**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 Новиков Г.В.

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

Изображение выглядит как текст

Автоматически созданное описание

Контрольный пример:

in1.txt:

first line line line line line line line line line line line line

second line

A line of 20 symbols

A line of 21 symbolsE

E

in2.txt:

second file

second line

A line of 20 symbols

in3.txt:

first line line line line line line line line line line line line

A line of 21 symbolsE

E

out.txt:

LIST 1

0. "first line line line"->" line line line line"->" line line line line"->" line"->NULL

-> 1. "second line"->NULL

-> 2. "A line of 20 symbols"->NULL

-> 3. "A line of 21 symbols"->"E"->NULL

-> 4. "E"->NULL

-> 5. ""->NULL

-> NULL

LIST 2

0. "second file"->NULL

-> 1. "second line"->NULL

-> 2. "A line of 20 symbols"->NULL

-> NULL

LIST 3

0. "first line line line"->" line line line line"->" line line line line"->" line"->NULL

-> 1. "A line of 21 symbols"->"E"->NULL

-> 2. "E"->NULL

-> NULL

LIST 1 \ LIST 2

0. "first line line line"->" line line line line"->" line line line line"->" line"->NULL

-> 1. "A line of 21 symbols"->"E"->NULL

-> 2. "E"->NULL

-> 3. ""->NULL

-> NULL

LIST 3 belongs LIST 1 \ LIST 2: true

Математическая постановка задачи:

Дано: списки со словами A, B, C

Найти: A\B, проверить истинность утверждения

**Макет ввода:**

Строки, содержащиеся в списках

Макет вывода:

Контрольный вывод, проверить истинность утверждения

Способы реализации ввода-вывода:

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

Написанные функции:

subtract – вычитание списков

belongs – принадлежность списку.

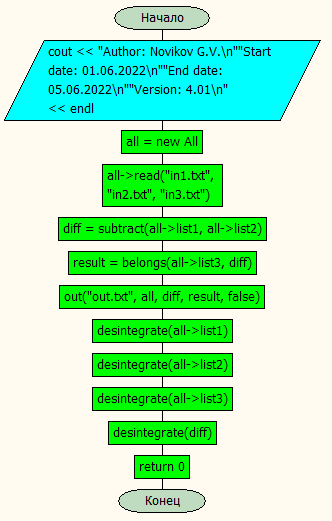
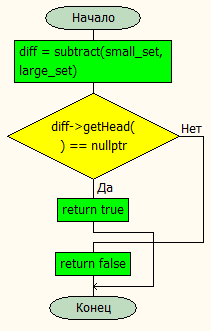
copy – копирует список.

el\_is\_in -проверяет принадлежит ли элемент списку

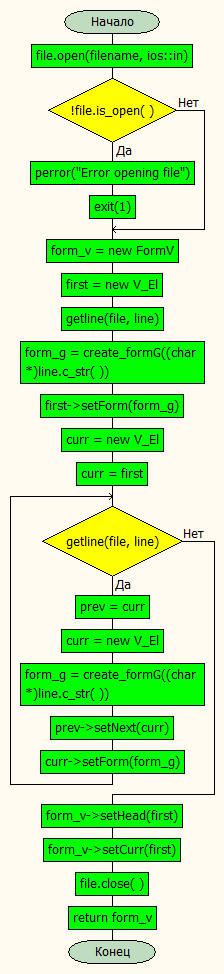
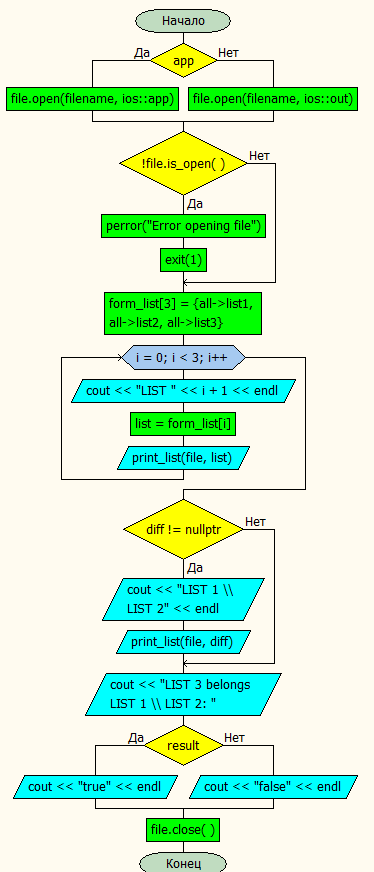
# **Разбиение программы на файлы**

|  |  |
| --- | --- |
| 4.cpp | Основа программы |
| read.h, read.cpp | Чтение из файла или из консоли |
| out.h, out.cpp | Вывод в файл |
| functions.h, functions.cpp | Функции обработки |
| FormV.h, FormV.cpp | Вертикальный формуляр |
| V\_El.h, V\_El.cpp | Вертикальный элемент |
| FormG.h, FormG.cpp | Горизонтальный формуляр |
| G\_El.h, G\_El.cpp | Горизонтальный элемент |
| Str.h, Str.cpp | Строка |
| In1.txt, in2.txt, in3.txt | Ввод |
| out.txt | Вывод |

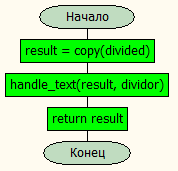
Блок-схема:

Main

belongs

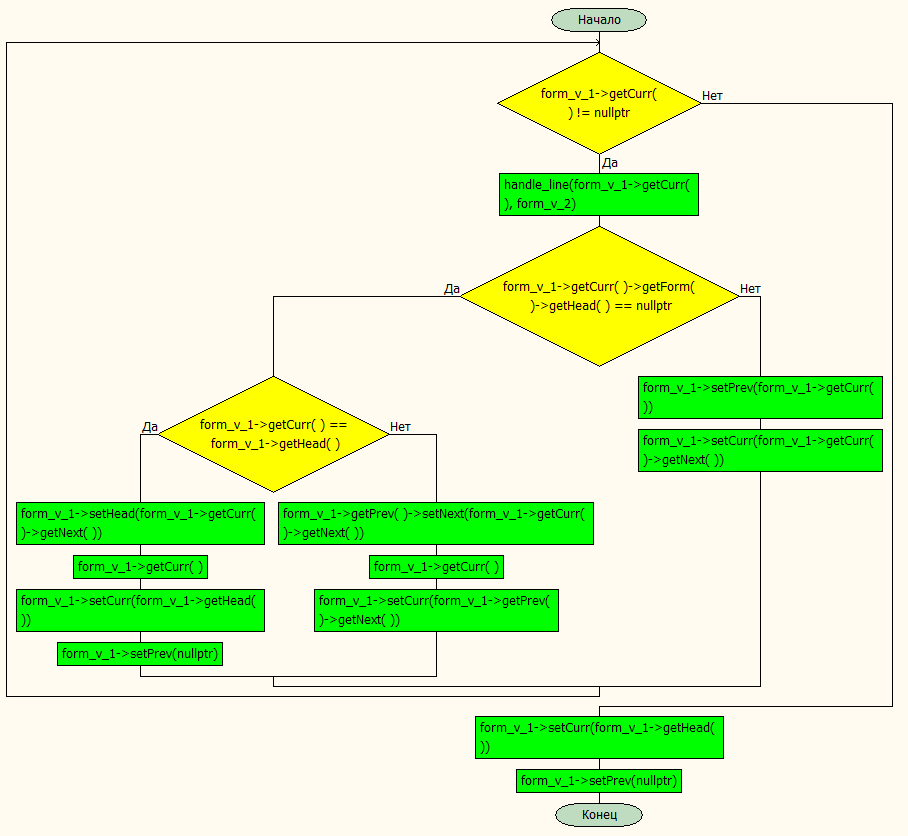


out



subtract

read



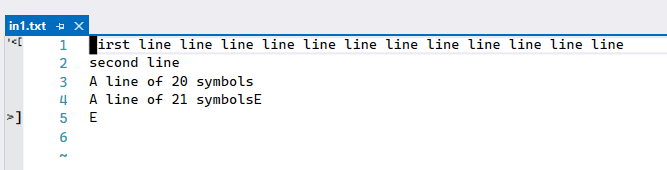
Handle\_text

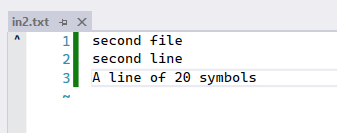
Текст программы:

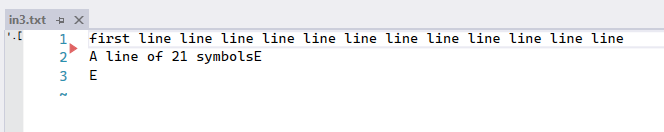
|  |  |  |  |
| --- | --- | --- | --- |
| Main.cpp() | All.h | formG.h | formV.h |
| #include "functions.h"  int main()  {  cout << "Author: Novikov G.V.\n"  "Start date: 01.06.2022\n"  "End date: 05.06.2022\n"  "Version: 4.01\n" << endl;  All\* all = new All;  all->read("in1.txt", "in2.txt", "in3.txt");  FormV\* diff = subtract(all->list1, all->list2);  bool result = belongs(all->list3, diff);  out("out.txt", all, diff, result, false);  desintegrate(all->list1);  desintegrate(all->list2);  desintegrate(all->list3);  desintegrate(diff);  delete all;  return 0;  } | #pragma once  #include "FormV.h"  class All {  public:  FormV\* list1 = nullptr;  FormV\* list2 = nullptr;  FormV\* list3 = nullptr;  void read(string filename1, string filename2, string filename3);  };  FormV\* read\_one(string filename);  FormG\* create\_formG(char\* line);  G\_El\* create\_G\_El(char\* line, unsigned int i);  Str\* create\_str(char\* line, unsigned int i); | #pragma once  #include "G\_El.h"  class FormG {  private:  G\_El\* head = nullptr;  G\_El\* curr = nullptr;  G\_El\* prev = nullptr;  public:  G\_El\* getHead();  void setHead(G\_El\* new\_head);  G\_El\* getCurr();  void setCurr(G\_El\* new\_curr);  G\_El\* getPrev();  void setPrev(G\_El\* new\_prev);  }; | #pragma once  #include "V\_El.h"  class FormV {  private:  V\_El\* head = nullptr;  V\_El\* curr = nullptr;  V\_El\* prev = nullptr;  public:  V\_El\* getHead();  void setHead(V\_El\* new\_head);  V\_El\* getCurr();  void setCurr(V\_El\* new\_curr);  V\_El\* getPrev();  void setPrev(V\_El\* new\_prev);  }; |
| Functions.h | Str.h | V\_El.h | G\_El.h |
| #pragma once  #include "All.h"  bool belongs(FormV\* small\_set, FormV\* large\_set);  FormV\* subtract(FormV\* form\_v\_1, FormV\* form\_v\_2);  bool el\_is\_in(G\_El\* target\_el, FormV\* form\_v);  void out(string filename, All\* all, FormV\* diff, bool result, bool app);  void print\_g\_el(fstream& file, G\_El\* g\_el);  void print\_v\_el(fstream& file, V\_El\* v\_el, int num);  void print\_list(fstream& file, FormV\* list);  void desintegrate(FormV\* form\_v); | #pragma once  #include <iostream>  #include <fstream>  #include <string>  using namespace std;  const int MAX\_SIZE = 20;  class Str {  private:  char letters[MAX\_SIZE];  int len = MAX\_SIZE;  public:  int getLen();  void setLen(unsigned int new\_len);  char getLetter(int i);  void setLetter(int i, char new\_letter);  bool equals(Str\* str);  }; | #pragma once  #include "FormG.h"  class V\_El{  private:  FormG\* form = nullptr;  V\_El\* next = nullptr;  public:  FormG\* getForm();  void setForm(FormG\* new\_form);  V\_El\* getNext();  void setNext(V\_El\* new\_next);  }; | #pragma once  #include "Str.h"  class G\_El {  private:  Str\* str;  G\_El\* next = nullptr;  public:  G\_El\* getNext();  void setNext(G\_El\* new\_next);  Str\* getStr();  void setStr(Str\* new\_str);  bool equals(G\_El\* el);  }; |
| FormG.cpp | FormV.cpp | V\_El.cpp | G\_El.cpp |
| #include "FormG.h"  G\_El\* FormG::getHead() {  return head;  };  void FormG::setHead(G\_El\* new\_head) {  head = new\_head;  };  G\_El\* FormG::getCurr() {  return curr;  };  void FormG::setCurr(G\_El\* new\_curr) {  curr = new\_curr;  };  G\_El\* FormG::getPrev() {  return prev;  };  void FormG::setPrev(G\_El\* new\_prev) {  prev = new\_prev;  }; | #include "FormV.h"  V\_El\* FormV::getHead() {  return head;  };  void FormV::setHead(V\_El\* new\_head) {  head = new\_head;  };  V\_El\* FormV::getCurr() {  return curr;  };  void FormV::setCurr(V\_El\* new\_curr) {  curr = new\_curr;  };  V\_El\* FormV::getPrev() {  return prev;  };  void FormV::setPrev(V\_El\* new\_prev) {  prev = new\_prev;  }; | #include "V\_El.h"  FormG\* V\_El::getForm() {  return form;  };  void V\_El::setForm(FormG\* new\_form) {  form = new\_form;  };  V\_El\* V\_El::getNext() {  return next;  };  void V\_El::setNext(V\_El\* new\_next) {  next = new\_next;  }; | #include "G\_El.h"  G\_El\* G\_El::getNext() {  return next;  };  void G\_El::setNext(G\_El\* new\_next) {  next = new\_next;  };  Str\* G\_El::getStr() {  return str;  };  void G\_El::setStr(Str\* new\_str) {  str = new\_str;  };  bool G\_El::equals(G\_El\* el) {  return getStr()->equals(el->getStr());  } |
| Out.cpp | Read.cpp | Str.cpp | functions.cpp |
| #include "functions.h"  void print\_g\_el(fstream& file, G\_El\* g\_el) {  cout << '"';  file << '"';  for (int i = 0; i < g\_el->getStr()->getLen(); i++) {  file << g\_el->getStr()->getLetter(i);  cout << g\_el->getStr()->getLetter(i);  }  cout << '"';  file << '"';  }  void print\_v\_el(fstream& file, V\_El\* v\_el, int num) {  if (num != 0) {  file << "-> ";  cout << "-> ";  }  file << num << ". ";  cout << num << ". ";  G\_El\* first\_g = v\_el->getForm()->getHead();  print\_g\_el(file, first\_g);  G\_El\* curr = first\_g->getNext();  while (curr != NULL) {  file << "->";  cout << "->";  print\_g\_el(file, curr);  curr = curr->getNext();  }  file << "->NULL" << endl;  cout << "->NULL" << endl;  }  void print\_list(fstream& file, FormV\* list) {  V\_El\* first\_v = list->getHead();  if (first\_v == nullptr) {  file << "The list is empty" << endl << endl;  cout << "The list is empty" << endl << endl;  }  else {  int num = 0;  print\_v\_el(file, first\_v, num);  num++;  V\_El\* curr = first\_v->getNext();  while (curr != NULL) {  print\_v\_el(file, curr, num);  curr = curr->getNext();  num++;  }  file << "-> NULL" << endl << endl;  cout << "-> NULL" << endl << endl;  }  }  void out(string filename, All\* all, FormV\* diff, bool result, bool app = false) {  fstream file;  if (app) {  file.open(filename, ios::app);  }  else {  file.open(filename, ios::out);  }  if (!file.is\_open()) {  perror("Error opening file");  exit(1);  }  FormV\* form\_list[3] = {all->list1, all->list2, all->list3};  for (int i = 0; i < 3; i++) {  cout << "LIST " << i + 1 << endl;  file << "LIST " << i + 1 << endl;  FormV\* list = form\_list[i];  print\_list(file, list);  }  if (diff != nullptr) {  cout << "LIST 1 \\ LIST 2" << endl;  file << "LIST 1 \\ LIST 2" << endl;  print\_list(file, diff);  }  cout << "LIST 3 belongs LIST 1 \\ LIST 2: ";  file << "LIST 3 belongs LIST 1 \\ LIST 2: ";  if (result) {  cout << "true" << endl;  file << "true" << endl;  }  else {  cout << "false" << endl;  file << "false" << endl;  }  file.close();  } | #include "All.h"  void All::read(string filename1, string filename2, string filename3) {  list1 = read\_one(filename1);  list2 = read\_one(filename2);  list3 = read\_one(filename3);  }  FormV\* read\_one(string filename) {  fstream file;  file.open(filename, ios::in);  if (!file.is\_open()) {  perror("Error opening file");  exit(1);  }  FormV\* form\_v = new FormV;  V\_El\* first = new V\_El;  string line;  getline(file, line);  FormG\* form\_g = create\_formG((char\*)line.c\_str());  first->setForm(form\_g);  V\_El\* prev;  V\_El\* curr = new V\_El;  curr = first;  while (getline(file, line)) {  prev = curr;  curr = new V\_El;  form\_g = create\_formG((char\*)line.c\_str());  prev->setNext(curr);  curr->setForm(form\_g);  }  form\_v->setHead(first);  form\_v->setCurr(first);  file.close();  return form\_v;  }  FormG\* create\_formG(char\* line) {  FormG\* form = new FormG;  G\_El\* head = create\_G\_El(line, 0);  G\_El\* curr = head;  G\_El\* prev = new G\_El;  unsigned int i = 0;  while (curr->getStr()->getLen() == MAX\_SIZE) {  i += MAX\_SIZE;  prev = curr;  curr = create\_G\_El(line, i);  if (curr->getStr()->getLen() != 0) {  prev->setNext(curr);  }  }  form->setCurr(head);  form->setHead(head);  return form;  }  G\_El\* create\_G\_El(char\* line, unsigned int i) {  G\_El\* el = new G\_El;  Str\* str = create\_str(line, i);  el->setStr(str);  return el;  }  Str\* create\_str(char\* line, unsigned int i) {  Str\* str = new Str;  unsigned int len = 0;  char sym = line[i];  while (sym != '\0' && len < MAX\_SIZE) {  str->setLetter(len, sym);  len++;  i++;  sym = line[i];  }  if (len < MAX\_SIZE) {  str->setLen(len);  }  return str;  } | #include "Str.h"  int Str::getLen() {  return len;  };  void Str::setLen(unsigned int new\_len) {  if (new\_len > MAX\_SIZE || new\_len < 0) {  throw "Invalid len";  }  len = new\_len;  };  char Str::getLetter(int i) {  return letters[i];  };  void Str::setLetter(int i, char new\_letter) {  if (i >= MAX\_SIZE || i < 0) {  throw "Invalid index";  }  letters[i] = new\_letter;  };  bool Str::equals(Str\* str) {  if (getLen() != str->getLen()) return false;  for (int i = 0; i < getLen(); i++) {  if (getLetter(i) != str->getLetter(i)) return false;  }  return true;  } | #include "All.h"  Str\* copy\_str(Str\* str) {  Str\* str\_copy = new Str;  for (int i = 0; i < str->getLen(); i++) {  str\_copy->setLetter(i, str->getLetter(i));  }  str\_copy->setLen(str->getLen());  return str\_copy;  }  FormG\* copy\_form\_g(FormG\* form\_g) {  G\_El\* g\_el\_copy\_prev = nullptr;  FormG\* form\_g\_copy = new FormG;  while (form\_g->getCurr() != nullptr) {  G\_El\* g\_el = form\_g->getCurr();  G\_El\* g\_el\_copy = new G\_El;  Str\* str = g\_el->getStr();  Str\* str\_copy = copy\_str(str);  g\_el\_copy->setStr(str\_copy);  if (g\_el\_copy\_prev == nullptr) {  form\_g\_copy->setHead(g\_el\_copy);  }  else {  g\_el\_copy\_prev->setNext(g\_el\_copy);  }  g\_el\_copy\_prev = g\_el\_copy;  form\_g->setPrev(form\_g->getCurr());  form\_g->setCurr(form\_g->getCurr()->getNext());  }  form\_g->setCurr(form\_g->getHead());  form\_g->setPrev(nullptr);  form\_g\_copy->setCurr(form\_g\_copy->getHead());  form\_g\_copy->setPrev(nullptr);  return form\_g\_copy;  }  FormV\* copy(FormV\* form\_v) {  FormV\* form\_v\_copy = new FormV;  V\_El\* v\_el\_copy\_prev = nullptr;  while (form\_v->getCurr() != nullptr) {  V\_El\* v\_el = form\_v->getCurr();  V\_El\* v\_el\_copy = new V\_El;  FormG\* form\_g = v\_el->getForm();  FormG\* form\_g\_copy = copy\_form\_g(form\_g);  v\_el\_copy->setForm(form\_g\_copy);  if (v\_el\_copy\_prev == nullptr) {  form\_v\_copy->setHead(v\_el\_copy);  }  else {  v\_el\_copy\_prev->setNext(v\_el\_copy);  }  v\_el\_copy\_prev = v\_el\_copy;  form\_v->setPrev(form\_v->getCurr());  form\_v->setCurr(form\_v->getCurr()->getNext());  }  form\_v->setCurr(form\_v->getHead());  form\_v->setPrev(nullptr);  form\_v\_copy->setCurr(form\_v\_copy->getHead());  form\_v\_copy->setPrev(nullptr);  return form\_v\_copy;  }  bool el\_is\_in(G\_El\* target\_el, FormV\* form\_v) {  while (form\_v->getCurr() != nullptr) {  V\_El\* curr\_v\_el = form\_v->getCurr();  FormG\* form\_g = curr\_v\_el->getForm();  while (form\_g->getCurr() != nullptr) {  if (target\_el->equals(form\_g->getCurr())) {  form\_g->setCurr(form\_g->getHead());  form\_v->setCurr(form\_v->getHead());  return true;  }  form\_g->setCurr(form\_g->getCurr()->getNext());  }  form\_g->setCurr(form\_g->getHead());  form\_v->setCurr(form\_v->getCurr()->getNext());  }  form\_v->setCurr(form\_v->getHead());  return false;  }  void handle\_line(V\_El\* curr\_v\_el, FormV\* form\_v\_2) {  FormG\* form\_g\_1 = curr\_v\_el->getForm();  while (form\_g\_1->getCurr() != nullptr) {  if (el\_is\_in(form\_g\_1->getCurr(), form\_v\_2)) {  if (form\_g\_1->getCurr() == form\_g\_1->getHead()) {  form\_g\_1->setHead(form\_g\_1->getCurr()->getNext());  delete form\_g\_1->getCurr();  form\_g\_1->setCurr(form\_g\_1->getHead());  form\_g\_1->setPrev(nullptr);  }  else {  form\_g\_1->getPrev()->setNext(form\_g\_1->getCurr()->getNext());  delete form\_g\_1->getCurr();  form\_g\_1->setCurr(form\_g\_1->getPrev()->getNext());  }  }  else {  form\_g\_1->setPrev(form\_g\_1->getCurr());  form\_g\_1->setCurr(form\_g\_1->getCurr()->getNext());  }  }  form\_g\_1->setCurr(form\_g\_1->getHead());  form\_g\_1->setPrev(nullptr);  }  void handle\_text(FormV\* form\_v\_1, FormV\* form\_v\_2) {  while (form\_v\_1->getCurr() != nullptr) {  handle\_line(form\_v\_1->getCurr(), form\_v\_2);  if (form\_v\_1->getCurr()->getForm()->getHead() == nullptr) {  if (form\_v\_1->getCurr() == form\_v\_1->getHead()) {  form\_v\_1->setHead(form\_v\_1->getCurr()->getNext());  delete form\_v\_1->getCurr();  form\_v\_1->setCurr(form\_v\_1->getHead());  form\_v\_1->setPrev(nullptr);  }  else {  form\_v\_1->getPrev()->setNext(form\_v\_1->getCurr()->getNext());  delete form\_v\_1->getCurr();  form\_v\_1->setCurr(form\_v\_1->getPrev()->getNext());  }  }  else {  form\_v\_1->setPrev(form\_v\_1->getCurr());  form\_v\_1->setCurr(form\_v\_1->getCurr()->getNext());  }  }  form\_v\_1->setCurr(form\_v\_1->getHead());  form\_v\_1->setPrev(nullptr);  }  FormV\* subtract(FormV\* divided, FormV\* dividor) {  FormV\* result = copy(divided);  handle\_text(result, dividor);  return result;  }    bool belongs(FormV\* small\_set, FormV\* large\_set) {  FormV\* diff = subtract(small\_set, large\_set);  if (diff->getHead() == nullptr) {  return true;  }  return false;  }  void desintegrate(FormV\* form\_v) {  while (form\_v->getCurr() != nullptr) {  V\_El\* v\_el = form\_v->getCurr();  FormG\* form\_g = v\_el->getForm();  while (form\_g->getCurr() != nullptr) {  delete form\_g->getCurr()->getStr();  form\_g->setCurr(form\_g->getCurr()->getNext());  delete form\_g->getPrev();  }  form\_v->setCurr(form\_v->getCurr()->getNext());  delete form\_v->getPrev();  }  delete form\_v;  } |

# Результат работы программы:

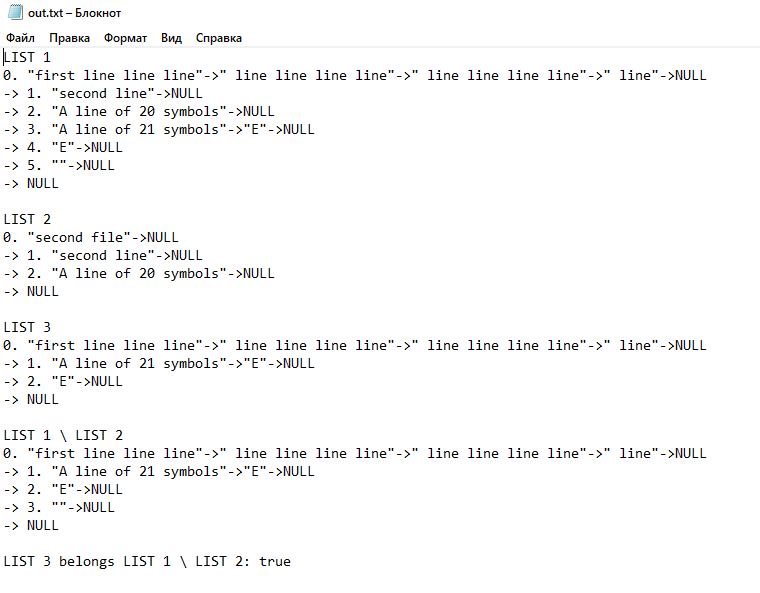
Ввод из файла:



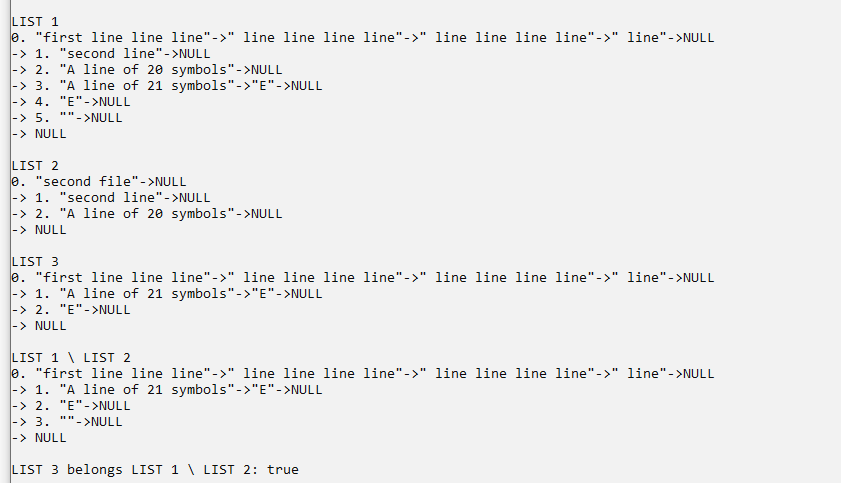




Вывод в файл:



Вывод в консоль:



# Вывод:

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