**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР.**

отчет

**по лабораторной работе №3**

**по дисциплине «Программирование»**

Тема: «Реализация связного списка».

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1309 |  | Ищенко Д.О. |
| Преподаватель |  | Калмычков В.А. |

Санкт-Петербург

2022

## Оглавление

[Формулировка задания. 2](#_Toc2096133358)

[Анализ задания и формат входных данных. 3](#_Toc1892259213)

[Способ внутреннего представления данных. 3](#_Toc1436172883)

[Способ реализации ввода-вывода. 4](#_Toc856069982)

[Файловая организация программы. 4](#_Toc1630091758)

[Алгоритмы решения задачи. 4](#_Toc917375627)

[Текст программы. 5](#_Toc949551386)

[Результаты работы программы. 9](#_Toc1271017860)

[Вывод. 10](#_Toc2124783326)

## Формулировка задания.

Найти номер первого элемента в списке с заданным значением. При этом элементами списка и искомым элементом являются строки пользовательского типа с маркерным представлением.

## Анализ задания и формат входных данных.

Во входном файле в первой строке находится искомый элемент. В последующих строках элементы списка. Каждая строка первым символом содержит маркер, далее до конца строки или до данного маркера содержится сама строка.

Пример:

\*Ave Maria\*

^Et benedictus^

\_Ave Maria\_sdfsdf

\*Ave Maria\*sdfsdf

Искомым элементом будет строка “Ave Maria” с маркером \*, список будет содержать 3 элемента: строки “Et benedictus” с маркером ^, Ave Maria” с маркером \_ и “Ave Maria” с маркером \*. Результатом работы программы должно быть число 3. То есть тип маркера при сравнении строк имеет значение, а нумерация элементов списка при поиске искомого начинается с единицы.

## Способ внутреннего представления данных.

Внутреннее хранение данных реализовано с использованием трех пользовательских типов: Node, LinkedList и StrM.

StrM является реализацией строкового представления массива символов с хранением маркера окончания строки. Для доступа к данным строки реализованы два метода: getData и getMarker. Также перегружен операторы считывания и ввода данных с потока и оператор сравнения.

Node - это структура, содержащая два поля: поле данных и указатель на следующую ноду. Node является приватной вложенной структурой класса LinkedList.

LinkedList - реализация односвязного списка. Единственным закрытым полем является указатель на первый элемент списка типа Node. Для работы со списком реализованы методы find, print и push\_back. LinkedList реализован как шаблонный класс.

## Способ реализации ввода-вывода.

Для класса StrM перегружен операторы ввода и вывода. При этом, поскольку добавление данных в список происходит через метод push\_back, для класса LinkedList реализован только метод print для вывода всего списка в файл.

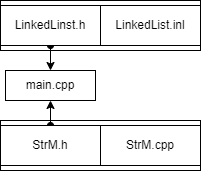
## Файловая организация программы.

**Main.cpp** содержит определение функций main и process\_data. В функции process\_data реализован алгоритм решения поставленной задачи с использованием описанных выше типов внутреннего хранения данных. См пункт «Алгоритмы решения задачи».

**StrM.h** содержит объявление класса StrM без реализации методов, их определение и некоторых вспомогательных функций вынесено в файл **StrM.cpp**.

Аналогично, объявление класса LinkedList содержится в **LinkedList.h,** а реализация его методов в **LinkedList.inl** (inl - из-за особенностей реализации шаблонных классов).

Порядок подключения файлов см на рис. 2. Проблема двойного включения файлов и дублирования кода решается с помощью использования директив препроцессора pragma\_once и ifndef-define-endif.



## Алгоритмы решения задачи.

Ниже проиллюстрирован принцип работы функций main и proccess\_data, которые отвечают за выполнение программы в целом и обработку данных соответственно.

## Текст программы.

**Main.cpp:**

#include <iostream>

#include <fstream>

#include "StrM.h"

#include "LinkedList.h"

void process\_data(std::fstream& in, std::fstream& out) {

StrM tmp, trgt;

in >> trgt;

LinkedList<StrM> l;

while (!in.eof()) {

in >> tmp;

l.push\_back(tmp);}

out << "Linked list data:\n\n";

l.print(out);

int res = l.find(trgt);

out << "\n<<";

if (res != -1)

out << trgt << ">> was found in " << res << " position.";

else

out << trgt << ">> wasn't found.";}

int main(){

std::fstream in("input.txt"),

out("output.txt", std::fstream::trunc | std::fstream::out);

if (in.is\_open() && out.is\_open())

process\_data(in, out);

else

std::cout << "files not open";

in.close(); out.close(); }

**LinkedList.h**

#ifndef LINKEDLIST\_H

#define LINKEDLIST\_H

#include <fstream>

template <class T>

class LinkedList {

struct Node {

T data;

Node\* next;

};

Node\* head = nullptr;

public:

LinkedList(const T\* arr, int len);

LinkedList() {}

void push\_back(T data);

void print(std::fstream& out);

int find(T trgt);

~LinkedList();

};

#include "LinkedList.inl"

#endif // !LINKEDLIST

**Linkedlist.inl**

#ifndef LINKEDLIST\_INL

#define LINKEDLIST\_INL

#include "LinkedList.h"

#include <fstream>

template <class T>

LinkedList<T>::LinkedList(const T\* arr, int len) {

head = new Node{ arr[0], nullptr };

Node\* It = head;

for (int i = 1; i < len; i++) {

Node\* new\_node = new Node{ arr[i], nullptr };

It->next = new\_node;

It = It->next; }}

template <class T>

void LinkedList<T>::push\_back(T data) {

Node\* last = head;

if (last == nullptr) {

head = new Node{ data, nullptr };

return; }

while (last->next != nullptr)

last = last->next;

last->next = new Node{ data, nullptr };}

template <class T>

void LinkedList<T>::print(std::fstream& out) {

if (head == nullptr) return;

Node\* It = head;

for (; It->next != nullptr; It = It->next)

out << It->data << '\n';

out << It->data << '\n';}

template <class T>

int LinkedList<T>::find(T trgt) {

Node\* It = head;

int i = 1;

for (; It->next != nullptr; It = It->next, i++)

if (It->data == trgt)

return i;

if (It->data == trgt)

return i;

return -1;}

template <class T>

LinkedList<T>::~LinkedList() {

Node\* next = head;

while (head != nullptr) {

next = head->next;

delete head;

head = next;}}

#endif // !LINKEDLIST

**StrL.h**

#pragma once

#include <fstream>

#include <stdarg.h>

#include "Globals.h"

class StrL{

char s[WORD\_LEN];

unsigned len = 0;

public:

StrL(const char\* s, unsigned len);

explicit StrL(const char ch, ...);

StrL() { }

const char\* get\_s() const { return s; }

const unsigned get\_len() const { return len; }

void set\_word(const char\* str);

void set\_len(unsigned n) { len = n; }

char get\_char(unsigned n) const { if (n < len) return s[n]; }};

std::fstream& operator<<(std::fstream& out, const StrL& s);

std::fstream& operator>>(std::fstream& in, StrL& s);

bool operator == (const StrL& a, const StrL& b);

**StrL.cpp**

#include "StrL.h"

std::fstream& operator<<(std::fstream& out, const StrL& s) {

const char\* word = s.get\_s();

for (int i = 0; i < s.get\_len(); i++) {

out.put(word[i]);}

return out;}

void StrL::set\_word(const char\* str) {

for (int i = 0; i < WORD\_LEN; i++)

s[i] = str[i];}

std::fstream& operator>>(std::fstream& in, StrL& s) {

char word[WORD\_LEN];

in >> word;

s.set\_word(word);

char t = word[0]; unsigned count = 0;

while (t != '\0') {

count++;

t = word[count];}

s.set\_len(count);

return in;}

StrL::StrL(const char ch, ...) {

len++;

s[0] = ch;

va\_list list;

va\_start(list, ch);

while (true) {

char t = va\_arg(list, char);

if (t == '\0')

break;

s[len] = t;

len++;}

va\_end(list);}

StrL::StrL(const char\* s, unsigned len) : len(len) {

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

this->s[i] = s[i];}}

bool operator == (const StrL& a, const StrL& b) {

if (a.get\_len() != b.get\_len())

return false;

for (int i = 0; i < a.get\_len(); i++) {

if (a.get\_char(i) != b.get\_char(i))

return false;}

return true;}

## Результаты работы программы.

**Input.txt:**

\*Ave Maria\*

^Et benedictus^

\_Ave Maria\_sdfsdf

\_Ave, ave dominus\_

\_Et benedictus\_

\_Ventris tui, Iesus\_

\_Ave Maria\_

\*Ave Maria\*

\_Maiden mild\_

\_I listen to a maidens prayer\_

\_Oh, mother hear a suppliant child\_

\*Ave Maria\*

**Output.txt**

Linked list data:

Et benedictus^

Ave Maria\_

Ave, ave dominus\_

Et benedictus\_

Ventris tui, Iesus\_

Ave Maria\_

Ave Maria\*

Maiden mild\_

I listen to a maidens prayer\_

Oh, mother hear a suppliant child\_

Ave Maria\*

<<Ave Maria\*>> was found in 7 position.

## Вывод.

В ходе выполнения лабораторной работы была написана программа, решающая поставленную задачу. Во время её разработки были получены практические навыки работы с различными конструкциями языка с++, такими как директивы препроцессора, перегрузка операторов, отделение объявления от определения методов. А также был получен навык реализации пользовательского односвязного списка и функции поиска в нем.