**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе № 4**

**по дисциплине «Программирование»**

Тема: Обработка данных с использованием иерархических списков

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

Санкт-Петербург

2023

**Формулировка задания.**

Дан текст — последовательность строк, не всегда имеющая смысл. Необходимо разбить строки на блоки фиксированного размера, составить из них однонаправленные списки, и связать полученные списки одним большим однонаправленным списком (полученная структура и является иерархическим списком). После чего придумать как минимум два действия, обрабатывающие элементы большого списка, начиная с заданного по индексу, и реализовать их.

**Анализ задания.**

Задание звучит именно так: придумать. Положим, на реализацию пойдут следующие функции: переворот списка и попарная перестановка элементов (если число элементов четное). Речь идет о поблочных функциях, поскольку список состоит из блоков строк, поэтому тот же переворот будет отличаться от стандартной реализации.

Также стоит пояснить, что поблочное разделение строк подразумевает заранее выделяемую память под фиксировано заданного размера блока и считывания в него каждого символа по отдельности до заполнения.

**Математическая** **постановка.**

Дано:

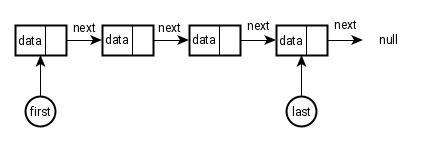
Два числа: первое — отметка, какое действие нужно совершить; второе — индекс начала действия. Текст — последовательность строк. Отдельно предоставляется еще одна константа — размер блоков.

Выполнить:

Разбить строки на блоки фиксированного размера, организовать из блоков списки, после чего из самих списков организовать один большой список, после чего совершить над ним выбранное действие.

Решение:

В ход, помимо самого списка, идет вспомогательная структура — формуляр. Формуляр — это именно то, что большинство называют списком, однако стоит помнить, что список — это искусственный способ организации данных в памяти, его можно создавать вручную, так и через дополнительный интерфейс. Вот таким устоявшимся интерфейсом и является формуляр. В ранее обрисованной структуре



формуляр — это та штука, что оперирует нарисованными элементами.

Так вот, начальные действия те же. Разберем на следующем тексте:

Hello, world!

Goodbye!

(фиксированная длина — 5 символов)

Формируем из строки элементы, они же блоки (рис. 1)

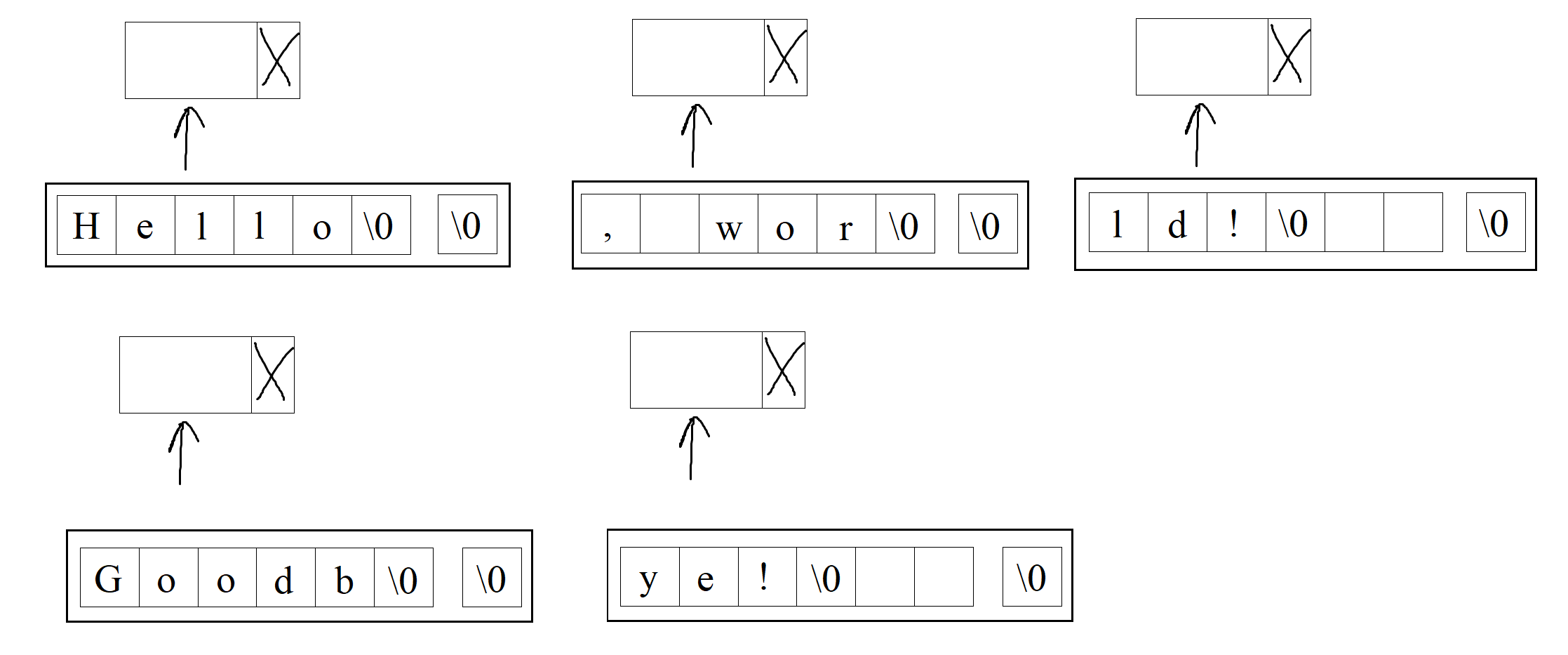


рис. 1 — полученные блоки

и соединяем их в один список. Как только достигается конец строки (символ перехода на новую строку), весь список добавляется в конец большого списка. Как именно? Через формуляр. Делаем формуляр элементом нового, большого списка, и образуем один большой, иерархический список, где содержательная часть каждого элемента будет давать прямой контроль над полученными выше списками (рис. 2).

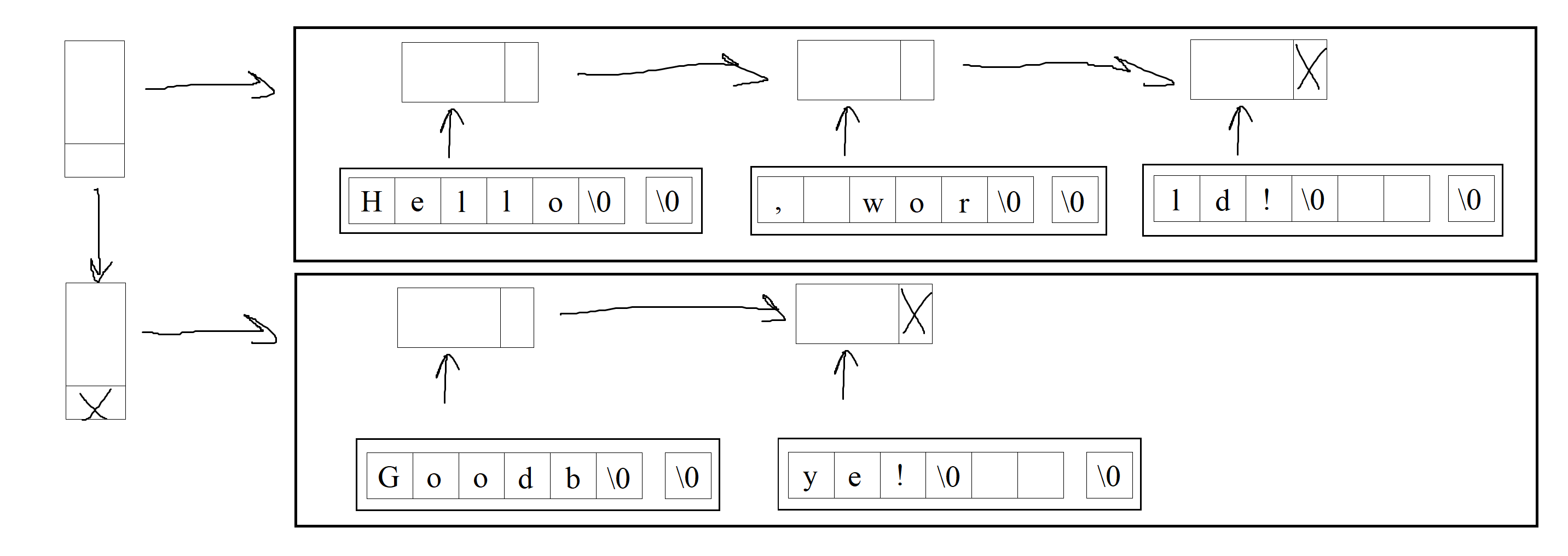


рис. 2 — большой иерархический список

Теперь к изменению списков. Переворот списка имеет следующий алгоритм:

1. Взять начальный элемент

2. Поменять местами взятый элемент и следующий за ним

3. Повторять до тех пор, пока первый элемент не встанет в конец списка

4. Зафиксировать первый элемент как конечный

Перестановка элементов имеет следующие правила:

1. Если элементов 2, переставить начальный и конечный

2. Если переставляются начальный и/или конечный, взять только следующий за начальным и/или предыдущий за конечным

3. В остальных случаях брать и предыдущие, и следующие элементы, и переставлять в них связи

Ответ:

Текст с измененной последовательностью блоков в строках или сообщение, что текст пустой, операции не были произведены.

**Контрольный пример.**

В качестве контрольного примера был придуман следующий текст:

Hello, world! How

Hey!

Well, I'm OK!

Fine!

Фиксированные настройки — обработка с 1-ой строки и считывание в блоки по 5 символов. При выборе переворота списков текст будет выглядеть так:

ow ld! H, worHello

! Hey

OK!, I'm Well

ne! Fi

А при выборе попарной перестановки — так:

ld! How Hello, wor

! Hey

Well, I'm OK!

ne!-> Fi

**Особенности реализации задания на компьютере.**

Поскольку используемая структура является иерархическим списком, то и структура программы тоже становится «иерархической» — базовый класс строки остается, но с маркером (рис. 3):

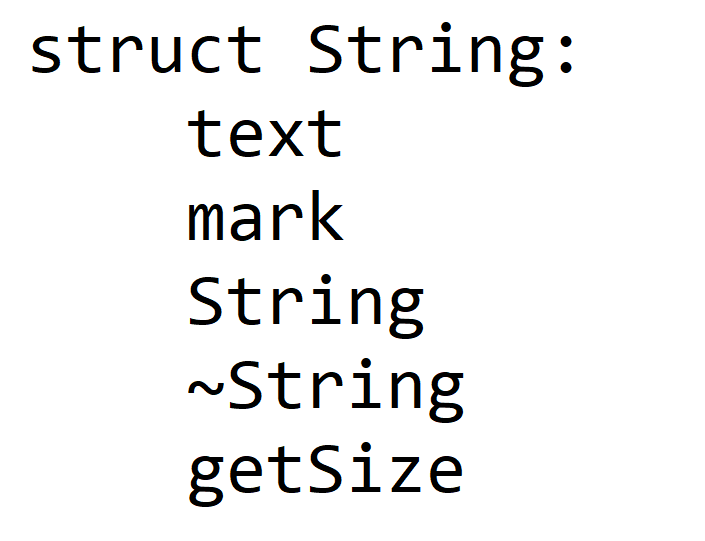


рис. 3 — описание структуры String

Маркер введен в описание, поскольку размер строк-блоков уже известен и передается как параметр по умолчанию, и это правило распространяется и на полупустые строки, в которых после содержательной части остается мусор. Именно поэтому нужен маркер: разграничить эти участки блока.

Эта структура входит в описание элемента так называемого «горизонтального» списка (рис. 4):

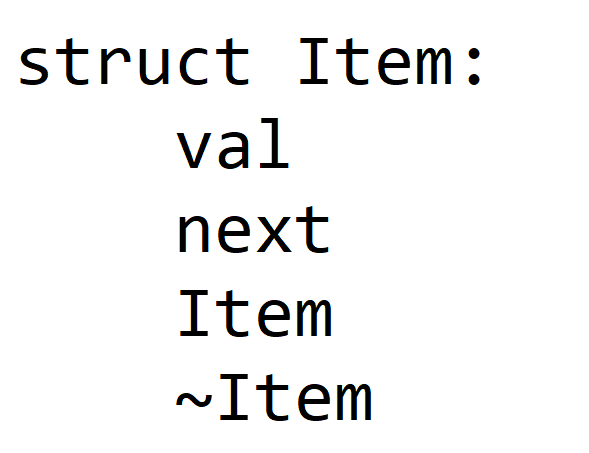


рис. 4 — описание структуры Item (базовый элемент горизонтального списка)

На деле горизонтальный список является уже привычным формуляром к однонаправленному списку, содержащим в себе цепочку из элементов (рис. 5):

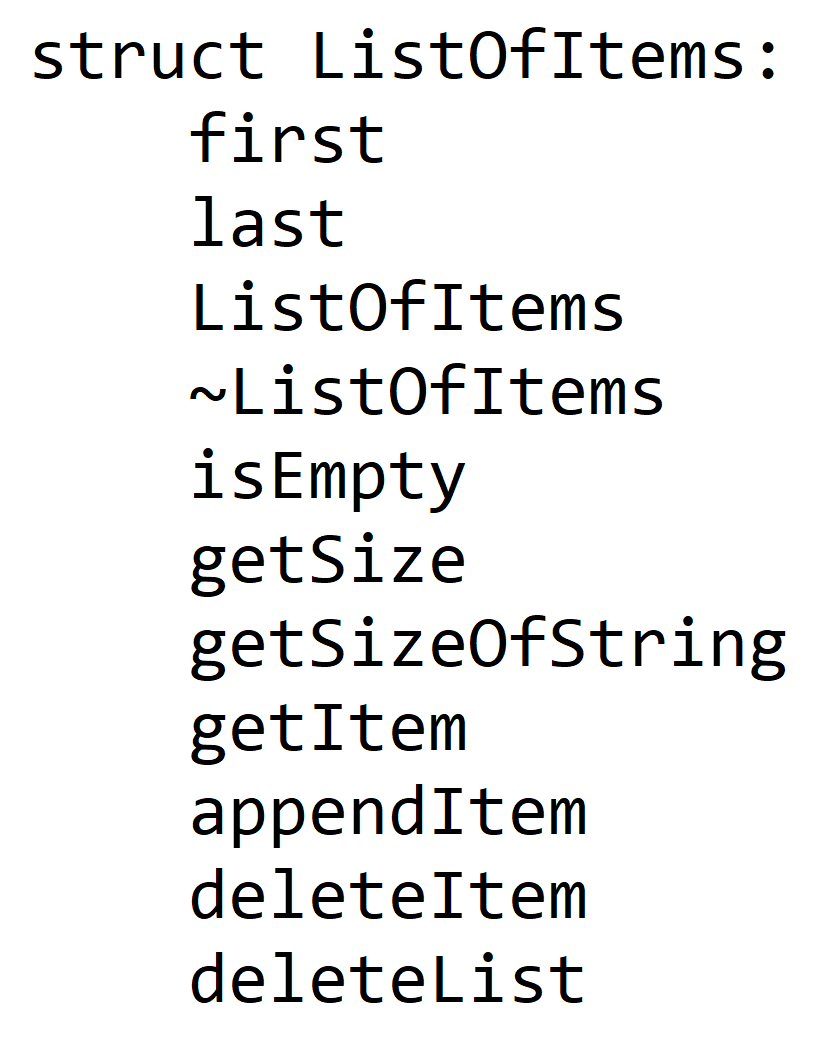


рис. 5 — формуляр горизонтального списка

А вот дальше — новое. Описанный формуляр к горизонтальному списку можно внести в описание элемента «вертикального» списка (рис. 6):

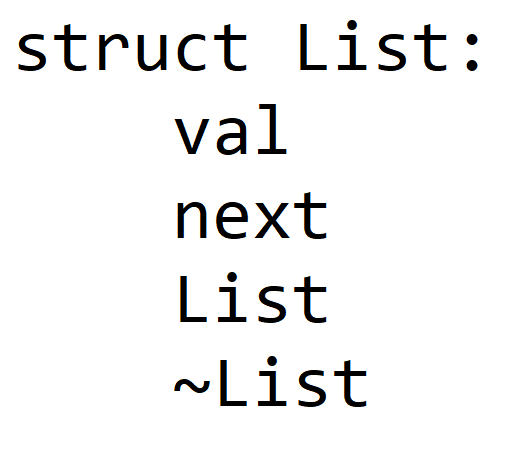


рис. 6 — описание элемента большого списка

Изначальная реализация через указатели является удачной, поскольку в таком случае к каждому из формуляров есть прямой доступ, а он, в свою очередь, предоставляет прямой доступ к функционалу отдельно взятому формуляру, что и является основной фишкой иерархических списков.

Конечной оберткой поверх всего описанного функционала является формуляр к так называемому «вертикальному» списку (рис. 7):

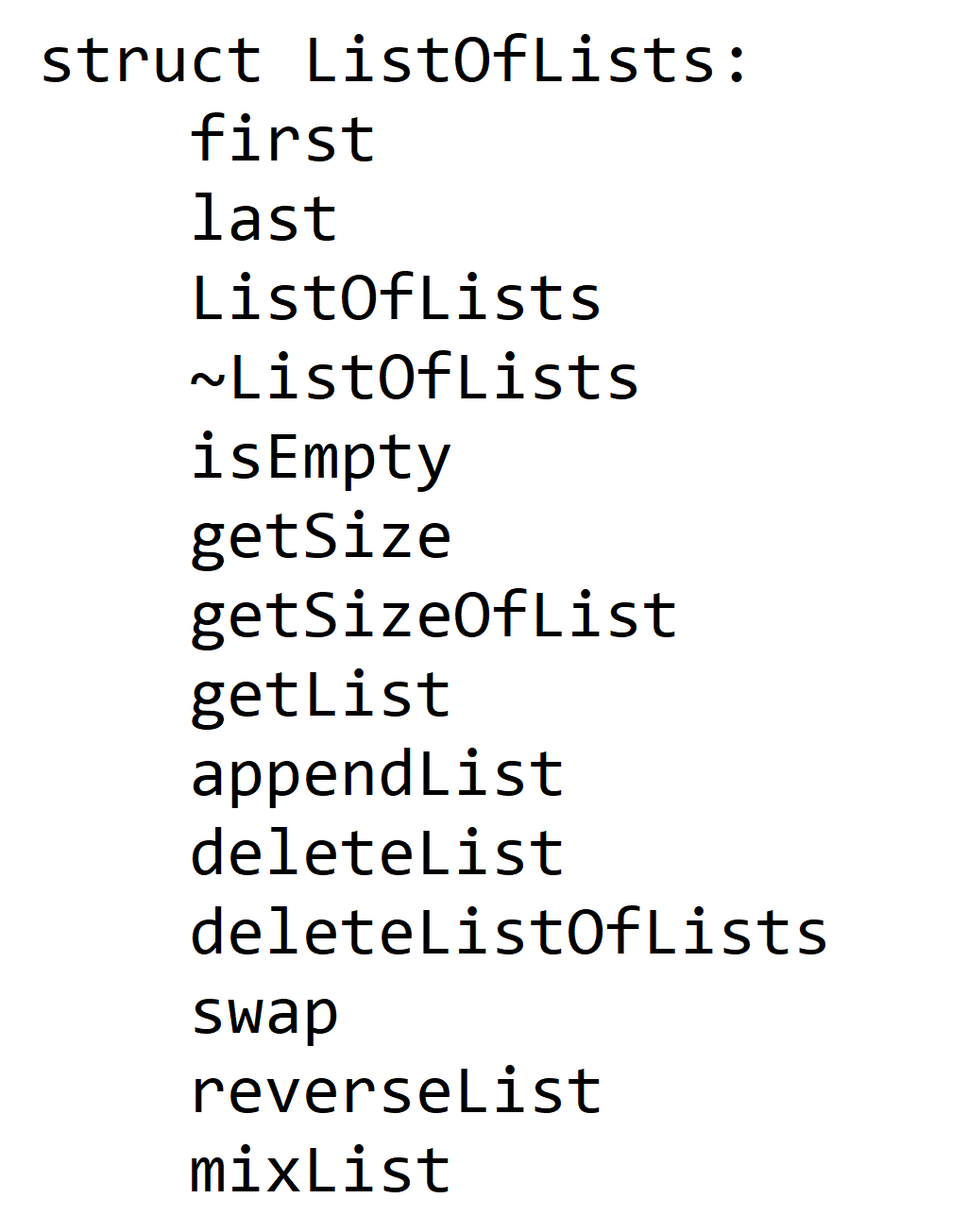


рис. 7 — формуляр к вертикальному списку

Этот формуляр оперирует формулярами как элементами со своим функционалом, а также имеет свой функционал же поверх всего реализованного.

Схема включений в программе следующая (рис. 8):

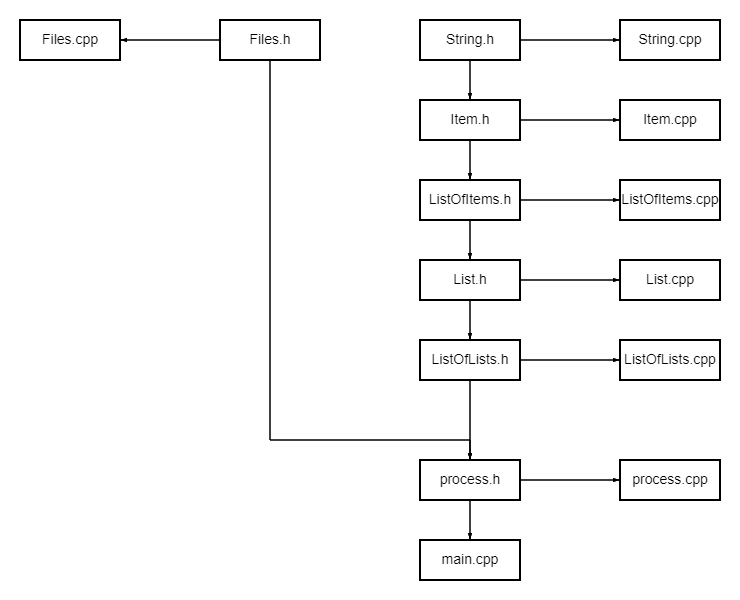


рис. 8 — схема включений в программе

**Разработка интерфейса пользователя.**

Входной файл:

I1: цифра 1 или 2

I2: индекс начала обработки

I3: текст

Выходной файл:

O1: Ошибка: входной файл не найден.

O2: Ошибка: считан неверный код действия (введите либо 1, либо 2). (если нарушен шаблон I1)

O3: Сформирован следующий список:

O4: Текст из I3 следующего формата (если он был):

блок->блок->…->NULL

|

V

…

|

V

NULL

O5: Список пуст. Дальнейшие операции с ним не имеют смысла. (защита от лишних действий)

O6: Считанный индекс оказался больше фактического размера списка. Дальнейшие операции не имеют смысла. (еще одна защита от лишних действий)

O7: Выбран разворот списков (режим 1). Все списки, начиная с (I2 + 1)-ого по счету, будут перевернуты в обратную сторону.

O8: Выбрана перемешка списков (режим 2). Все списки с четным числом элементов, начиная с (I2 + 1)-ого по счету, будут попарно перемешаны.

O9: Итоговый список:

O10: Шаблон O4 еще раз (текст изменен)

O11: Проверка на корректное удаление:

До:

O12: адрес начала списка, адрес конца списка

O13: После:

O14: адрес начала списка, адрес конца списка (изменены)

Лог-файл:

O1: Программа начала свою работу.

Производится попытка открыть входной файл.

O2: Входной файл не найден.

O3: Производится попытка считать код действия и индекс.

O4: Считан неверный код действия.

O5: Сформирован следующий список:

O6: Текст из I3 следующего формата (если он был):

блок->блок->…->NULL

|

V

…

|

V

NULL

O7: Список пуст. Дальнейшие операции с ним не будут выполняться. (защита от лишних действий)

O8: Считанный индекс оказался больше фактического размера списка. Дальнейшие операции не будут выполняться. (еще одна защита от лишних действий)

O9: Производится разворот всех списков, начиная с (I2 + 1)-ого по счету.

O10: Производится перемешка всех списков с четным числом элементов, начиная с (I2 + 1)-ого по счету.

O11: Итоговый список:

O12: Шаблон O4 еще раз (текст изменен)

O13: Проверка на корректное удаление:

До:

O14: адрес начала списка, адрес конца списка

O15: После:

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

O17: Программа завершила свою работу.

**Описание используемых данных.**

Структура Files:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Имя | Модификатор | Назначение |
| std::ifstream | inFile | public | входной файл |
| std::ofstream | outFile | выходной файл |
| logFile | лог-файл |

Структура String:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Имя | Модификатор | Назначение |
| char\* | text | public | содержимое строки |
| char | mark | маркер |

Структура Item:

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

Структура ListOfItems:

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

Структура List:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Имя | Модификатор | Назначение |
| ListOfItems\* | val | public | указатель на содержимое элемента |
| List\* | next | указатель на следующий элемент |

Структура ListOfLists:

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

Отдельные переменные:

|  |  |  |
| --- | --- | --- |
| Тип | Имя | Назначение |
| Files\* | files | файловая переменная |
| int | mode | выбранное действие обработки |
| index | индекс начала обработки |
| charsCount | счетчик символов в блоке |
| blocksCount | счетчик блоков |
| strCount | счетчик строк |
| char | tmp | заглушка для считывания символов |
| ListOfLists\* | list | переменная для иерархического списка |
| List\* | listItem | переменная для элементов вертикального списка |
| tmpList | переменная для вывода элементов вертикального списка |
| Item\* | item | переменная для элементов горизонтального списка |
| tmpItem | переменная для вывода элементов горизонтального списка |

**Описание используемых функций.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Имя | Файл | Назначение | Параметры | | | |
| входные | выходные | модифицируемые | транзитные |
| Files::Files | Files.cpp | конструктор Files | in, out, log |  |  |  |
| Files::~Files | деструктор Files |  |  |  |  |
| String::String | String.cpp | конструктор String | size |  |  |  |
| String::~String | деструктор String |  |  |  |  |
| String::getSize | возврат размера содержательной части строки |  | i |  |  |
| Item::Item | Item.cpp | конструктор Item | size |  |  |  |
| Item::~Item | деструктор Item |  |  |  |  |
| ListOfItems::ListOfItems | ListOfItems.cpp | конструктор ListOfItems |  |  |  |  |
| ListOfItems::~ListOfItems | деструктор ListOfItems |  |  |  |  |
| ListOfItems::isEmpty | проверка списка на пустоту |  |  |  |  |
| ListOfItems::getSize | возврат размера списка в блоках |  | length |  |  |
| ListOfItems::getSizeOfString | возврат размера строки |  | length |  |  |
| ListOfItems::getItem | возврат блока по заданному индексу, если он корректен | index | p |  |  |
| ListOfItems::appendItem | добавление блока в конец списка | val |  |  |  |
| ListOfItems::deleteItem | удаление блока по заданному индексу, если он корректен | index |  |  |  |
| ListOfItems::deleteList | удаление списка |  |  |  |  |
| List::List | List.cpp | конструктор List |  |  |  |  |
| List::~List | деструктор List |  |  |  |  |
| ListOfLists::ListOfLists | ListOfLists.cpp | конструктор ListOfLists |  |  |  |  |
| ListOfLists::~ListOfLists | деструктор ListOfLists |  |  |  |  |
| ListOfLists::isEmpty | проверка списка на пустоту |  |  |  |  |
| ListOfLists::getSize | возврат размера списка |  | length |  |  |
| ListOfLists::getSizeOfList | возврат размера элемента по заданному индексу, если он корректен | index |  |  |  |
| ListOfLists::getList | возврат списка по заданному индексу, если он корректен | index | p |  |  |
| ListOfLists::appendList | добавление списка в конец | val |  |  |  |
| ListOfLists::deleteList | удаление списка по заданному индексу, если он корректен | index |  |  |  |
| ListOfLists::deleteListOfLists | удаление списка |  |  |  |  |
| ListOfLists::swap | перестановка элементов списка | list, index1, index2 |  |  |  |
| ListOfLists::reverseList | переворот списков | index |  |  |  |
| ListOfLists::mixList | попарная перестановка элементов списка | index |  |  |  |
| reverseLists | process.cpp | вызов reverseList для всех списков, начиная с заданного индекса | list, index |  |  |  |
| mixLists | вызов mixList для всех списков, начиная с заданного индекса | list, index |  |  |  |
| process | решение задачи |  |  |  |  |

**Организация ввода-вывода.**

|  |  |  |  |
| --- | --- | --- | --- |
| Библиотека | Имя | Команда | Назначение |
| fstream |  | >> | считывание с файла |
|  | << | запись в файл |
|  | unsetf | снятие флага чтения/записи |
|  | eof | проверка на достижение EOF |
|  | open | открытие файла |
|  | close | закрытие файла |

**Представление алгоритма решения задачи.**

1. Считать отметку действия и индекс начала

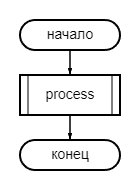
2. Если все корректно, считать текст по алгоритму

3. Если текст непустой и индекс покрывает полученный список, выполнить действие

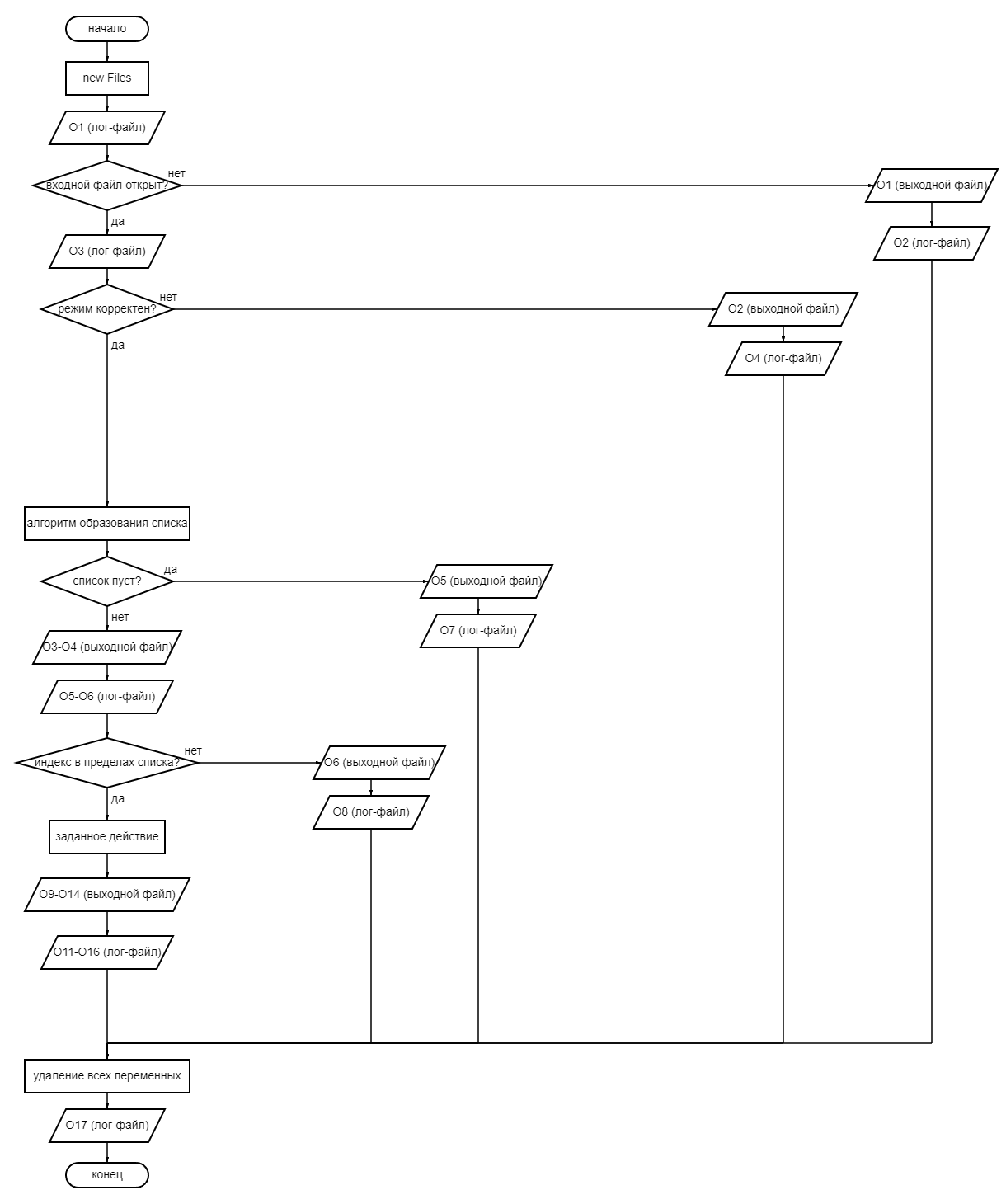
4. Во всех противных случаях прервать решение задачи

**Блок-схема.**

main:



process:



**Текст программы.**

main.cpp:

/\*

Программа по поблочному считыванию строк текста, формированию из них двухуровневого иерархического списка

и обработке полученных списков по выбору пользователя

Автор: Савин Павел Алексеевич, группа 2309. Версия 4.1.1

Даты: Начало: 04.04.2023 Окончание: 11.04.2023

\*/

#include "process.h"

int main() {

process();

return 0;

}

Files.h (Files):

#pragma once

#include <fstream>

struct Files {

std::ifstream inFile;

std::ofstream outFile;

std::ofstream logFile;

Files(const char\* in = "in.txt", const char\* out = "out.txt", const char\* log = "log.txt");

~Files();

};

Files.cpp (Files):

#include "Files.h"

Files::Files(const char\* in, const char\* out, const char\* log) {

this->inFile.open(in);

this->outFile.open(out);

this->logFile.open(log);

}

Files::~Files() {

this->inFile.close();

this->outFile.close();

this->logFile.close();

}

String.h (String):

#pragma once

struct String {

char\* text;

char mark;

String(const int size);

~String();

int getSize();

};

String.cpp (String):

#include "String.h"

String::String(const int size) {

this->text = new char[size + 1];

this->text[0] = '\0';

this->mark = '\0';

}

String::~String() {

delete[] this->text;

}

int String::getSize() {

int i = 0;

while (this->text[i++] != this->mark);

return i;

}

Item.h (Item):

#pragma once

#include "String.h"

struct Item {

String\* val;

Item\* next;

Item(const int size);

~Item();

};

Item.cpp (Item):

#include "Item.h"

Item::Item(const int size) {

this->val = new String(size);

this->next = nullptr;

}

Item::~Item() {

delete this->val;

}

ListOfItems.h (ListOfItems):

#pragma once

#include "Item.h"

struct ListOfItems {

Item\* first;

Item\* last;

ListOfItems();

~ListOfItems();

bool isEmpty();

int getSize();

int getSizeOfString();

Item\* getItem(const int index);

void appendItem(Item\* val);

void deleteItem(const int index);

void deleteList();

};

ListOfItems.cpp (ListOfItems):

#include "ListOfItems.h"

ListOfItems::ListOfItems() {

this->first = nullptr;

this->last = nullptr;

}

ListOfItems::~ListOfItems() {

this->deleteList();

}

bool ListOfItems::isEmpty() {

return this->first == nullptr;

}

int ListOfItems::getSize() {

int length = 0;

Item\* p = this->first;

while (p != nullptr) {

++length;

p = p->next;

}

return length;

}

int ListOfItems::getSizeOfString() {

int length = 0;

Item\* p = this->first;

while (p != nullptr) {

length += p->val->getSize();

}

return length;

}

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

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

int pos = 0;

Item\* p = this->first;

while (pos != index) {

++pos;

p = p->next;

}

return p;

}

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

if (this->isEmpty()) {

this->first = val;

this->last = val;

}

else {

this->last->next = val;

this->last = val;

}

}

void ListOfItems::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 ListOfItems::deleteList() {

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

while (this->first != nullptr) {

this->deleteItem(curIndex);

--curIndex;

}

}

List.h (List):

#pragma once

#include "ListOfItems.h"

struct List {

ListOfItems\* val;

List\* next;

List();

~List();

};

List.cpp (List):

#include "List.h"

List::List() {

this->val = new ListOfItems;

this->next = nullptr;

}

List::~List() {

delete this->val;

}

ListOfLists.h (ListOfLists):

#pragma once

#include "List.h"

struct ListOfLists {

List\* first;

List\* last;

ListOfLists();

~ListOfLists();

bool isEmpty();

int getSize();

int getSizeOfList(const int index);

List\* getList(const int index);

void appendList(List\* val);

void deleteList(const int index);

void deleteListOfLists();

void swap(List\* list, const int index1, const int index2);

void reverseList(const int index);

void mixList(const int index);

};

ListOfLists.cpp (ListOfLists):

#include "ListOfLists.h"

ListOfLists::ListOfLists() {

this->first = nullptr;

this->last = nullptr;

}

ListOfLists::~ListOfLists() {

this->deleteListOfLists();

}

bool ListOfLists::isEmpty() {

return this->first == nullptr;

}

int ListOfLists::getSize() {

int length = 0;

List\* p = this->first;

while (p != nullptr) {

++length;

p = p->next;

}

return length;

}

int ListOfLists::getSizeOfList(const int index) {

if (index < 0 || index >= this->getSize()) return -1;

return this->getList(index)->val->getSizeOfString();

}

List\* ListOfLists::getList(const int index) {

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

int pos = 0;

List\* p = this->first;

while (pos != index) {

++pos;

p = p->next;

}

return p;

}

void ListOfLists::appendList(List\* val) {

if (this->isEmpty()) {

this->first = val;

this->last = val;

}

else {

this->last->next = val;

this->last = val;

}

}

void ListOfLists::deleteList(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 {

List\* del = this->getList(index);

if (index == 0) {

this->first = del->next;

delete del;

}

else if (index == size - 1) {

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

this->last->next = nullptr;

delete del;

}

else {

List\* prev = this->getList(index - 1);

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

delete del;

prev = nullptr;

}

del = nullptr;

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

this->last = this->first;

}

}

}

void ListOfLists::deleteListOfLists() {

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

while (this->first != nullptr) {

this->deleteList(curIndex);

--curIndex;

}

}

void ListOfLists::swap(List\* list, const int index1, const int index2) {

Item\* toSwap1 = list->val->getItem(index1);

Item\* toSwap2 = list->val->getItem(index2);

if (list->val->getSize() == 2) {

toSwap2->next = toSwap1;

toSwap1->next = nullptr;

list->val->first = toSwap2;

list->val->last = toSwap1;

}

else if (toSwap1 == list->val->first && toSwap2 == list->val->last) {

Item\* next1 = list->val->getItem(index1 + 1);

Item\* prev2 = list->val->getItem(index2 - 1);

toSwap2->next = next1;

prev2->next = toSwap1;

toSwap1->next = nullptr;

list->val->first = toSwap2;

list->val->last = toSwap1;

}

else if (toSwap1 == list->val->first && toSwap2 != list->val->last) {

Item\* next1 = list->val->getItem(index1 + 1);

Item\* prev2 = list->val->getItem(index2 - 1);

Item\* next2 = list->val->getItem(index2 + 1);

prev2->next = toSwap1;

toSwap1->next = next2;

toSwap2->next = next1;

list->val->first = toSwap2;

}

else if (toSwap1 != list->val->first && toSwap2 == list->val->last) {

Item\* prev1 = list->val->getItem(index1 - 1);

Item\* next1 = list->val->getItem(index1 + 1);

Item\* prev2 = list->val->getItem(index2 - 1);

prev1->next = toSwap2;

toSwap2->next = next1;

prev2->next = toSwap1;

toSwap1->next = nullptr;

list->val->last = toSwap1;

}

else {

Item\* prev1 = list->val->getItem(index1 - 1);

Item\* next1 = list->val->getItem(index1 + 1);

Item\* prev2 = list->val->getItem(index2 - 1);

Item\* next2 = list->val->getItem(index2 + 1);

prev1->next = toSwap2;

toSwap2->next = next1;

prev2->next = toSwap1;

toSwap1->next = next2;

}

}

void ListOfLists::reverseList(const int index) {

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

List\* list = this->getList(index);

Item\* cur = list->val->first;

Item\* tmp = nullptr;

Item\* prev = nullptr;

while (cur != nullptr) {

tmp = cur->next;

cur->next = prev;

prev = cur;

cur = tmp;

}

list->val->first = prev;

}

void ListOfLists::mixList(const int index) {

List\* list = this->getList(index);

int size = list->val->getSize();

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

if (size % 2 != 0) return;

if (size == 2) {

this->swap(list, 0, 1);

return;

}

for (int i = 0; i < size / 2; ++i) this->swap(list, i, i + 2);

}

process.h:

#pragma once

#include "Files.h"

#include "ListOfLists.h"

void reverseLists(ListOfLists\* list, const int index);

void mixLists(ListOfLists\* list, const int index);

void process();

process.cpp:

#include "process.h"

const unsigned int N = 5;

void reverseLists(ListOfLists\* list, const int index) {

for (int i = index; i < list->getSize(); ++i) list->reverseList(i);

}

void mixLists(ListOfLists\* list, const int index) {

for (int i = index; i < list->getSize(); ++i) list->mixList(i);

}

void process() {

Files\* files = new Files;

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

if (files->inFile.is\_open()) {

files->logFile << "Производится попытка считать код действия и индекс.\n";

int mode;

files->inFile >> mode;

if (mode == 1 || mode == 2) {

int index;

files->inFile >> index;

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

char tmp;

ListOfLists\* list = new ListOfLists;

List\* listItem = new List;

Item\* item = new Item(N);

int charsCount = 0, blocksCount = 0, strCount = 0;

files->inFile >> tmp;

do {

files->inFile >> tmp;

if (files->inFile.eof()) {

if (charsCount || blocksCount || strCount) {

listItem->val->appendItem(item);

list->appendList(listItem);

listItem = nullptr;

item = nullptr;

charsCount = 0;

++blocksCount;

++strCount;

}

else {

delete listItem;

delete item;

}

break;

}

if (tmp == '\n') {

listItem->val->appendItem(item);

++blocksCount;

list->appendList(listItem);

++strCount;

item = nullptr;

listItem = nullptr;

item = new Item(N);

listItem = new List;

charsCount = 0;

}

else {

if (charsCount == N) {

listItem->val->appendItem(item);

++blocksCount;

item = nullptr;

item = new Item(N);

charsCount = 0;

}

item->val->text[charsCount] = tmp;

item->val->text[charsCount + 1] = '\0';

++charsCount;

}

}

while (true);

if (list->first != nullptr) {

files->outFile << "Сформирован следующий список:\n";

files->logFile << "Сформирован следующий список:\n";

List\* tmpList = list->first;

while (tmpList != nullptr) {

Item\* tmpItem = tmpList->val->first;

while (tmpItem != nullptr) {

int i = 0;

while (tmpItem->val->text[i] != '\0') {

files->outFile << tmpItem->val->text[i];

files->logFile << tmpItem->val->text[i];

++i;

}

files->outFile << "->";

files->logFile << "->";

tmpItem = tmpItem->next;

}

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

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

tmpList = tmpList->next;

}

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

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

if (index < 0 || index >= strCount) {

files->outFile << "Считанный индекс оказался больше фактического размера списка. Дальнейшие операции не имеют смысла.\n";

files->logFile << "Считанный индекс оказался больше фактического размера списка. Дальнейшие операции не будут выполняться.\n";

}

else {

if (mode == 1) {

files->outFile << "Выбран разворот списков (режим 1). Все списки, начиная с " << index + 1 << "-ого по счету, будут перевернуты в обратную сторону.\n";

files->logFile << "Производится разворот всех списков, начиная с " << index + 1 << "-ого по счету.\n";

reverseLists(list, index);

}

else {

files->outFile << "Выбрана перемешка списков (режим 2). Все списки с четным числом элементов, начиная с " << index + 1 << "-ого по счету, будут попарно перемешаны.\n";

files->logFile << "Производится перемешка всех списков с четным числом элементов, начиная с " << index + 1 << "-ого по счету.\n";

mixLists(list, index);

}

files->outFile << "Итоговый список:\n";

files->logFile << "Итоговый список:\n";

List\* tmpList = list->first;

while (tmpList != nullptr) {

Item\* tmpItem = tmpList->val->first;

while (tmpItem != nullptr) {

int i = 0;

while (tmpItem->val->text[i] != '\0') {

files->outFile << tmpItem->val->text[i];

files->logFile << tmpItem->val->text[i];

++i;

}

files->outFile << "->";

files->logFile << "->";

tmpItem = tmpItem->next;

}

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

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

tmpList = tmpList->next;

}

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

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

}

}

else {

files->outFile << "Список пуст. Дальнейшие операции с ним не имеют смысла.";

files->logFile << "Список пуст. Дальнейшие операции не будут выполняться.\n";

}

files->outFile << "Проверка на корректное удаление:\nДо:\n" << list->first << ' ' << list->last;

files->logFile << "Проверка на корректное удаление:\nДо:\n" << list->first << ' ' << list->last;

delete list;

files->outFile << "\nПосле:\n" << list->first << ' ' << list->last;

files->logFile << "\nПосле:\n" << list->first << ' ' << list->last << '\n';

}

else {

files->outFile << "Ошибка: считан неверный код действия (введите либо 1, либо 2).";

files->logFile << "Считан неверный код действия.\n";

}

}

else {

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

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

}

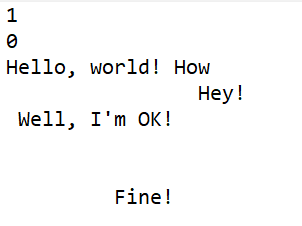
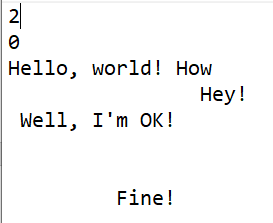
files->logFile << "Программа завершила свою работу.";

delete files;

}

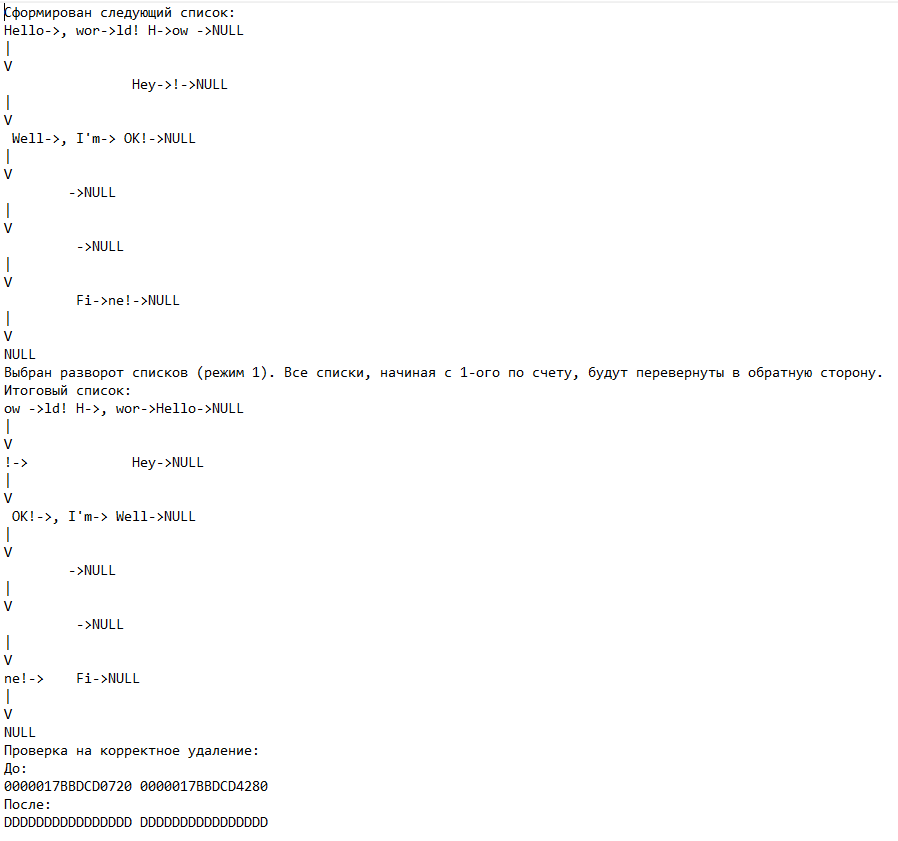
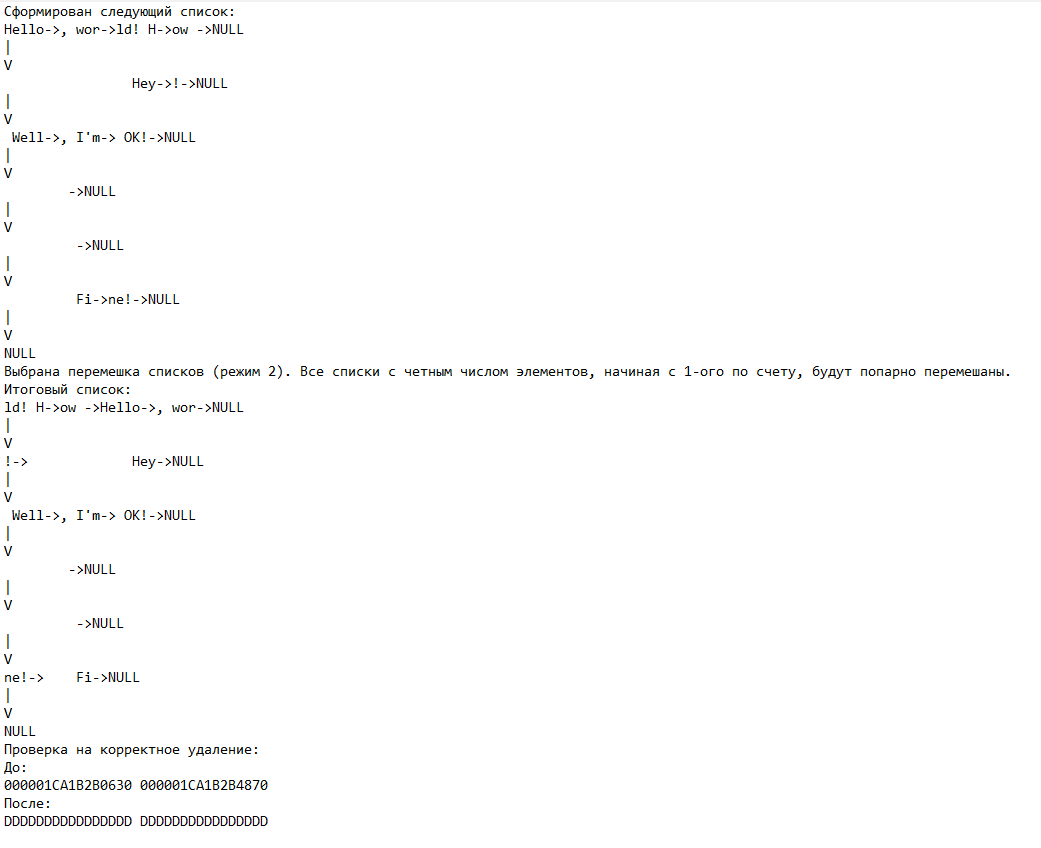
**Результаты работы программы.**

Контрольный пример:

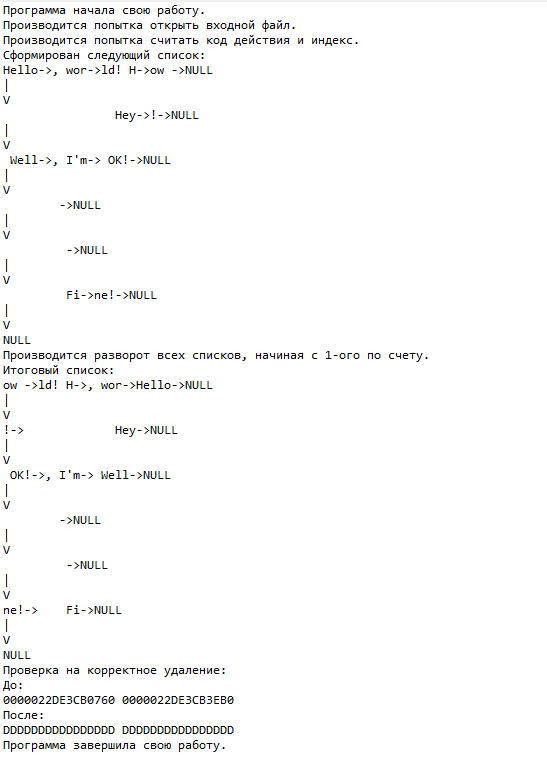
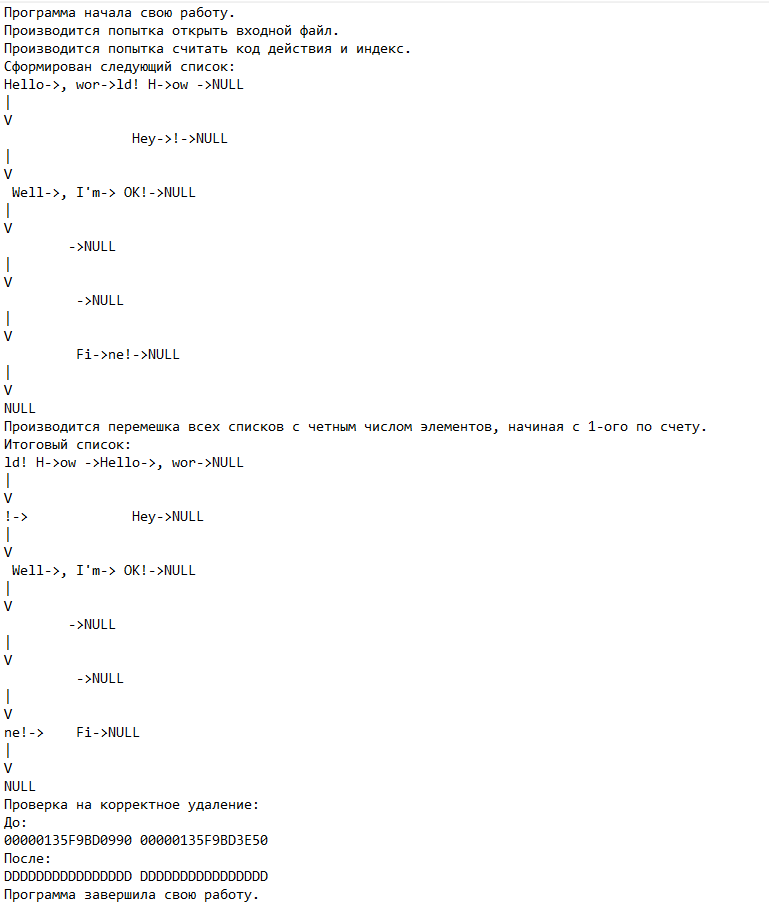
 

выдает следующие результаты:

Выходной файл:

Лог-файл:

**Вывод о проделанной работе.**

Изучено использование формуляров к однонаправленным спискам в качестве элементов большого иерархического списка с прямым доступом ко внутренним элементам списков.