**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №4**

**по дисциплине «Программирование»**

Студент гр. 1302 Гордиенко М. Ю.

Преподаватель Калмычков В. А.

Санкт-Петербург

2022

Оглавление

[Цель работы 3](#_Toc104977679)

[Формулировка задания 3](#_Toc104977680)

[Контрольный пример 3](#_Toc104977681)

[Математическая постановка задачи 4](#_Toc104977682)

[Макет вывода 4](#_Toc104977683)

[Способы реализации ввода-вывода 4](#_Toc104977684)

[Написанные функции: 4](#_Toc104977685)

[Разбиение программы на файлы 4](#_Toc104977686)

[Блок-схема 4](#_Toc104977687)

[Текст программы 7](#_Toc104977688)

[Результат работы программы: 14](#_Toc104977689)

[Вывод: 14](#_Toc104977690)

Цель работы: научиться использовать вложенные связные списки и разбивать программу на несколько файлов при написании программ разного вида.

Формулировка задания: вставить заданное число элементов после всех элементов, имеющих заданное значение.

Изображение выглядит как текст

Автоматически созданное описание

Контрольный пример:

in1.txt:

aaa

bbb

ссс

dddd

eeeee

in2.txt:

bbb

dddd

kkk

mmm

out.txt:

aaa

ccc

eeeee

Математическая постановка задачи:

Дано: списки со словами A, B

Найти: A\B

**Макет ввода:**

Строки, содержащиеся в списках

Макет вывода:

Контрольный вывод, разность множеств

Способы реализации ввода-вывода:

|  |  |  |
| --- | --- | --- |
|  | iostream | fstream |
| Ввод | cin | ios::in |
| Вывод | cout | ios::out |

Написанные функции:

**add** – добавляет элемент в список.

**copy** – копирует список.

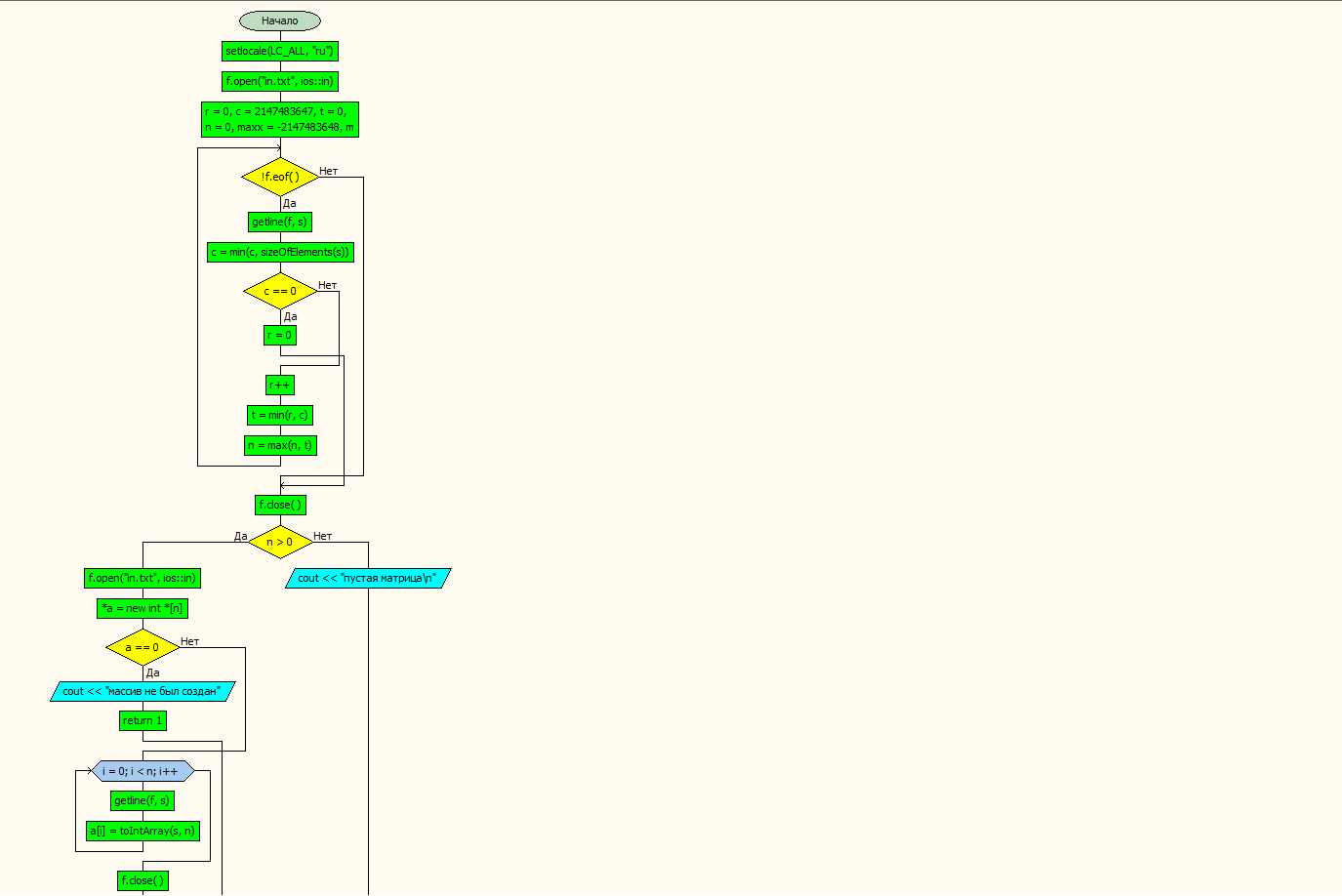
**set –** вставляет по индексу заданное значкние

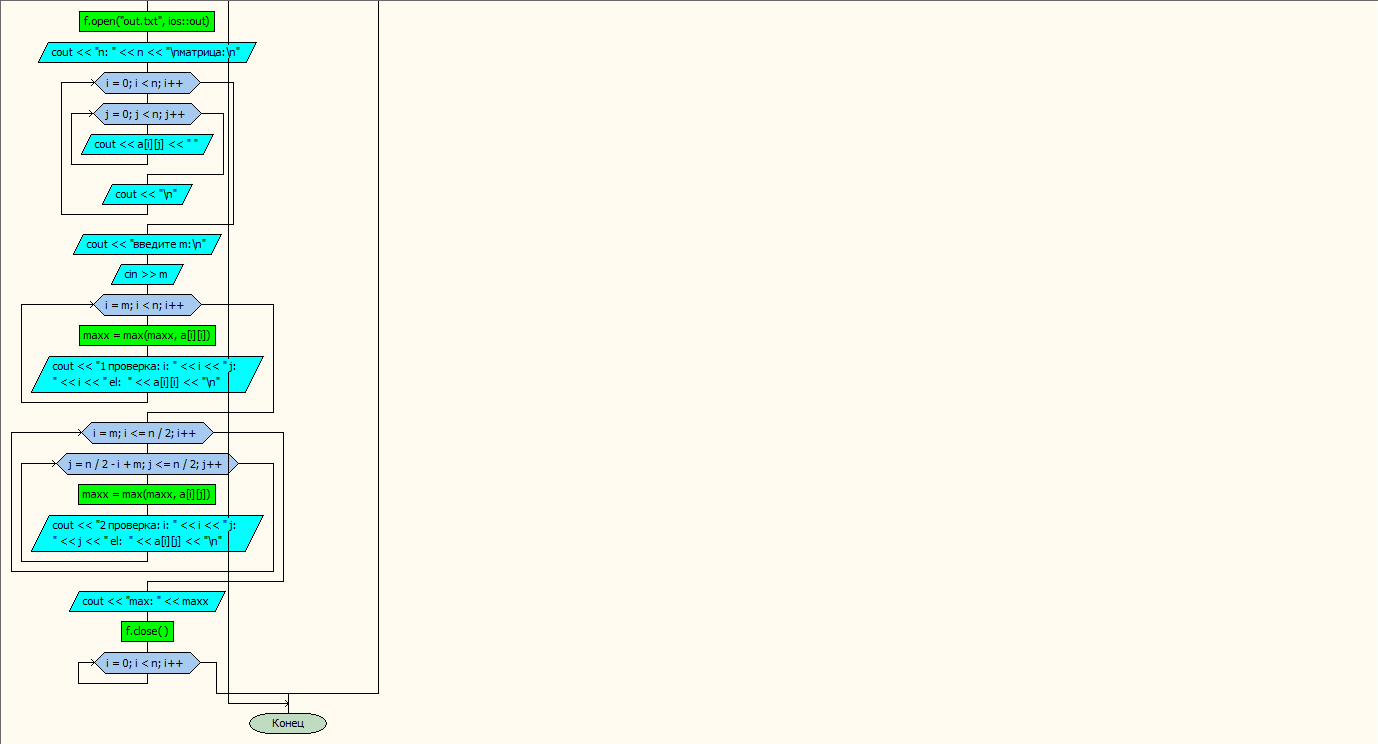
**addList** – вставляет список по индексу.

# **Разбиение программы на файлы**

|  |  |
| --- | --- |
| main.cpp | Основа программы |
| StrL.cpp | Структура слова, из которых состоит список |
| StrL.h | Интерфейс структуры слова |
| LinkedList.h | Интерфейс списка вместе с реализацией |
| LinkedList.cpp | Остается пустым, так как шаблонные классы реализуются в заголовочном файле |

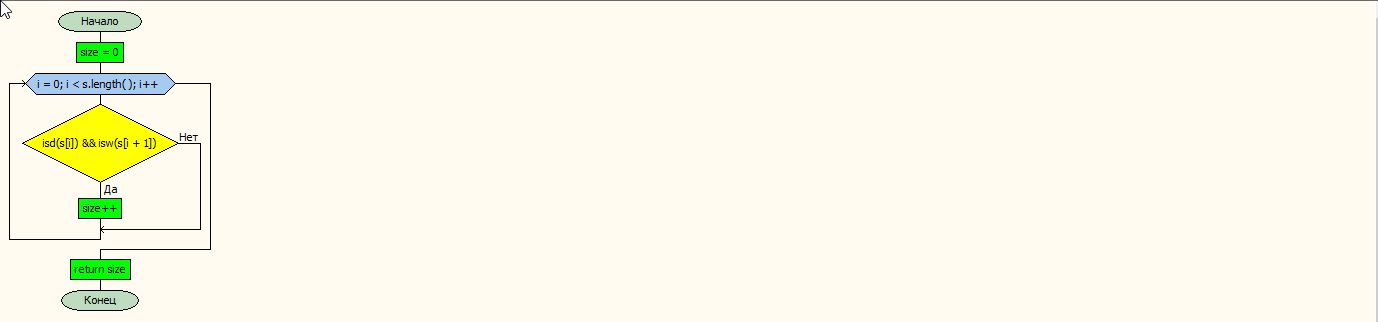
Блок-схема:

Main

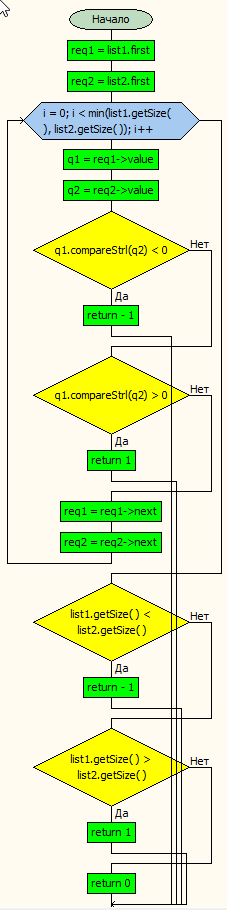


copy

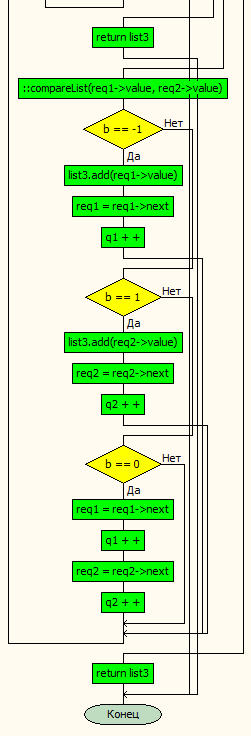
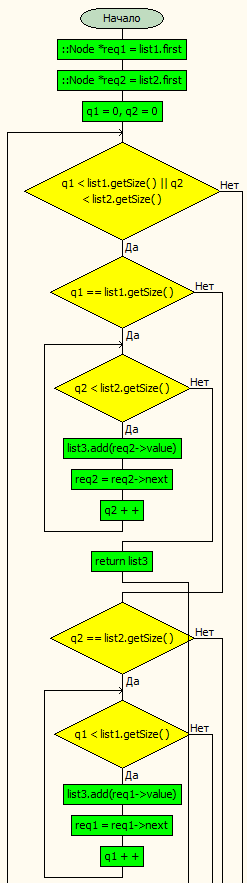
add



compareList



diff



Текст программы:

**main.cpp**

#include <iostream>  
#include <fstream>  
#include <string>  
#include "LinkedList.h"  
#include "StrL.h"  
  
using namespace std;  
  
void read(fstream &f, LinkedList<LinkedList<StrL>> &list) {  
 const int N = 10;  
 string s;  
 while (!f.eof()) {  
 getline(f, s);  
 LinkedList<StrL> a;  
 for (int i = 0; i < (s.length() + N - 1) / N; i++) {  
 a.add(s.substr(i \* N, N));  
 }  
 list.add(a);  
 }  
}  
  
int main() {  
 setlocale(LC\_ALL, "ru.utf-8");  
 LinkedList<LinkedList<StrL>> list1;  
 LinkedList<LinkedList<StrL>> list2;  
 LinkedList<LinkedList<StrL>> list3;  
 fstream f;  
 f.open("in1.txt", ios::in); //ОБА СПИСКА ВВОДЯТСЯ В ЛЕКСИКОГРАФИЧЕСКОМ ПОРЯДКЕ. ОНИ НЕ СОРТИРУЮТСЯ  
 read(f, list1);  
 f.close();  
 f.open("in2.txt", ios::in);  
 read(f, list2);  
 f.close();  
 f.open("out.txt", ios::out);  
 LinkedList<StrL> \*a1 = list1.toArray();  
 LinkedList<StrL> \*a2 = list2.toArray();  
 f << "контрольный вывод списка 1:\n\n";  
 for(int i = 0; i < list1.getSize(); i++){  
 StrL \*q = a1[i].toArray();  
 for (int j = 0; j < a1[i].getSize(); j++){  
 f << q[j].arr << ' ';  
 }  
 f << "\n↓\n";  
 }  
 if (list1.getSize()==0){  
 f << "пустой список\n";  
 }  
 f << '\n';  
 f << "контрольный вывод списка 2:\n\n";  
 for(int i = 0; i < list2.getSize(); i++){  
 StrL \*q = a2[i].toArray();  
 for (int j = 0; j < a2[i].getSize(); j++){  
 f << q[j].arr << ' ';  
 }  
 f << "\n↓\n";  
 }  
 if (list2.getSize()==0){  
 f << "пустой список\n";  
 }  
 f << '\n';  
 list3 = LinkedList<StrL>::diff(list1, list2);  
 LinkedList<StrL> \*aX = list3.toArray();  
 f << '\n';  
 f << "разность списков 1 и 2:\n\n";  
 for(int i = 0; i < list3.getSize(); i++){  
 StrL \*q = aX[i].toArray();  
 for (int j = 0; j < aX[i].getSize(); j++){  
 f << q[j].arr << ' ';  
 }  
 f << "\n↓\n";  
 }  
 if (list3.getSize()==0){  
 f << "пустой список\n";  
 }  
 f.close();  
 delete a1;  
 delete a2;  
 list1.clear();  
 list2.clear();  
 list3.clear();  
 return 0;  
}

**LinkedList.h**

#pragma once  
#include <iostream>  
#include <cassert>  
#include "StrL.h"  
  
using namespace std;  
  
template<typename T>  
struct LinkedList {  
  
 struct Node {  
 T value;  
 Node \*next;  
  
 Node(T value, Node \*next) {  
 this->value = value;  
 this->next = next;  
 };  
 };  
  
 int size = 0;  
 Node \*first;  
 Node \*last;  
  
 int getSize();  
  
 void addFirst(T value);  
  
 void add(T value);  
  
 void add(T value, int ind);  
  
 void insertList(LinkedList<T> a, int ind);  
  
 void set(T value, int ind);  
  
 T get(int ind);  
  
 LinkedList<T> copy();  
  
 void clear();  
  
 void print();  
  
 T \*toArray();  
  
 static int compareList(LinkedList<StrL> list1, LinkedList<StrL> list2);  
  
 static LinkedList<LinkedList<StrL>>  
 diff(LinkedList<LinkedList<StrL>> list1, LinkedList<LinkedList<StrL>> list2);  
};  
  
template<typename T>  
int LinkedList<T>::getSize() {  
 return size;  
}  
  
template<typename T>  
void LinkedList<T>::addFirst(T value) {  
 if (size > 0) {  
 first = new Node(value, first);  
 } else {  
 first = new Node(value, nullptr);  
 last = first;  
 }  
 size++;  
}  
  
template<typename T>  
void LinkedList<T>::add(T value) {  
 if (size > 0) {  
 last->next = new Node(value, nullptr);  
 last = last->next;  
 } else {  
 first = new Node(value, nullptr);  
 last = first;  
 }  
 size++;  
}  
  
template<typename T>  
void LinkedList<T>::add(T value, int ind) {  
 assert(ind <= size && ind > -1);  
 if (ind == 0) {  
 addFirst(value);  
 return;  
 }  
 if (ind == size) {  
 add(value);  
 return;  
 }  
 Node \*req = first;  
 for (int i = 0; i < ind - 1; i++) {  
 req = req->next;  
 }  
 Node \*temp = new Node(value, req->next);  
 req->next = temp;  
 size++;  
  
}  
  
template<typename T>  
void LinkedList<T>::insertList(LinkedList<T> a, int ind) {  
 assert(ind <= size && ind > -1);  
 LinkedList<T> b = a.copy();  
 if (size == 0) {  
 first = b.first;  
 last = b.last;  
 size = b.size;  
 } else {  
 if (ind == 0) {  
 first = b.first;  
 b.last->next = first;  
 } else if (ind == size) {  
 last->next = b.first;  
 last = b.last;  
 } else {  
 Node \*req = first;  
 for (int i = 0; i < ind - 1; i++) {  
 req = req->next;  
 }  
 b.last->next = req->next;  
 req->next = b.first;  
 }  
 size += b.size;  
 }  
}  
  
template<typename T>  
void LinkedList<T>::set(T value, int ind) {  
 assert(ind < size && ind > -1);  
 Node \*req = first;  
 for (int i = 0; i < ind; i++) {  
 req = req->next;  
 }  
 req->value = value;  
}  
  
template<typename T>  
T LinkedList<T>::get(int ind) {  
 assert(ind < size && ind > -1);  
 Node \*req = first;  
 for (int i = 0; i < ind; i++) {  
 req = req->next;  
 }  
 return req->value;  
}  
  
template<typename T>  
LinkedList<T> LinkedList<T>::copy() {  
 LinkedList<T> b;  
 Node \*req = first;  
 for (int i = 0; i < size; i++) {  
 b.add(req->value);  
 req = req->next;  
 }  
 return b;  
}  
  
template<typename T>  
void LinkedList<T>::clear() {  
 while (size > 0) {  
 Node \*second = first->next;  
 delete first;  
 first = second;  
 size--;  
 }  
}  
  
template<typename T>  
void LinkedList<T>::print() {  
 Node \*req = first;  
 for (int i = 0; i < size; i++) {  
 cout << (req->value);  
 req = req->next;  
 }  
 cout << '\n';  
}  
  
template<typename T>  
T \*LinkedList<T>::toArray() {  
 T \*a = new T[getSize()];  
 Node \*req = first;  
 for (int i = 0; i < size; i++) {  
 a[i] = (req->value);  
 req = req->next;  
 }  
 return a;  
}  
  
template<typename T>  
int LinkedList<T>::compareList(LinkedList<StrL> list1, LinkedList<StrL> list2) {  
 Node \*req1 = list1.first;  
 Node \*req2 = list2.first;  
 for (int i = 0; i < min(list1.getSize(), list2.getSize()); i++) {  
 StrL q1 = req1->value;  
 StrL q2 = req2->value;  
 if (q1.compareStrl(q2) < 0) {  
 return -1;  
 }  
 if (q1.compareStrl(q2) > 0) {  
 return 1;  
 }  
 req1 = req1->next;  
 req2 = req2->next;  
 }  
 if (list1.getSize() < list2.getSize())  
 return -1;  
 if (list1.getSize() > list2.getSize())  
 return 1;  
 return 0;  
}  
  
template<typename T>  
LinkedList<LinkedList<StrL>>  
LinkedList<T>::diff(LinkedList<LinkedList<StrL>> list1, LinkedList<LinkedList<StrL>> list2) {  
 LinkedList<LinkedList<StrL>> list3;  
 LinkedList<LinkedList<StrL>>::Node \*req1 = list1.first;  
 LinkedList<LinkedList<StrL>>::Node \*req2 = list2.first;  
 int q1 = 0, q2 = 0;  
 while (q1 < list1.getSize() || q2 < list2.getSize()) {  
 if (q1 == list1.getSize()) {  
 return list3;  
 }  
 if (q2 == list2.getSize()) {  
 while (q1 < list1.getSize()) {  
 list3.add(req1->value);  
 req1 = req1->next;  
 q1++;  
 }  
 return list3;  
 }  
 int b = LinkedList<StrL>::compareList(req1->value, req2->value);  
 if (b == -1){  
 list3.add(req1->value);  
 req1 = req1->next;  
 q1++;  
 continue;  
 }  
 if (b == 1){  
 //list3.add(req2->value);  
 req2 = req2->next;  
 q2++;  
 continue;  
 }  
 if (b == 0){  
 req1 = req1->next;  
 q1++;  
 req2 = req2->next;  
 q2++;  
 }  
 }  
 return list3;  
}

}

**StrL.h**

#pragma once  
  
#include <string>  
#include <iostream>  
  
using namespace std;  
  
struct StrL {  
 char \*arr;  
 int len;  
  
 StrL(string s);  
  
 StrL();  
  
 int compareStrl(StrL s);  
};

**StrL.cpp**

#include "StrL.h"  
  
using namespace std;  
  
StrL::StrL(string s) {  
 arr = new char[s.length() + 1];  
 strcpy(arr, s.c\_str());  
 len = s.length();  
}  
  
StrL::StrL() {  
  
}  
  
int StrL::compareStrl(StrL s) {  
 for (int i = 0; i < min(len, s.len); i++) {  
 if (arr[i] < s.arr[i]) {  
 return -1;  
 }  
 if (arr[i] > s.arr[i]) {  
 return 1;  
 }  
 }  
 if (len < s.len) {  
 return -1;  
 }  
 if (len > s.len) {  
 return 1;  
 }  
 return 0;  
}

# Результат работы программы:

in1.txt:

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
bbbbbbbbbbbbbbbbbbbbbbbb  
ccc

in2.txt:

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
ddd  
fff

out.txt:

контрольный вывод списка 1:  
  
aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa aa   
↓  
bbbbbbbbbb bbbbbbbbbb bbbb   
↓  
ccc   
↓  
  
контрольный вывод списка 2:  
  
aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa aa   
↓  
ddd   
↓  
fff   
↓  
  
  
разность списков 1 и 2:  
  
bbbbbbbbbb bbbbbbbbbb bbbb   
↓  
ccc   
↓

# Вывод:

Я научился работать со связными списками, разбивать программу на файлы и использовать полученные знания для решения задач.