**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](#_Toc196688810)

[Задание 2](#_Toc196688811)

[Контрольные примеры 2](#_Toc196688812)

[Организация интерфейса пользователя 2](#_Toc196688813)

[Организация входного файла 2](#_Toc196688814)

[Организация выходного файла 3](#_Toc196688815)

[Организация консоли 3](#_Toc196688816)

[Реализация ввода/вывода 3](#_Toc196688817)

[Модульное представление 3](#_Toc196688818)

[Внутреннее представление данных 3](#_Toc196688819)

[Описание внутренних функций 4](#_Toc196688820)

[Определение функций 4](#_Toc196688821)

[Назначение функций 4](#_Toc196688822)

[Описание алгоритма 5](#_Toc196688823)

[Общее описание алгоритма программы 5](#_Toc196688824)

[Принцип работы односвязного списка 5](#_Toc196688825)

[Принцип работы функции removeAllExcept 5](#_Toc196688826)

[Блок-схема 6](#_Toc196688827)

[Текст программы 8](#_Toc196688828)

[Результат работы программы 10](#_Toc196688829)

[Вывод 11](#_Toc196688830)

# 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

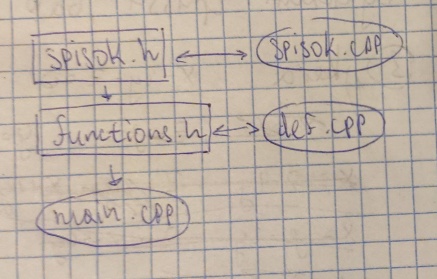
О1. Вывод ошибок.

# 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 |
| SPISOK | class | Node\* head | Головной элемент списка |
| Node\* tail | Хвостовой элемент списка |
| int size | Размер |
| Node | struct | char\* data | Дата |
| Node\* next | Следующий |

|  |  |  |
| --- | --- | --- |
| Имя переменной | Тип | Назначение |
| inputFile, outputFile | const char\* | Имена файлов обработки |
| match, showInfo | bool | Вспомогательная переменная |
| i, len, initialSize, removedCount | int |
| toDelete, this, newNode | Node\* |
| temp |
| current | Текущий элемент |
| prev | Предыдущий элемент |
| buffer[] | char | Буфер |
| lineCount | int | Счет строк |
| markerFound, foundMarker | bool | Истинность маркера |
| spisok | SPISOK | Список |
| out, in | ofstream, ifstream | Выходной и входной файлы |
| marker | char | Маркер |
| valueToSave[] | Заданное значение |

# Описание внутренних функций

## Определение функций

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Принадлежность | Имя функции | Аргументы | | | | Возвращаемое значение |
| Входные | Выходные | Модифи-цируемые | Транзитные |
| - | readMarker | ifstream& in, const char marker |  | ofstream& out, SPISOK& spisok | char buffer[256], int lineCount, bool markerFound | bool |
| processFile | const char\* inputFile, const char\* outputFile |  | ofstream out, ifstream in, SPISOK spisok | char marker, char valueToSave[256], int initialSize, int removedCount |  |
| Class SPISOK | SPISOK |  |  | head, tail, size |  |  |
| ~SPISOK | - |  |  |  |  |
| Push\_back | const char\* data |  | head, tail, size |  |  |
| Clear | - |  | head, tail, size |  |  |
| getSize | - | int |  |  |  |
| removeAllExcept | const char\* value |  | head, tail, size |  |  |
| printToFile | ofstream& out, bool showInfo |  |  |  |  |
| Struct Node | Node | const char\* data, Node\* next |  |  |  |  |
| ~Node | - |  | data |  |  |
| - | main | - |  |  |  |  |

## Назначение функций

Функция readMarker читает строки из входного файла до тех пор, пока не встретит указанный маркер. Каждая строка обрабатывается и сохраняется в список SPISOK. Если маркер найден, функция завершает чтение.

Функция processFile обрабатывает входной файл: читает маркер и целевое значение, вызывает readMarker для чтения данных, удаляет из списка все элементы, кроме тех, которые соответствуют целевому значению, и сохраняет результаты в выходной файл.

SPISOK конструктор, инициализирует список (голову, хвост и размер).

~SPISOK деструктор, очищает список.

Функция getSize возвращает размер списка.

Функция clear очищает список, удаляя все узлы.

Функция removeAllExcept удаляет все элементы списка, кроме тех, которые совпадают с value.

Функция printToFile выводит содержимое списка в файл.

Функция push\_back добавляет новый узел с данными в конец списка.

Node конструктор узла, копирует переданные данные.

~Node деструктор узла, освобождает память данных.

# Описание алгоритма

## Общее описание алгоритма программы

Программа выполняет обработку текстового файла по заданному алгоритму. На первом этапе осуществляется чтение специального маркера из входного файла, который определяет условие прекращения обработки. Следующая за маркером строка принимается в качестве целевого значения для последующего сравнения. Далее программа последовательно считывает строки из файла, сохраняя их в односвязный список, до момента обнаружения строки с указанным маркером. После завершения чтения выполняется фильтрация списка путем удаления всех элементов, не соответствующих целевому значению. Результаты обработки, включая итоговый набор строк и служебную информацию о количестве обработанных и удаленных элементов, записываются в выходной файл.

## Принцип работы односвязного списка

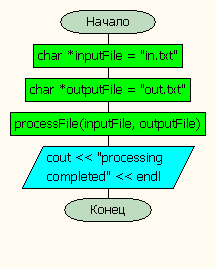
Организация данных реализована в виде односвязного списка, где каждый элемент содержит строковое значение и ссылку на следующий элемент. Основной функционал включает операции добавления новых элементов в конец списка, выборочного удаления элементов по заданному критерию, полной очистки списка и вывода содержимого в файловый поток. Особенностью реализации является строгий контроль за использованием памяти: при удалении отдельных элементов немедленно освобождается выделенная под строки память, а деструктор списка обеспечивает корректное удаление всех оставшихся элементов.

## Принцип работы функции removeAllExcept

Данная функция реализует алгоритм фильтрации списка, сохраняя только элементы, полностью соответствующие заданному значению. Работа начинается с проверки начальных условий - наличия элементов в списке и валидности входного параметра. Основной цикл обработки использует два указателя для последовательного просмотра всех элементов списка. Для каждого элемента выполняется посимвольное сравнение с целевым значением. В случае несоответствия элемент исключается из списка с одновременным освобождением занимаемой памяти и корректировкой связей между соседними элементами. Особое внимание уделяется обработке граничных случаев - удалению первого и последнего элементов списка. В процессе работы функция поддерживает актуальное значение размера списка, уменьшая его при каждом удалении элемента. Результатом является список, содержащий исключительно элементы, идентичные целевому значению, причем алгоритм гарантирует выполнение операции за однократный проход по списку, что обеспечивает оптимальную вычислительную сложность.

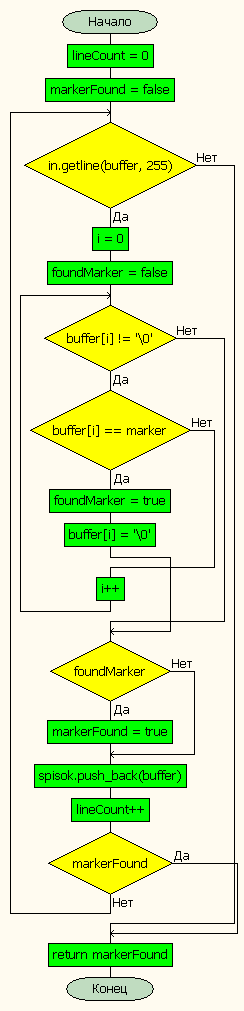
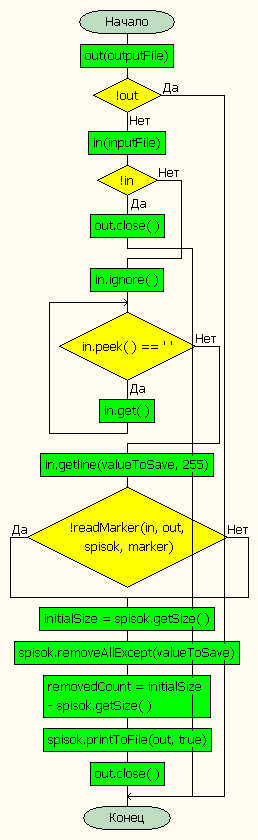
## Блок-схема

Main.cpp



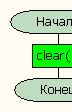
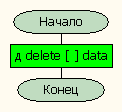
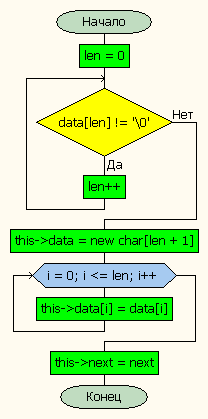
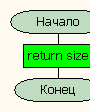
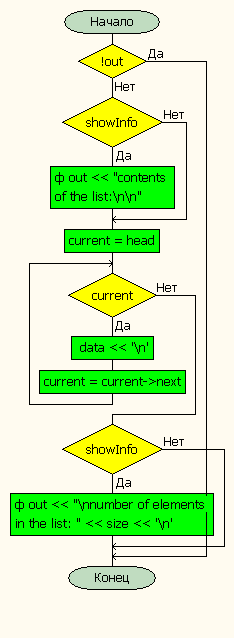
Def.cpp

processFile readMarker

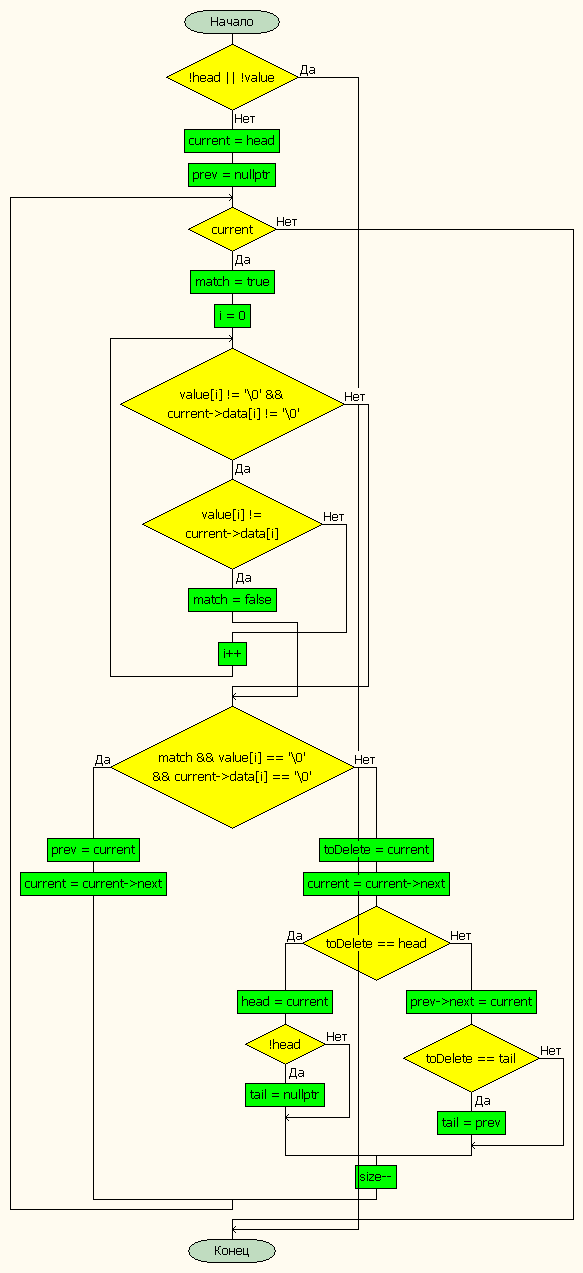
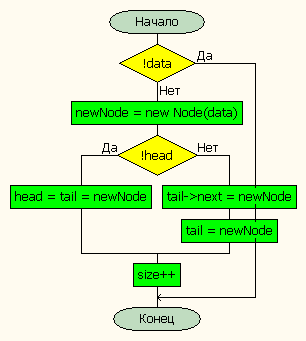
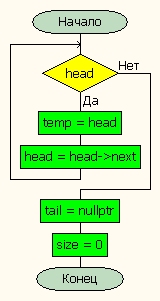


Spisok.cpp

Node ~Node ~SPISOK getSize printToFile

Clear PushBack removeAll



# Текст программы

1. main.cpp

#include "functions.h"

#include <iostream>

using namespace std;

void main()

{

const char\* inputFile = "in.txt";

const char\* outputFile = "out.txt";

processFile(inputFile, outputFile);

cout << "processing completed" << endl;

}

2. spisok.h

#pragma once

#include <fstream>

using namespace std;

class SPISOK

{

public:

SPISOK();

~SPISOK();

void push\_back(const char\* data);

void clear();

int getSize() const;

void removeAllExcept(const char\* value);

void printToFile(ofstream& out, bool showInfo) const;

private:

struct Node

{

char\* data;

Node\* next;

Node(const char\* data, Node\* next = nullptr);

~Node();

};

Node\* head;

Node\* tail;

int size;

};

3. spisok.cpp

#include "spisok.h"

#include <iostream>

SPISOK::SPISOK() : head(nullptr), tail(nullptr), size(0) {}

SPISOK::~SPISOK()

{

clear();

}

int SPISOK::getSize() const

{

return size;

}

void SPISOK::clear()

{

while (head)

{

Node\* temp = head;

head = head->next;

delete temp;

}

tail = nullptr;

size = 0;

}

void SPISOK::removeAllExcept(const char\* value)

{

if (!head || !value) return;

Node\* current = head;

Node\* prev = nullptr;

while (current)

{

bool match = true;

int i = 0;

while (value[i] != '\0' && current->data[i] != '\0')

{

if (value[i] != current->data[i])

{

match = false;

break;

}

i++;

}

if (match && value[i] == '\0' && current->data[i] == '\0')

{

prev = current;

current = current->next;

}

else

{

Node\* toDelete = current;

current = current->next;

if (toDelete == head)

{

head = current;

if (!head) tail = nullptr;

}

else

{

prev->next = current;

if (toDelete == tail) tail = prev;

}

delete toDelete;

size--;

}

}

}

SPISOK::Node::Node(const char\* data, Node\* next)

{

int len = 0;

while (data[len] != '\0') len++;

this->data = new char[len + 1];

for (int i = 0; i <= len; i++)

{

this->data[i] = data[i];

}

this->next = next;

}

SPISOK::Node::~Node()

{

delete[] data;

}

void SPISOK::push\_back(const char\* data)

{

if (!data) return;

Node\* newNode = new Node(data);

if (!head)

{

head = tail = newNode;

}

else

{

tail->next = newNode;

tail = newNode;

}

size++;

}

void SPISOK::printToFile(ofstream& out, bool showInfo) const {

if (!out) return;

if (showInfo)

{

out << "contents of the list:\n\n";

}

Node\* current = head;

while (current)

{

out << "->" << current->data << '\n';

current = current->next;

}

if (showInfo)

{

out << "\nnumber of elements in the list: " << size << '\n';

}

}

4. functions.h

#pragma once

using namespace std;

#include "spisok.h"

#include <fstream>

bool readMarker(ifstream& in, ofstream& out, SPISOK& spisok, const char marker);

void processFile(const char\* inputFile, const char\* outputFile);

5. def.cpp

#include "functions.h"

#include <iostream>

bool readMarker(ifstream& in, ofstream& out, SPISOK& spisok, const char marker)

{

char buffer[256];

int lineCount = 0;

bool markerFound = false;

while (in.getline(buffer, 255))

{

int i = 0;

bool foundMarker = false;

while (buffer[i] != '\0')

{

if (buffer[i] == marker)

{

foundMarker = true;

buffer[i] = '\0';

break;

}

i++;

}

if (foundMarker)

{

markerFound = true;

}

spisok.push\_back(buffer);

out << "-> " << buffer << '\n';

lineCount++;

if (markerFound) break;

}

out << "\nnumber of readed lines: " << lineCount << '\n';

return markerFound;

}

void processFile(const char\* inputFile, const char\* outputFile)

{

SPISOK spisok;

ofstream out(outputFile);

if (!out)

{

cerr << "out file is not open" << endl;

return;

}

ifstream in(inputFile);

if (!in)

{

out << "in file is not open" << endl;

out.close();

return;

}

char marker;

in >> marker;

in.ignore();

while (in.peek() == ' ') in.get();

char valueToSave[256];

in.getline(valueToSave, 255);

out << "target value: \"" << valueToSave << "\"\n\n";

out << "file data:\n\n";

if (!readMarker(in, out, spisok, marker))

{

out << "marker not found" << endl;

}

out << "\nresult of processing:\n\n";

int initialSize = spisok.getSize();

spisok.removeAllExcept(valueToSave);

int removedCount = initialSize - spisok.getSize();

spisok.printToFile(out, true);

out.close();

}

# Результат работы программы

1. Есть заданное значение.

Входной файл:

@

okay wow

dont read

this text

or actually you can try

okay wow

you did it

okay wow

wait wait wait

okay stop

Выходной файл:

target value: "okay wow"

file data:

-> dont read

-> this text

-> or actually you can try

-> okay wow

-> you did it

-> okay wow

->

-> wait wait wait

-> okay stop

number of readed lines: 9

marker not found

result of processing:

contents of the list:

->okay wow

->okay wow

number of elements in the list: 2

1. Нет заданного значения.

Входной файл:

@

okay wow

dont read

this text

or actually you can try

you did it

wait wa@it wait

okay stop

Выходной файл:

target value: "okay wow"

file data:

-> dont read

-> this text

-> or actually you can try

->

-> you did it

->

-> wait wa

number of readed lines: 7

result of processing:

contents of the list:

number of elements in the list: 0

1. Есть маркер.

Входной файл:

@

okay wow

dont read

this text

or actually you can try

okay wow

you did it

okay wow

wait wa@it wait

okay stop

Выхолной файл:

target value: "okay wow"

file data:

-> dont read

-> this text

-> or actually you can try

-> okay wow

-> you did it

-> okay wow

->

-> wait wa

number of readed lines: 8

result of processing:

contents of the list:

->okay wow

->okay wow

number of elements in the list: 2

# Вывод

Были изучены дополнительные возможности классов в языке программирования C++, а также была проведена работа со списками.