**current = current->next;**

**}**

**if (!result.h) {**

**result.h = result.last = newLevel;**

**}**

**else {**

**result.last->next = newLevel;**

**result.last = newLevel;**

**}**

**level1 = level1->next;**

**level2 = level2->next;**

**}**

**else if (cmp < 0) {**

**level1 = level1->next;**

**}**

**else {**

**level2 = level2->next;**

**}**

**}**

**}**

**bool IsSubset(const Form\_V& subset, const Form\_V& mainSet) {**

**EL\_V\* subLevel = subset.h;**

**EL\_V\* mainLevel = mainSet.h;**

**while (subLevel && mainLevel) {**

**char subStr[MAX\_FULL\_STR];**

**char mainStr[MAX\_FULL\_STR];**

**GetFullString(subLevel, subStr, MAX\_FULL\_STR);**

**GetFullString(mainLevel, mainStr, MAX\_FULL\_STR);**

**int cmp = CompareStrings(subStr, mainStr);**

**if (cmp == 0) {**

**subLevel = subLevel->next;**

**mainLevel = mainLevel->next;**

**}**

**else if (cmp < 0) {**

**return false;**

**}**

**else {**

**mainLevel = mainLevel->next;**

**}**

**}**

**return subLevel == nullptr;**

**}**

**Main.cpp**

#include "list.h"

#include "InpS.h"

#include "DelS.h"

#include "set\_operations.h"

#include <iostream>

#include <fstream>

int main() {

Form\_V S1, S2, S3;

setlocale(LC\_ALL, "Russian");

InpThreeSets(S1, S2, S3, "input.txt");

std::ofstream fout("result.txt");

PrintHierarchy(S1, fout, "S1");

PrintHierarchy(S2, fout, "S2");

PrintHierarchy(S3, fout, "S3");

Form\_V intersection;

Intersection(S1, S2, intersection);

PrintHierarchy(intersection, fout, "S1 п S2");

bool isSubset = IsSubset(S3, intersection);

fout << "Результат:\nS3 с (S1 п S2): " << (isSubset ? "Истина" : "Ложь") << "\n";

Delete(S1);

Delete(S2);

Delete(S3);

Delete(intersection);

fout.close();

return 0;

}

# Результат работы программы

1. Истина 2.Ложь

Входной файл: Входной файл:

Aaaaabbbbbccccc aaaaabbbbb

gggggggg gggggggg

dddddeeeeesaddddddddddddd dddddeeeeesaddddddddddddd

dddddeeeeesaddddddddddddd dddddeeeeesaddddddddddddd

aaaaabbbbbccccc aaaaabbbbbccccc

ggggg ggggg

dshfuoishfdoishfoisdhfoixzscvhos dshfuoishfdoishfoisdhfoixzscvhos

aaaaabbbbbccccc aaaaabbbbbccccc

Выходной файл: Выходной файл:

--- S1 --- --- S1 ---

[aaaaa] -> [bbbbb] -> [ccccc] [aaaaa] -> [bbbbb]

\/ \/

[ggggg] -> [ggg] [ggggg] -> [ggg]

\/ \/

[ddddd] -> [eeeee] -> [saddd] -> [ddddd] -> [ddddd] [ddddd] -> [eeeee] -> [saddd] -> [ddddd] -> [ddddd]

\/ \/

[ddddd] -> [eeeee] -> [saddd] -> [ddddd] -> [ddddd] [ddddd] -> [eeeee] -> [saddd] -> [ddddd] -> [ddddd]

----------------

--- S2 --- --- S2 ---

[aaaaa] -> [bbbbb] -> [ccccc] [aaaaa] -> [bbbbb] -> [ccccc]

\/ \/

[ggggg] [ggggg]

\/ \/

[dshfu] -> [oishf] -> [doish] -> [foisd] -> [hfoix] -> [zscvh] -> [os] [dshfu] -> [oishf] -> [doish] -> [foisd] -> [hfoix] -> [zscvh] -> [os]

---------------- ----------------

--- S3 --- --- S3 ---

[aaaaa] -> [bbbbb] -> [ccccc] [aaaaa] -> [bbbbb] -> [ccccc]

---------------- ----------------

--- S1 п S2 --- --- S1 п S2 ---

[aaaaa] -> [bbbbb] -> [ccccc] ----------------

---------------- Результат:

S3 с (S1 п S2): Ложь

Результат:

S3 с (S1 п S2): Истина

# Вывод

В ходе работы реализованы иерархические списки для хранения множеств строк. Разработаны алгоритмы поиска пересечения и проверки подмножества с линейной сложностью Освоены динамические структуры данных.