**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе № 3**

**по дисциплине «Программирование»**

Тема: Обработка структурированных в однонаправленный список данных

|  |  |  |
| --- | --- | --- |
| Студент гр. 2309 |  | Савин П.А. |
| Преподаватель |  | Калмычков В.А. |

Санкт-Петербург

2023

**Формулировка задания.**

Дан текст — последовательность строк, не всегда имеющая смысл. Необходимо сформировать из строк однонаправленный список и выполнить следующее действие: дан второй исходный текст, из него нужно сформировать второй список и вставить справа после заданного элемента первого списка.

**Анализ задания.**

Тексты могут быть и пустыми, но даже так программа отработает корректно (подробнее в «Особенностях реализации задания на компьютере»). В остальном никаких нюансов из формулировки больше не вылезает.

**Математическая** **постановка.**

На этот раз придется отступиться от чисто математической постановки и прибегнуть еще и к информатике.

Дано:

Текст 1 — исходный текст, текст 2 — вставочный текст, число — индекс вставки текста 2. Текст представляет собой последовательность строк.

Выполнить:

Вставку текста 2 в текст 1 справа от элемента по индексу вставки.

Решение:

Обратимся к ранее описанной структуре, которая и будет отвечать за хранение и обработку данных — линейный односвязный список. Это структура, организующая хранение односторонней цепи элементов (так называемых узлов), каждый из которых состоит из информационного содержимого и адреса области памяти, которая выделена под условно следующий узел. Выглядит все это так (рис. 1):

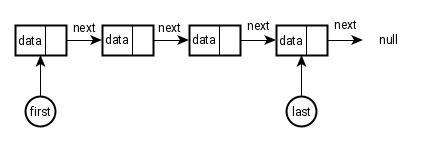


рис. 1 — общая схема организации линейного однонаправленного списка

При этом важно понимать, что используемые переменные локальны, а области памяти, выделенные под элементы, глобальны (изменение адреса в переменной локально, изменение адреса в элементе глобально).

Рассмотрим следующий пример:

Hello!

Goodbye!

И

0

Hi!

Изначально тексты находятся в линейном представлении (рис. 2 и 3):



рис. 2 — текст 1 в линейном представлении

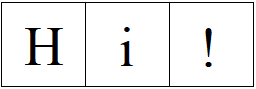


рис. 3 — текст 2 в линейном представлении

Создаем для каждого текста отдельный список, и заносим строки как элементы. Список при инициализации не имеет в доступе ничего — ни единой области памяти, поэтому создаем новые элементы и заносим их в список. Алгоритм образования элементов на примере выглядит так (рис. 4 и 5):

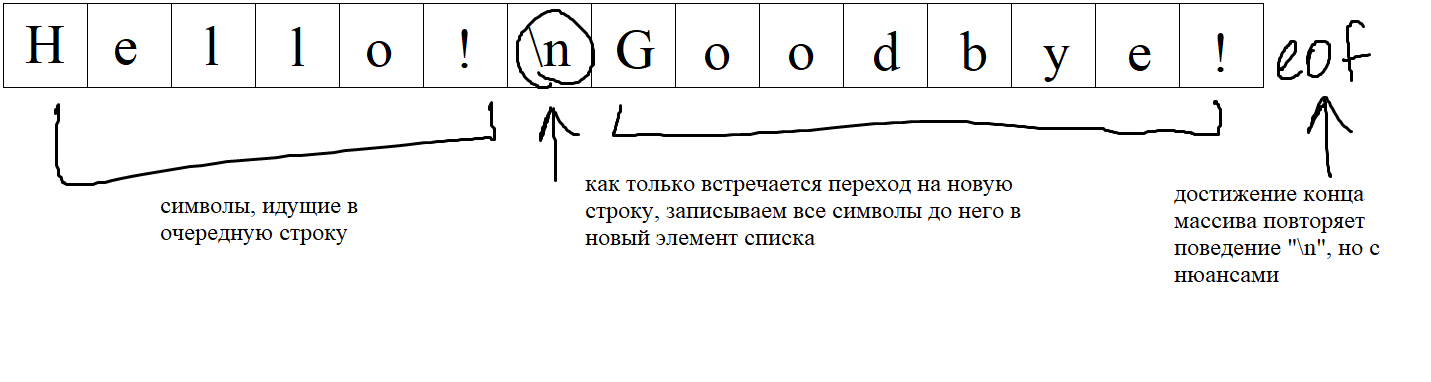


рис. 4 — алгоритм формирования элементов на тексте 1

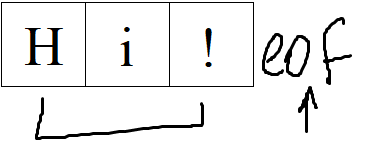


рис. 5 — алгоритм формирования элементов на тексте 2

Получаются следующие элементы (рис. 6):

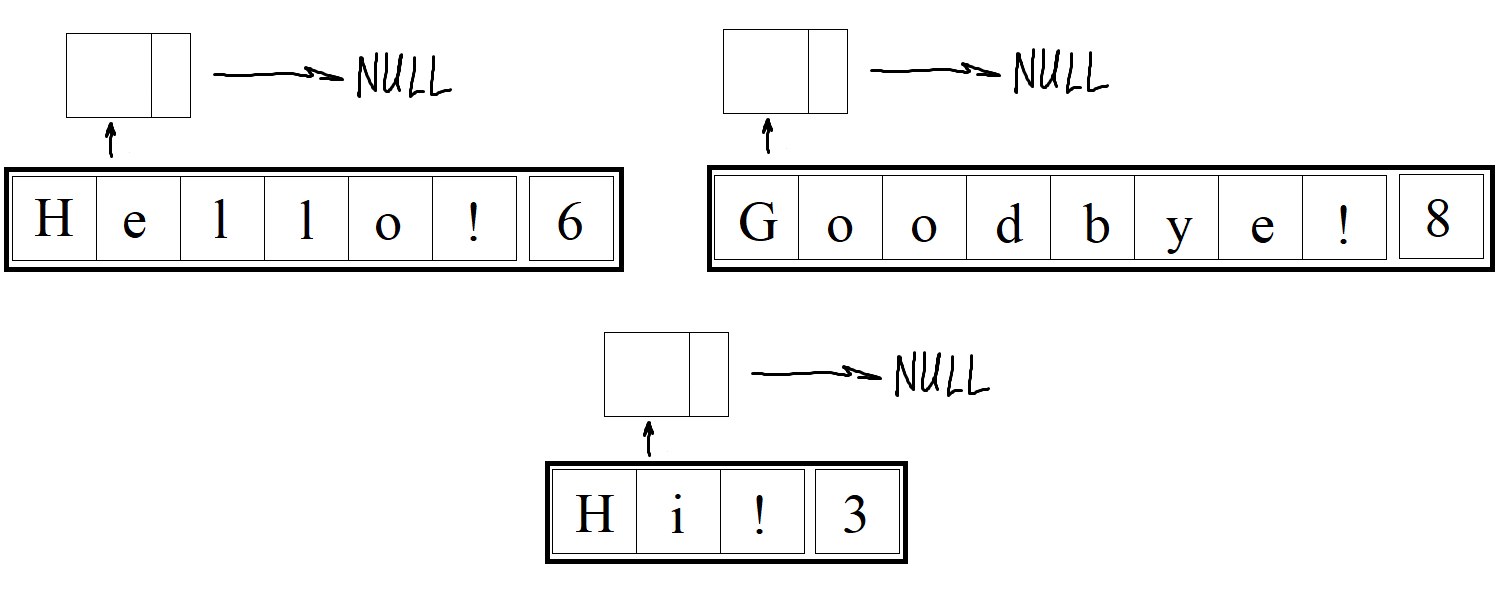


рис. 6 — полученные элементы

Они находятся в вакууме, то есть, не имеют никаких связей, они просто существуют в памяти, именно поэтому важно заносить их в список по мере формирования очередного элемента. Алгоритм последовательного формирования списков выглядит так (рис. 7):

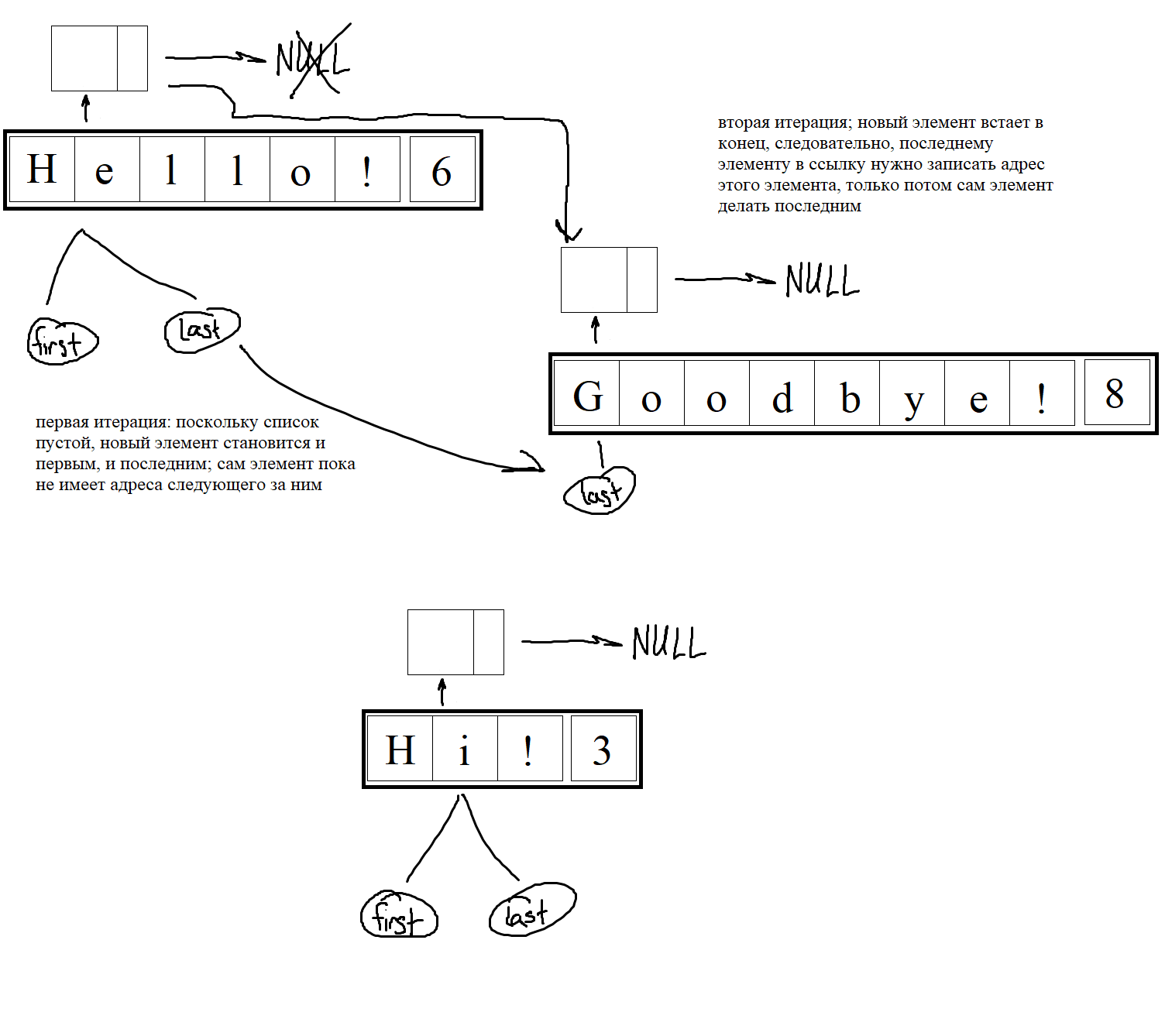


рис. 7 — алгоритм формирования списков

Теперь осталось только вставить содержимое второго списка в первый. Вставка итеративная, со сдвигом относительно номера очередного запущенного процесса вставки. Общий алгоритм вставки посередине одного элемента выглядит так (рис. 8):

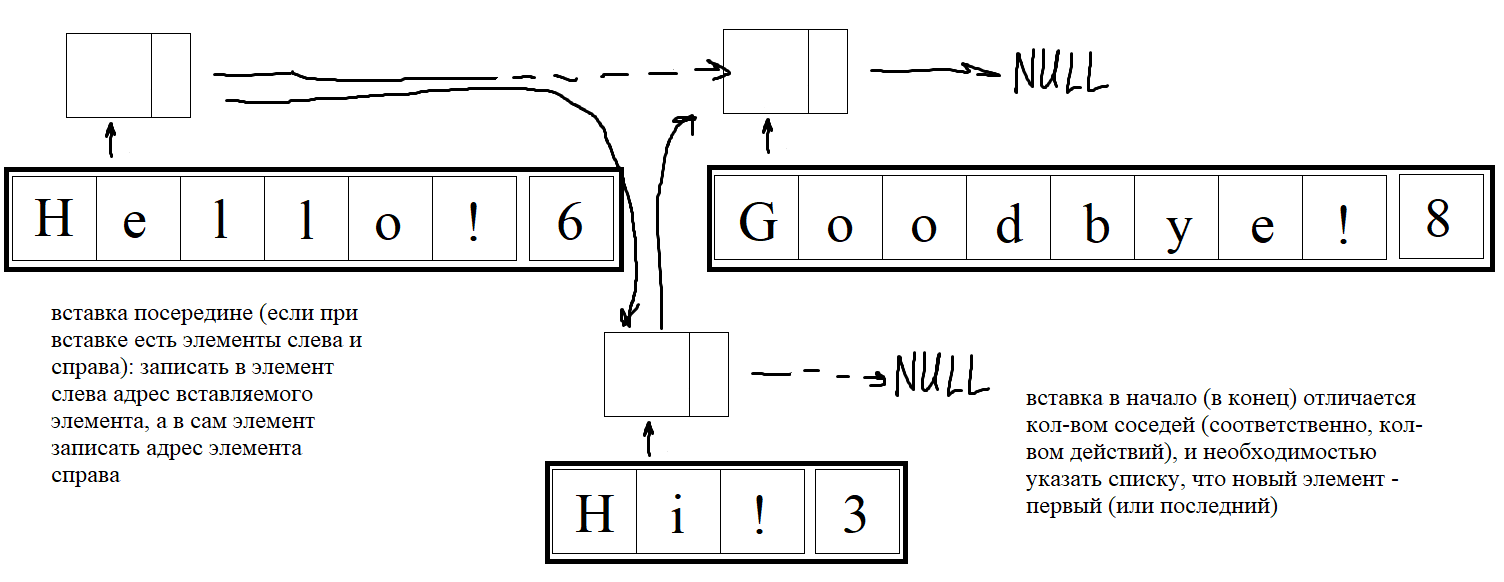


рис. 8 — общий алгоритм вставки нового элемента

Ответ:

Hello!

Hi!

Goodbye!

**Контрольный пример.**

В качестве контрольного примера были взяты следующие тексты:

Hello, world!

Goodbye, world!

и

0

Welcome to zombo.com!

Welcome!

You can do anything at zombo.com!

Anything at all!

Ответом на задачу будет следующий текст:

Hello, world!

Welcome to zombo.com!

Welcome!

You can do anything at zombo.com!

Anything at all!

Goodbye, world!

**Особенности реализации задания на компьютере.**

Дополнительное условие к заданию — исходные данные должны быть удалены после выполнения программы, соответственно, в структуре должна появиться реализация удаления самой структуры. Есть много способов удаления, но для всего списка удаление будем делать с конца, алгоритм следующий (рис. 9):

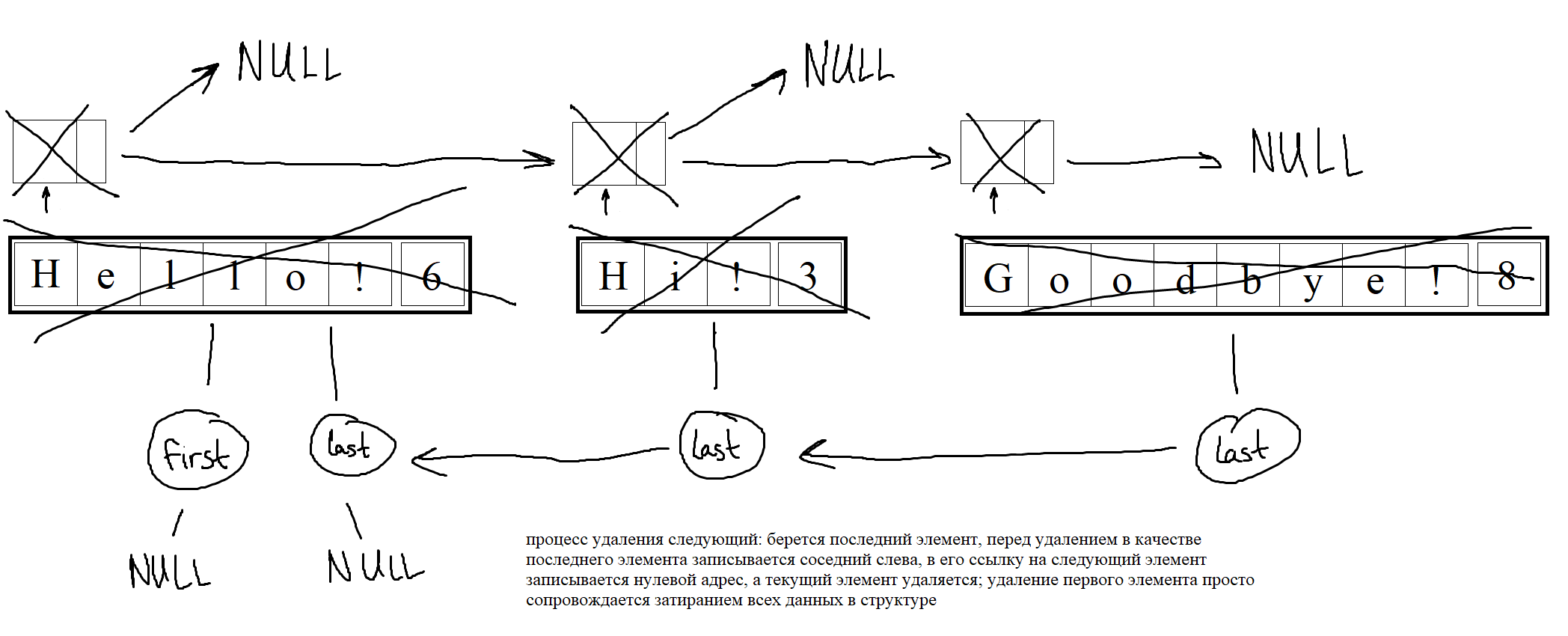


рис. 9 — алгоритм удаления списка

Ранее было упомянуто, что текст может быть и пустым, но даже так программа отработает корректно, потому что пустая строка в случае обработки — тоже строка, правда, в ней будет размещаться нуль-терминатор для опознавания пустой строки.

Модельное представление остается прежним — массив символов с привязанной длиной строки (рис. 10):

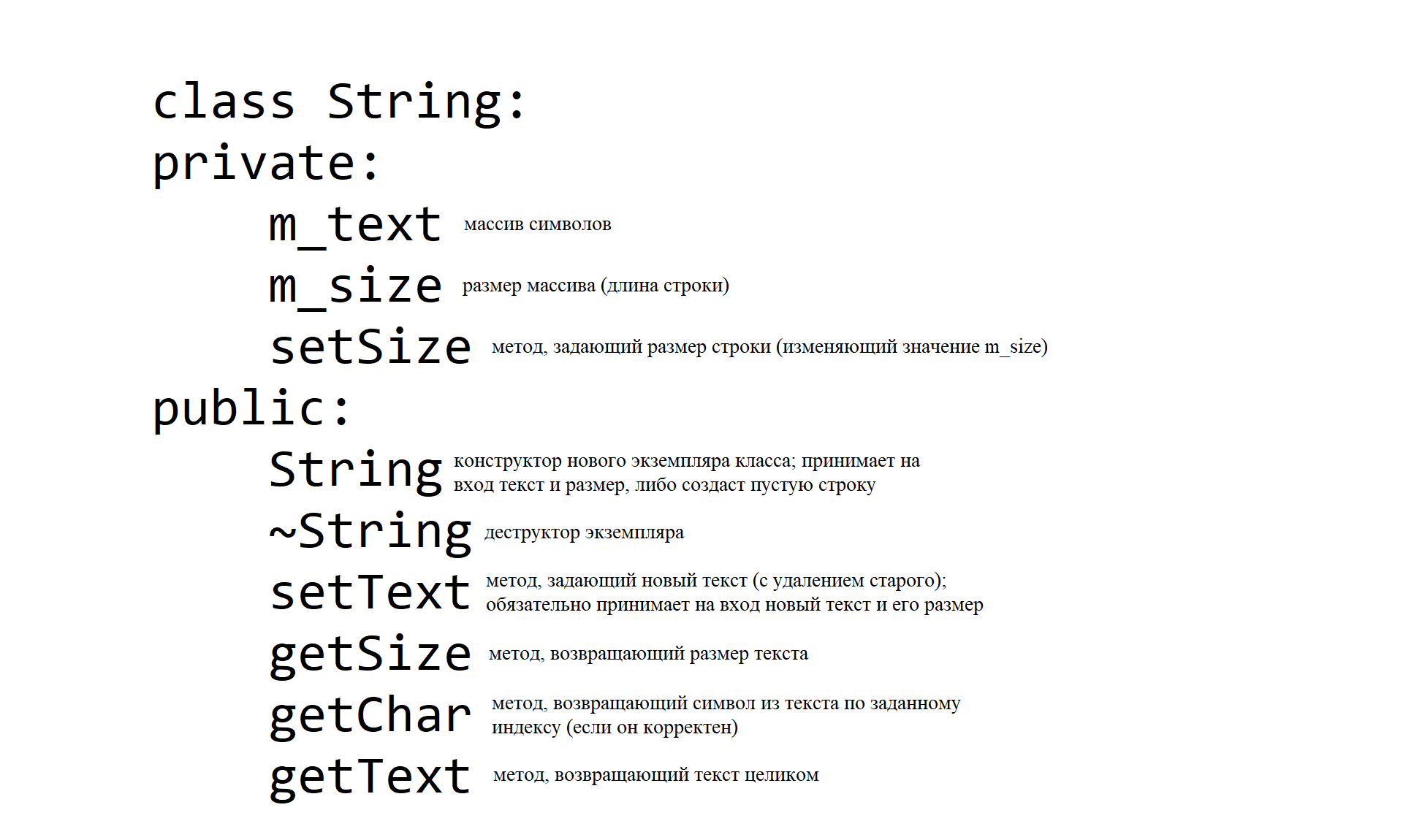


рис. 10 — описание класса String (базовый класс для всего)

Теперь к реализации структуры данных. Для полной реализации списка требуется описать элемент списка (рис. 11) и сам список (рис. 12):

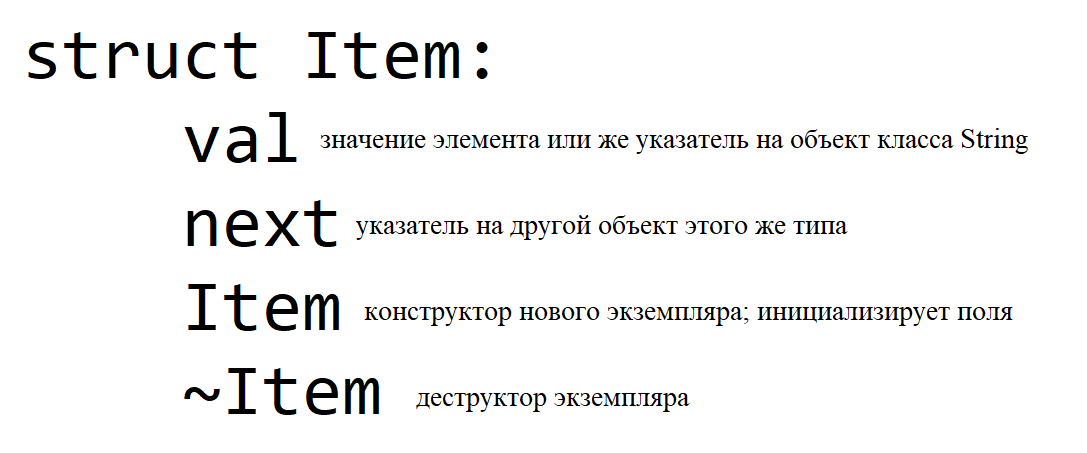


рис. 11 — описание структуры Item (базовый элемент списка)

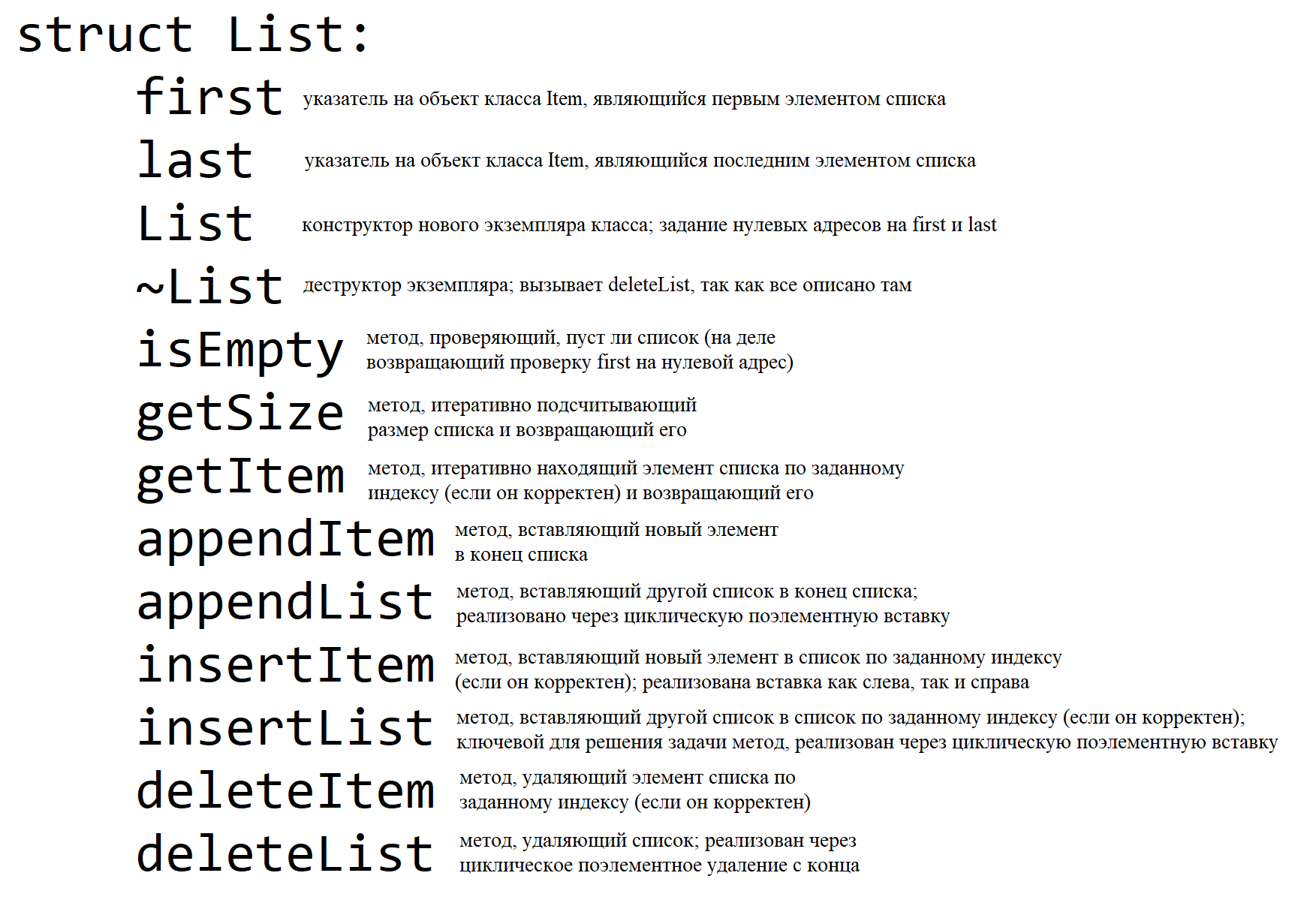


рис. 12 — описание структуры List

Это описание немного излишнее, так как чисто для решения код был бы процентов на 20-30, но мы забегаем вперед, на будущее, на случай, когда понадобится полная реализация списка как структуры данных.

Дерево подключений в этот раз выглядит так (рис. 13):

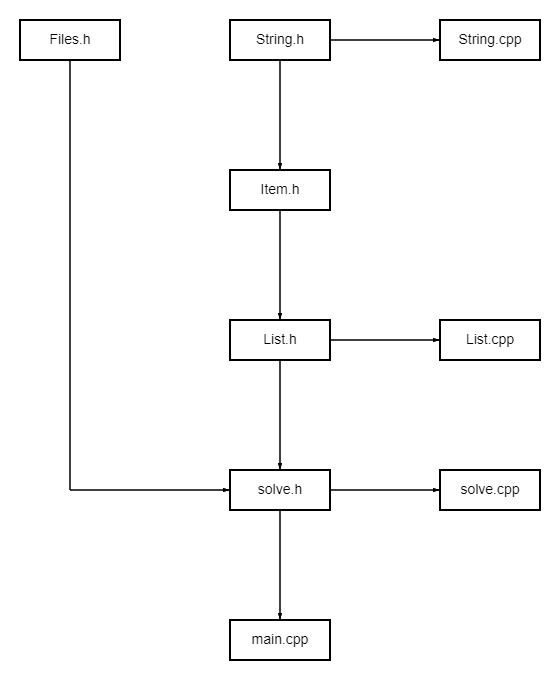


рис. 13 — дерево подключений

**Разработка интерфейса пользователя.**

Входные файлы:

I1: текст

I2: индекс вставки

I3: текст

Выходной файл:

O1: Ошибка: входной файл 1 не найден. (не найден файл с текстом, куда будет производиться вставка)

O2: Ошибка: входной файл 2 не найден. (не найден файл с текстом, который будет вставляться)

O3: Сформированы списки:

Список 1:

O4: текст из I1 следующего формата:

строка->

строка->

NULL

O5: Список 2:

O6: Текст из I3 следующего формата:

строка->

строка->

NULL

O7: Был указан индекс, выходящий за пределы списка 1. Программа все равно отработала вставку, но это действие не имело смысла. (если такое произошло)

O8: Список 1 после вставки списка 2 справа по индексу (индекс из I2):

строка->

строка->

NULL

O9: Проверка на корректное удаление:

До:

O10: адрес списка 1, адрес начала списка 1, адрес конца списка 1

адрес списка 2, адрес начала списка 2, адрес конца списка 2

O11: После:

O12: адрес списка 1, адрес начала списка 1, адрес конца списка 1 (изменены)

адрес списка 2, адрес начала списка 2, адрес конца списка 2 (изменены)

Лог-файл:

O1: Программа начала свою работу.

Производится попытка открыть входные файлы.

O2: Входной файл 1 не найден. (не найден файл с текстом, куда будет производиться вставка)

O3: Входной файл 2 не найден. (не найден файл с текстом, который будет вставляться)

O4: Программа завершила свою работу. (если хотя бы один файл не обнаружен)

O5: Производится запись с двух входных файлов в списки.

O6: Сформированы списки:

Список 1:

O7: текст из I1 следующего формата:

строка->

строка->

NULL

O8: Список 2:

O9: Текст из I3 следующего формата:

строка->

строка->

NULL

O10: Производится попытка вставить список 2 в список 1.

O11: Был указан индекс, выходящий за пределы списка 1. Программа все равно отработала вставку, но это действие не имело смысла. (если такое произошло)

O12: Список 1 после вставки списка 2 справа по индексу (индекс из I2):

строка->

строка->

NULL

O13: Проверка на корректное удаление:

До:

O14: адрес списка 1, адрес начала списка 1, адрес конца списка 1

адрес списка 2, адрес начала списка 2, адрес конца списка 2

O15: После:

O16: адрес списка 1, адрес начала списка 1, адрес конца списка 1 (изменены)

адрес списка 2, адрес начала списка 2, адрес конца списка 2 (изменены)

O17: Программа завершила свою работу.

**Описание используемых данных.**

Структура Files:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Имя | Модификатор | Назначение |
| std::ifstream | sourceFile | public | входной файл 1 |
| insertFile | входной файл 2 |
| std::ofstream | outFile | выходной файл |
| logFile | лог-файл |

Класс String:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Имя | Модификатор | Назначение |
| char\* | m\_text | private | массив символов (текст) |
| int | m\_size | привязанная длина текста |

Структура Item:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Имя | Модификатор | Назначение |
| String\* | val | public | значение элемента |
| Item\* | next | указатель на следующий элемент |

Структура List:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Имя | Модификатор | Назначение |
| Item\* | first | public | указатель на первый элемент в списке |
| last | указатель на последний элемент в списке |

Отдельные переменные:

|  |  |  |
| --- | --- | --- |
| Тип | Имя | Назначение |
| Files\* | files (main) | файловая переменная |
| int | index | хранение индекса вставки |
| charsCount1 | подсчет символов для строк в файле 1 |
| charsCount2 | подсчет символов для строк в файле 2 |
| startPos1 | хранение позиции начала считывания в файле 1 |
| startPos2 | хранение позиции начала считывания в файле 2 |
| char | tmp1 | заглушка для символов в файле 1 |
| tmp2 | заглушка для символов в файле 2 |
| char\* | tmpStr | заглушка для строки |
| Item\* | tmpItem | заглушка для нового элемента списка |
| List\* | l1, l2 | исходные списки |
| bool | eof1, eof2 | отслеживание первого EOF для файлов 1 и 2 |

**Описание используемых функций.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Имя | Файл | Назначение | Параметры | | | |
| входные | выходные | модифицируемые | транзитные |
| String::setSize | String.cpp | задание привязанной длины строки | size (из setText) |  |  |  |
| String::setText | задание нового текста | text |  |  | size (переходит в setSize) |
| String::getSize | возврат длины строки |  |  |  |  |
| String::getChar | возврат символа строки по указанному индексу | index |  |  |  |
| String::getText | возврат текста |  |  |  |  |
| List::isEmpty | List.cpp | проверка на пустоту списка |  |  |  |  |
| List::getSize | возврат размера списка |  |  |  |  |
| List::getItem | возврат элемента списка по указанному индексу | index |  |  |  |
| List::appendItem | вставка нового элемента списка в конец | val |  |  |  |
| List::appendList | вставка другого списка в конец | list |  |  |  |
| List::insertItem | вставка нового элемента в список по указанному индексу | val, index, left |  |  | val (может перейти в appendItem) |
| List::insertList | вставка другого списка в список по указанному индексу |  |  |  | list, index, left (переходят с изменениями в insertItem) |
| List::deleteItem | удаление элемента из списка по заданному индексу | index |  |  |  |
| List::deleteList | удаление списка |  |  |  |  |
| checkFiles | solve.cpp | проверка существования входных файлов |  |  | files |  |
| solve | решение задачи |  |  | files |  |

**Организация ввода-вывода.**

|  |  |  |  |
| --- | --- | --- | --- |
| Библиотека | Имя | Команда | Назначение |
| fstream |  | >> | считывание с файла в переменную |
|  | << | запись в файл |
|  | seekg | перевод курсора на заданную позицию |
|  | tellg | возврат текущей позиции курсора |
|  | open | открытие файла |
|  | close | закрытие файла |
|  | clear | снятие флагов |
|  | is\_open | проверка на открытый файл |
|  | eof | проверка на EOF |
|  | unsetf | снятие флага считывания |

**Представление алгоритма решения задачи.**

1. Проверить существование обоих входных файлов

2. Считать одновременно с двух файлов тексты и записать их в списки (алгоритм на рис. 4 и 5)

3. Вывести списки по шаблонам O3-O6 и O4-O9

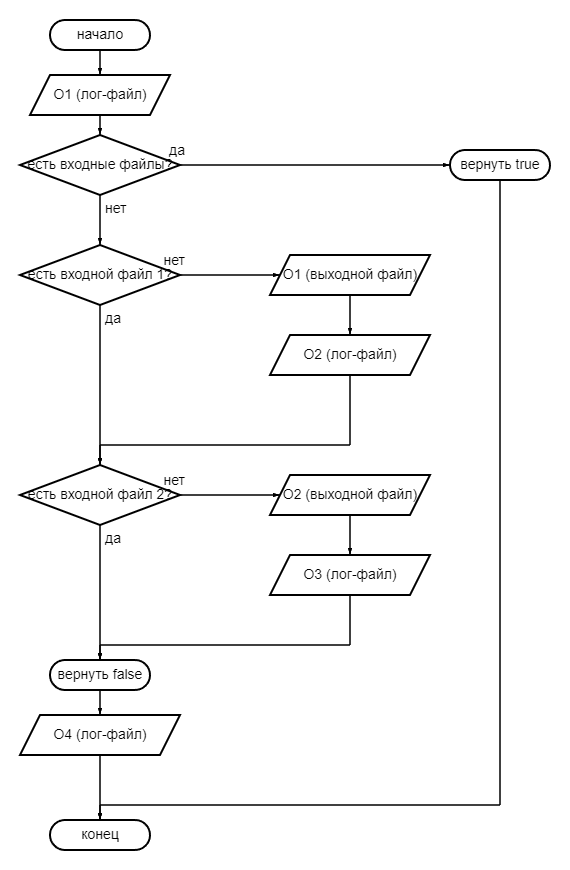
4. Произвести вставку списка 2 в список 1 по индексу из шаблона I2

5. Вывести новый список

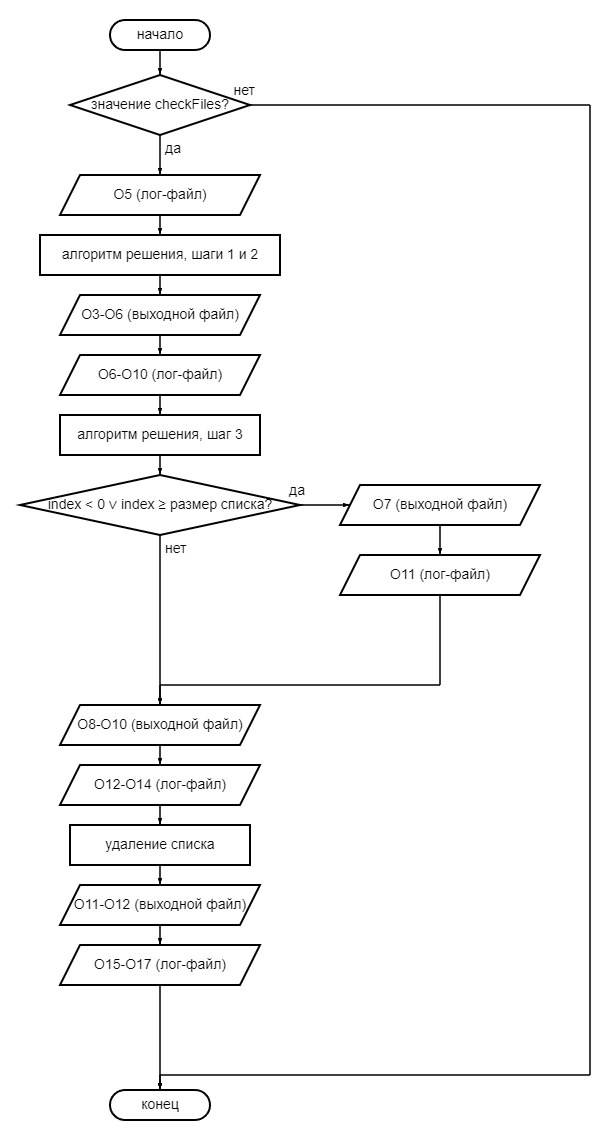
6. Произвести удаление списков и продемонстрировать факт удаления

**Блок-схема.**

main и checkFiles:



solve:



**Текст программы.**

main.cpp:

/\*

Программа по считыванию данных из файла в односвязный список

Автор: Савин Павел Алексеевич, группа 2309. Версия 3.1.1

Даты: Начало: 20.03.2023 Окончание: 28.03.2023

\*/

#include "solve.h"

int main() {

Files\* files = new Files;

solve(files);

delete files;

return 0;

}

Files.h (Files):

#pragma once

#include <fstream>

struct Files {

std::ifstream sourceFile;

std::ifstream insertFile;

std::ofstream outFile;

std::ofstream logFile;

Files(const char\* source = "source.txt", const char\* insert = "insert.txt", const char\* out = "out.txt", const char\* log = "log.txt") {

this->sourceFile.open(source);

this->insertFile.open(insert);

this->outFile.open(out);

this->logFile.open(log);

}

~Files() {

this->sourceFile.close();

this->insertFile.close();

this->outFile.close();

this->logFile.close();

}

};

String.h (String):

#pragma once

class String {

private:

char\* m\_text;

int m\_size;

void setSize(const int size);

public:

String(const char\* text = " ", const int size = 1) {

this->m\_text = new char[size];

for (int i = 0; i < size; ++i) this->m\_text[i] = text[i];

this->m\_size = size;

}

~String() {

delete[] this->m\_text;

}

void setText(const char\* text, const int size);

int getSize();

char getChar(const int index);

char\* getText();

};

String.cpp (String):

#include "String.h"

void String::setSize(const int size) {

this->m\_size = size;

}

void String::setText(const char\* text, const int size) {

if (this->m\_text != nullptr) delete[] this->m\_text;

this->m\_text = new char[size];

for (int i = 0; i < size; ++i) this->m\_text[i] = text[i];

this->setSize(size);

}

int String::getSize() {

return this->m\_size;

}

char String::getChar(const int index) {

if (index < 0 || index >= this->getSize()) return 0;

return this->m\_text[index];

}

char\* String::getText() {

return this->m\_text;

}

Item.h (Item):

#pragma once

#include "String.h"

struct Item {

String\* val;

Item\* next;

Item() {

this->val = new String;

this->next = nullptr;

}

~Item() {

delete this->val;

this->next = nullptr;

}

};

List.h (List):

#pragma once

#include "Item.h"

struct List {

Item\* first;

Item\* last;

List() {

this->first = nullptr;

this->last = nullptr;

}

~List() {

this->deleteList();

}

bool isEmpty();

int getSize();

Item\* getItem(const int index);

void appendItem(Item\* val);

void appendList(List\* list);

void insertItem(Item\* val, const int index, bool left);

void insertList(List\* list, const int index, bool left);

void deleteItem(const int index);

void deleteList();

};

List.cpp (List):

#include "List.h"

bool List::isEmpty() {

return this->first == nullptr;

}

int List::getSize() {

int length = 0;

Item\* p = this->first;

while (p != nullptr) {

++length;

p = p->next;

}

return length;

}

Item\* List::getItem(const int index) {

int size = this->getSize();

if (index < 0 || index >= size) return nullptr;

int pos = 0;

Item\* p = this->first;

while (pos != index) {

++pos;

p = p->next;

}

return p;

}

void List::appendItem(Item\* val) {

Item\* tmp = new Item;

tmp->val->setText(val->val->getText(), val->val->getSize());

if (this->isEmpty()) {

this->first = tmp;

this->last = tmp;

}

else {

this->last->next = tmp;

this->last = tmp;

}

}

void List::appendList(List\* list) {

for (int i = 0; i < list->getSize(); ++i) this->appendItem(this->getItem(i));

}

void List::insertItem(Item\* val, const int index, bool left = true) {

int size = this->getSize();

if (index < 0 || (size && index >= size)) return;

if (index + left == size - 1 || (index == 0 && size == 0)) this->appendItem(val);

else if (index + (!left) == 0) {

Item\* tmp = new Item;

tmp->val->setText(val->val->getText(), val->val->getSize());

tmp->next = this->first;

this->first = tmp;

tmp = nullptr;

}

else {

Item\* tmp = new Item;

tmp->val->setText(val->val->getText(), val->val->getSize());

Item\* prev = this->getItem(index - left);

Item\* next = this->getItem(index + (!left));

prev->next = tmp;

tmp->next = next;

tmp = nullptr;

prev = nullptr;

next = nullptr;

}

}

void List::insertList(List\* list, const int index, bool left = true) {

for (int i = 0; i < list->getSize(); ++i) this->insertItem(list->getItem(i), index + i, left);

}

void List::deleteItem(const int index) {

if (this->isEmpty()) return;

int size = this->getSize();

if (index < 0 || index >= size) return;

if (size == 1) {

this->last = nullptr;

delete this->first;

this->first = nullptr;

}

else {

Item\* del = this->getItem(index);

if (index == 0) {

this->first = del->next;

delete del;

}

else if (index == size - 1) {

this->last = this->getItem(index - 1);

this->last->next = nullptr;

delete del;

}

else {

Item\* prev = this->getItem(index - 1);

prev->next = this->getItem(index + 1);

delete del;

prev = nullptr;

}

del = nullptr;

if (this->getSize() == 1) {

this->last = this->first;

}

}

}

void List::deleteList() {

int curIndex = this->getSize() - 1;

while (this->first != nullptr) {

this->deleteItem(curIndex);

--curIndex;

}

}

solve.h:

#pragma once

#include "Files.h"

#include "List.h"

bool checkFiles(Files\* files);

void solve(Files\* files);

solve.cpp:

#include "solve.h"

bool checkFiles(Files\* files) {

files->logFile << "Программа начала свою работу.\nПроизводится попытка открыть входные файлы.\n";

if (files->sourceFile.is\_open() && files->insertFile.is\_open()) return true;

if (!files->sourceFile.is\_open()) {

files->logFile << "Входной файл 1 не найден.\n";

files->outFile << "Ошибка: входной файл 1 не найден.\n";

}

if (!files->insertFile.is\_open()) {

files->logFile << "Входной файл 2 не найден.\n";

files->outFile << "Ошибка: входной файл 2 не найден.";

}

files->logFile << "Программа завершила свою работу.";

return false;

}

void solve(Files\* files) {

if (!checkFiles(files)) return;

files->logFile << "Производится запись с двух входных файлов в списки.\n";

int index;

files->insertFile >> index;

files->sourceFile.unsetf(std::ios::skipws);

files->insertFile.unsetf(std::ios::skipws);

char tmp1, tmp2;

files->insertFile >> tmp2;

int charsCount1 = 0, charsCount2 = 0, startPos1 = files->sourceFile.tellg(), startPos2 = files->insertFile.tellg();

bool eof1 = false, eof2 = false;

Item\* tmpItem;

char\* tmpStr;

List\* l1 = new List;

List\* l2 = new List;

do {

if (eof1 && eof2) break;

files->sourceFile >> tmp1;

files->insertFile >> tmp2;

if (files->sourceFile.eof() && !eof1) {

files->sourceFile.clear();

files->sourceFile.seekg(startPos1, std::ios::beg);

tmpItem = new Item;

tmpStr = new char[(charsCount1 ? charsCount1 : 1)];

tmpStr[0] = '\0';

for (int i = 0; i < charsCount1; ++i) files->sourceFile >> tmpStr[i];

files->sourceFile >> tmp1;

tmpItem->val->setText(tmpStr, charsCount1);

l1->appendItem(tmpItem);

tmpItem = nullptr;

delete[] tmpStr;

tmpStr = nullptr;

charsCount1 = 0;

eof1 = true;

}

else if (tmp1 == '\n' && !eof1) {

files->sourceFile.seekg(startPos1, std::ios::beg);

tmpItem = new Item;

tmpStr = new char[(charsCount1 ? charsCount1 : 1)];

tmpStr[0] = '\0';

for (int i = 0; i < charsCount1; ++i) files->sourceFile >> tmpStr[i];

files->sourceFile >> tmp1;

startPos1 = files->sourceFile.tellg();

tmpItem->val->setText(tmpStr, charsCount1);

l1->appendItem(tmpItem);

tmpItem = nullptr;

delete[] tmpStr;

tmpStr = nullptr;

charsCount1 = 0;

}

else if (!eof1) ++charsCount1;

if (files->insertFile.eof() && !eof2) {

files->insertFile.clear();

files->insertFile.seekg(startPos2, std::ios::beg);

tmpItem = new Item;

tmpStr = new char[(charsCount2 ? charsCount2 : 1)];

tmpStr[0] = '\0';

for (int i = 0; i < charsCount2; ++i) files->insertFile >> tmpStr[i];

files->insertFile >> tmp2;

tmpItem->val->setText(tmpStr, charsCount2);

l2->appendItem(tmpItem);

tmpItem = nullptr;

delete[] tmpStr;

tmpStr = nullptr;

charsCount2 = 0;

eof2 = true;

}

else if (tmp2 == '\n' && !eof2) {

files->insertFile.seekg(startPos2, std::ios::beg);

tmpItem = new Item;

tmpStr = new char[(charsCount2 ? charsCount2 : 1)];

tmpStr[0] = '\0';

for (int i = 0; i < charsCount2; ++i) files->insertFile >> tmpStr[i];

files->insertFile >> tmp2;

startPos2 = files->insertFile.tellg();

tmpItem->val->setText(tmpStr, charsCount2);

l2->appendItem(tmpItem);

tmpItem = nullptr;

delete[] tmpStr;

tmpStr = nullptr;

charsCount2 = 0;

}

else if (!eof2) ++charsCount2;

}

while (true);

files->logFile << "Сформированы списки:\n";

files->logFile << "Список 1:\n";

files->outFile << "Сформированы списки:\n";

files->outFile << "Список 1:\n";

tmpItem = l1->getItem(0);

while (tmpItem != nullptr) {

if (tmpItem->val->getSize() == 1 && tmpItem->val->getChar(0) == '\0');

else {

for (int i = 0; i < tmpItem->val->getSize(); ++i) {

files->logFile << tmpItem->val->getChar(i);

files->outFile << tmpItem->val->getChar(i);

}

}

files->logFile << "->\n";

files->outFile << "->\n";

tmpItem = tmpItem->next;

}

files->logFile << "NULL\n";

files->logFile << "Список 2:\n";

files->outFile << "NULL\n";

files->outFile << "Список 2:\n";

tmpItem = l2->getItem(0);

while (tmpItem != nullptr) {

if (tmpItem->val->getSize() == 1 && tmpItem->val->getChar(0) == '\0');

else {

for (int i = 0; i < tmpItem->val->getSize(); ++i) {

files->logFile << tmpItem->val->getChar(i);

files->outFile << tmpItem->val->getChar(i);

}

}

files->logFile << "->\n";

files->outFile << "->\n";

tmpItem = tmpItem->next;

}

files->logFile << "NULL\n";

files->outFile << "NULL\n";

files->logFile << "Производится попытка вставить список 2 в список 1.\n";

l1->insertList(l2, index, false);

if (index < 0 || index >= l1->getSize()) {

files->logFile << "Был указан индекс, выходящий за пределы списка 1. Программа все равно отработала вставку, но это действие не имело смысла.\n";

files->outFile << "Был указан индекс, выходящий за пределы списка 1. Программа все равно отработала вставку, но это действие не имело смысла.\n";

}

files->logFile << "Список 1 после вставки списка 2 справа по индексу " << index << ":\n";

files->outFile << "Список 1 после вставки списка 2 справа по индексу " << index << ":\n";

tmpItem = l1->getItem(0);

while (tmpItem != nullptr) {

if (tmpItem->val->getSize() == 1 && tmpItem->val->getChar(0) == '\0');

else {

for (int i = 0; i < tmpItem->val->getSize(); ++i) {

files->logFile << tmpItem->val->getChar(i);

files->outFile << tmpItem->val->getChar(i);

}

}

files->logFile << "->\n";

files->outFile << "->\n";

tmpItem = tmpItem->next;

}

files->logFile << "NULL\n";

files->logFile << "Проверка на корректное удаление:\nДо:\n";

files->logFile << l1 << ' ' << l1->first << ' ' << l1->last << '\n';

files->logFile << l2 << ' ' << l2->first << ' ' << l2->last << '\n';

files->outFile << "NULL\n";

files->outFile << "Проверка на корректное удаление:\nДо:\n";

files->outFile << l1 << ' ' << l1->first << ' ' << l1->last << '\n';

files->outFile << l2 << ' ' << l2->first << ' ' << l2->last << '\n';

delete l1;

delete l2;

files->logFile << "После:\n";

files->logFile << l1 << ' ' << l1->first << ' ' << l1->last << '\n';

files->logFile << l2 << ' ' << l2->first << ' ' << l2->last << '\n';

files->outFile << "После:\n";

files->outFile << l1 << ' ' << l1->first << ' ' << l1->last << '\n';

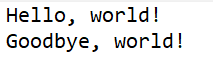
files->outFile << l2 << ' ' << l2->first << ' ' << l2->last << '\n';

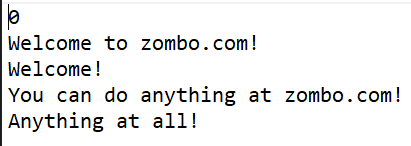
files->logFile << "Программа завершила свою работу.";

}

**Результаты работы программы.**

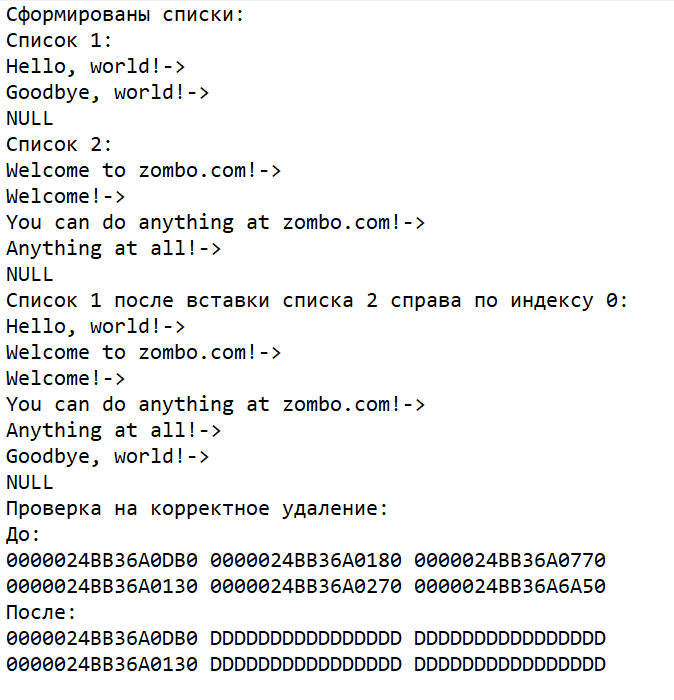
На примере следующих входных файлов:



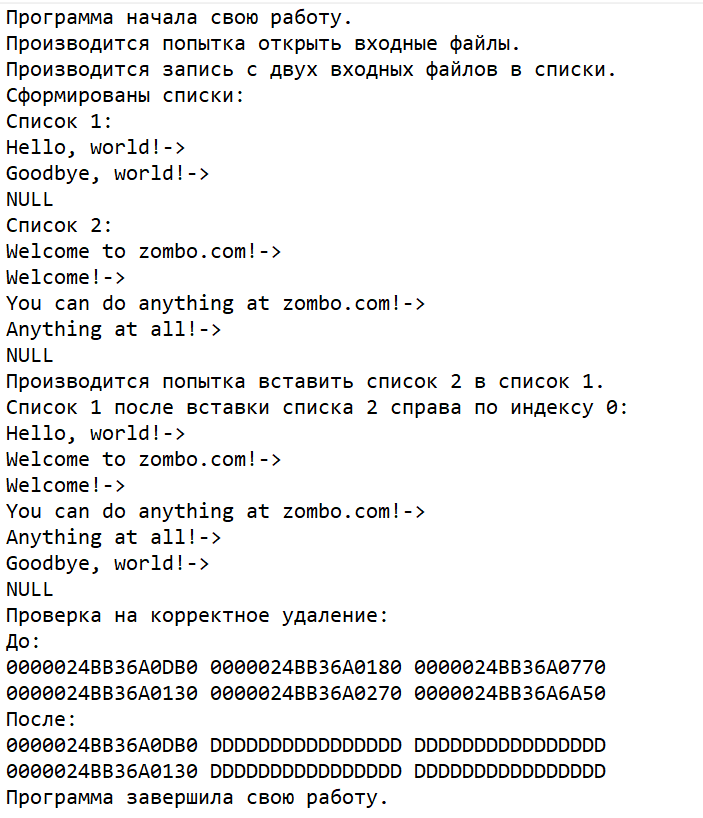


получены следующие результаты:

Выходной файл:



Лог-файл:



**Вывод о проделанной работе.**

Изучены линейные односвязные списки и взаимодействие с ними.