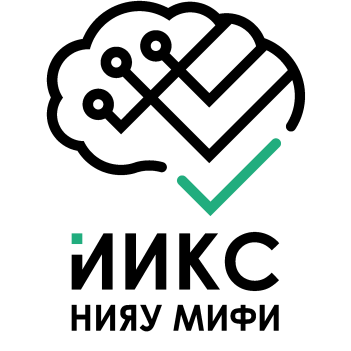
**Национальный исследовательский ядерный университет «МИФИ»**

**Институт интеллектуальных кибернетических систем**

**Кафедра №12 «Компьютерные системы и технологии»**



**ОТЧЕТ**

**О выполнении лабораторной работы №6**

**«Работа со структурами данных**

**на основе списков»**

Студент: Кафанов С.П.

Группа: Б21-515

Преподаватель: Храпов А.С.

Москва — 2021

**Задача. Вариант №14**

Необходимо спроектировать и разработать на языке C программу, осуществляющую обработку строковых данных, на физическом уровне представленных в виде списков.

Из входного потока вводится произвольное количество строк произвольной длины. Каждая строка в общем случае содержит одно или более слов, разделенных пробелами и/или знаками табуляции.

Завершение ввода определяется концом файла.

Каждая выходная строка формируется путем модификации исходной строки в соответствии с требованиями, предъявляемыми индивидуальным заданием. В полученной строке слова разделяются только одним пробелом. Исходная и полученная строки выводятся в кавычках на экран.

Удалить из строки все слова с чётным количеством гласных.

**Использованные типы данных.**

*Int –* для работы с простыми целочисленными данными

*Char –* для строк, чисел небольшого диапазона и работы с байтами памяти

*\* -* для работы с динамической памятью и передачи переменных в функцию, просто раоты с указателями

*Void \* -* для написания универсальных сортировок

Struct – для создания структур (свой тип данных)

**Код.**

#include <stdio.h>  
#include <stdlib.h>  
  
#include "list.h"  
  
List\* init\_list() {  
 List \* l = malloc(sizeof(List));  
 l->begin = NULL;  
 l->end = NULL;  
 l->len = 0;  
 return l;  
}  
  
void delete\_list(List \* l) {  
 Item \*el = l->begin, \*p\_el;  
 while (el) {  
 p\_el = el;  
 el = el->next;  
 free(p\_el);  
 }  
 free(l);  
}  
  
int push\_back(List \* l, char a) {  
 if (l->begin == NULL) {  
 l->begin = malloc(sizeof(Item));  
 if (!l->begin)  
 return -1;  
 l->begin->next = NULL;  
 l->begin->a = a;  
 l->begin->previos = NULL;  
 l->end = l->begin;  
 l->len++;  
 return 0;  
 }  
  
 l->end->next = malloc(sizeof(Item));  
 Item \* el = l->end->next;  
 if (!el)  
 return -1;  
 el->previos = l->end;  
 el->next = NULL;  
 el->a = a;  
 l->end = el;  
 l->len++;  
 return 1;  
}  
  
char get\_c(List \* l, int i) {  
 if (!l->end)  
 return '\0';  
 if (i == -1)  
 return l->end->a;  
  
 Item \* el = l->begin;  
 int j = 0;  
 while (j != i && el->next) {  
 el = el->next;  
 j++;  
 }  
 if (j == i)  
 return el->a;  
 else  
 return '\0';  
}  
  
Item \* get\_el(List \* l, int i) {  
 if (!l->end)  
 return NULL;  
 Item \* el = l->begin;  
 int j = 0;  
 while (j != i && el->next) {  
 el = el->next;  
 j++;  
 }  
 if (j == i)  
 return el;  
 else  
 return NULL;  
}  
  
void copy\_list(List \* l1, int ind1, List \* l2, int ind2, int len) {  
 int from = ind2;  
 int to = ind1;  
 while (from-ind2 < len && get\_el(l2, from)) {  
 Item \* el = get\_el(l1, to);  
 if (el)  
 el->a = get\_el(l2, from)->a;  
 else  
 push\_back(l1, get\_el(l2, from)->a);  
 }  
}  
  
List \* join\_list(List \* l1, List \* l2) {  
 List \* res = init\_list();  
 if (!l1->end && l2->end) {  
 copy\_list(res, 0, l2, 0, l2->len);  
 return res;  
 }  
 else if (!l2->end && l1->end) {  
 copy\_list(res, 0, l1, 0, l1->len);  
 return res;  
 }  
 else if (!l2->end && !l1->end)  
 return res;  
 else {  
 copy\_list(res, 0, l1, 0, l1->len);  
 copy\_list(res, res->len-1, l2, 0, l2->len);  
 return res;  
 }  
}  
  
List \* get\_list\_line() {  
 char buf = '\0';  
 List \* s = init\_list();  
 while (buf != EOF) {  
 buf = getchar();  
 if (buf != EOF && push\_back(s, buf) < 0) {  
 printf("NOT ENOUGH MEMORY.\n");  
 return NULL;  
 }  
 //printf("%c\n", get\_c(s, 0));  
 }  
 return s;  
}  
  
List \* get\_list\_lines() {  
 List \*buf = init\_list();  
 List \*res = init\_list();  
  
 return res;  
}  
  
void delete\_spaces(List \* line) {  
 Item \* el = line->begin;  
 while (el) {  
 if (el->a == '\t')  
 el->a = ' ';  
 el = el->next;  
 }  
  
 if (line->len == 1) {  
 if (line->begin->a == ' ') {  
 line->len = 0;  
 free(line->begin);  
 line->begin = NULL;  
 line->end = NULL;  
 }  
 return;  
 }  
 el = line->begin->next;  
 Item \* el\_p = line->begin;  
 char flag = 0;  
 while (el) {  
 if (el->a == ' ' || el->a == '\n') {  
 if (flag) {  
 if (el->a == ' ') {  
 el\_p->next = el->next;  
 if (el->next)  
 el->next->previos = el\_p;  
 free(el);  
 el = el\_p->next;  
 line->len--;  
 }  
 else {  
 flag = 1;  
 el\_p->previos->next = el;  
 el->previos = el\_p->previos;  
 free(el\_p);  
 el\_p = el;  
 el = el->next;  
 line->len--;  
 }  
 }  
 else {  
 flag = 1;  
 el\_p = el;  
 el = el->next;  
 }  
 }  
 else {  
 flag = 0;  
 el\_p = el;  
 el = el->next;  
 }  
 }  
 if (line->begin && line->begin->a == ' ') {  
 if (line->len == 1) {  
 if (line->begin->a == ' ') {  
 line->len = 0;  
 free(line->begin);  
 line->begin = NULL;  
 line->end = NULL;  
 }  
 return;  
 }  
 line->begin = line->begin->next;  
 free(line->begin->previos);  
 }  
}  
  
void print\_list(List \* l) {  
 Item \* el = l->begin;  
 printf("\n");  
 while (el) {  
 printf("%c", el->a);  
 el = el->next;  
 }  
 printf("\n");  
}  
  
void delete\_words(List \* line) {  
 if (line->begin == NULL) {  
 return;  
 }  
 int GL = 0;  
 char flag = 0;  
 Item \* last = line->begin;  
 Item \* el = line->begin;  
 while (el && last && line->begin) {  
 // если слово закончилось  
 if (el->a == ' ' || el->a == '\n' || el->next == NULL) {  
 if (GL%2 == 0 && GL) {  
 if (last) {  
 if (last->previos) {  
 last->previos->next = el;  
 }  
 if (last->previos)  
 last->previos->next = el;  
 else  
 line->begin = el;  
 if (el) {  
 el->previos = last->previos;  
 }  
 // удалить слово  
 while (last != el) {  
 Item \* t = last;  
 last = last->next;  
 free(t);  
 line->len--;  
 }  
 }  
 else {  
 if (el->next)  
 while (last != NULL) {  
 Item \* t = last;  
 last = last->next;  
 free(t);  
 line->len--;  
 }  
 else {  
 while (last != el) {  
 Item \* t = last;  
 last = last->next;  
 free(t);  
 line->len--;  
 }  
 }  
 printf("LAST IS NULL!!!\n");  
 break;  
 }  
  
 }  
 if (!el)  
 break;  
 last = el;  
 el = el->next;  
 GL = 0;  
 }  
 // если буква в слове  
 else {  
 if (el->a == 'A' || el->a == 'E' || el->a == 'I' || el->a == 'O' || el->a == 'U' || el->a == 'Y')  
 GL++;  
 else if (el->a == 'a' || el->a == 'e' || el->a == 'i' || el->a == 'o' || el->a == 'u' || el->a == 'y')  
 GL++;  
 el = el->next;  
 }  
 }  
}

**Тесты.**

Тексты для тестов:

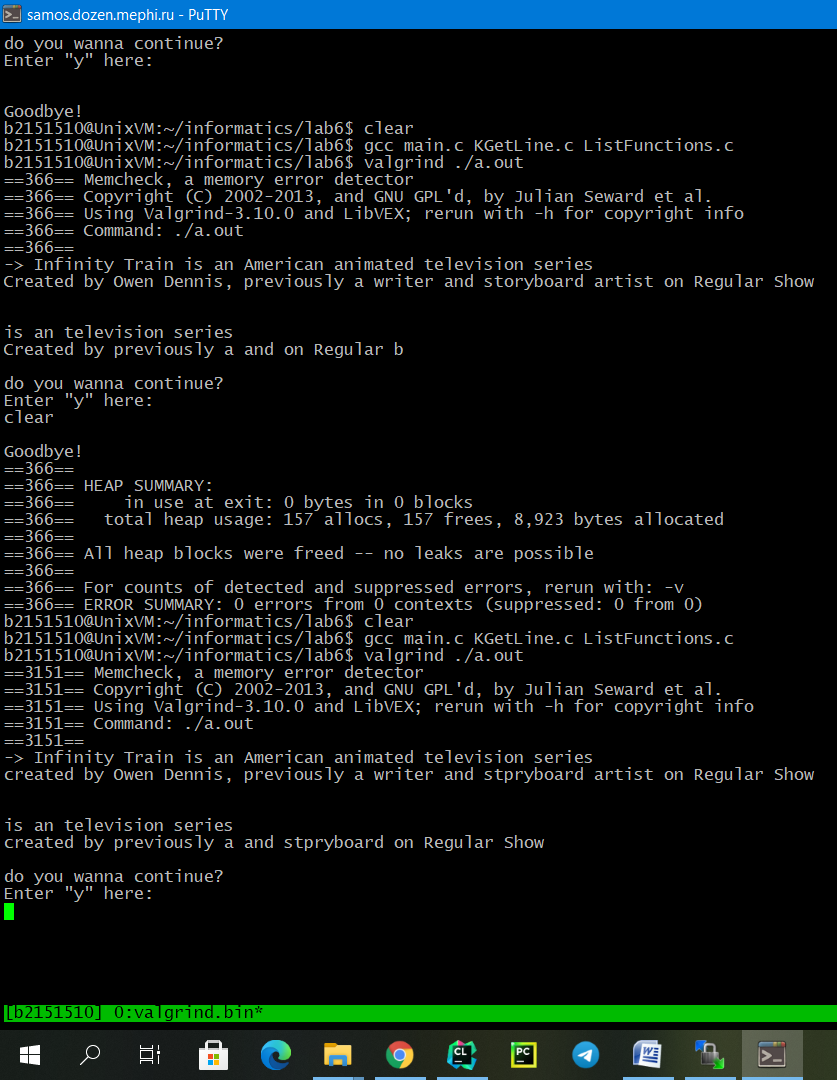
* Infinity Train is an American animated television series

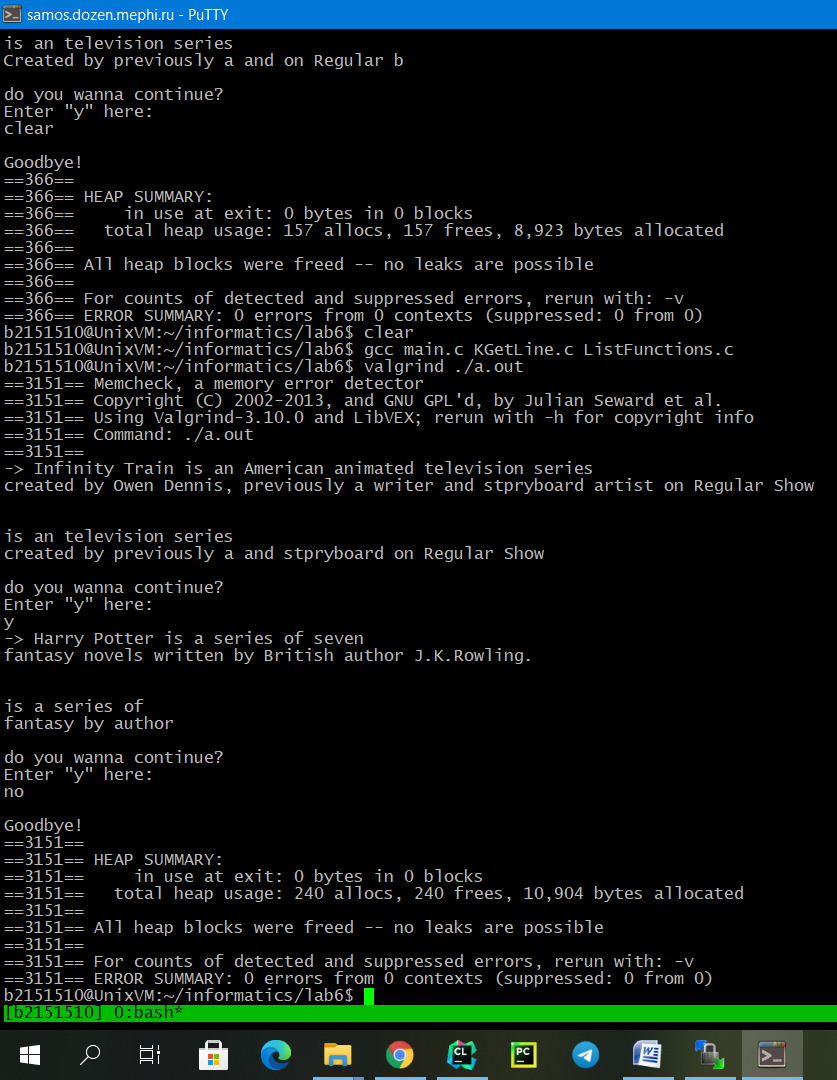
created by Owen Dennis, previously a writer and storyboard artist on Regular Show.

* Harry Potter is a series of seven

fantasy novels written by British author J. K. Rowling.

**Скриншоты.**





**Блоксхемы функций обработки.**

*Сама структура*

typedef struct Item {  
 struct Item \* next;  
 struct Item \* previos;  
 char a;  
} Item;  
  
typedef struct List {  
 Item \* begin;  
 Item \* end;  
 int len;  
} List;

init\_lits() и delete\_list() в моём исполнении практически не отличаются от лекционных вариантов, а потому я не вижу смысла в очередной раз пересказывать их детальное устройство.

1

2

3

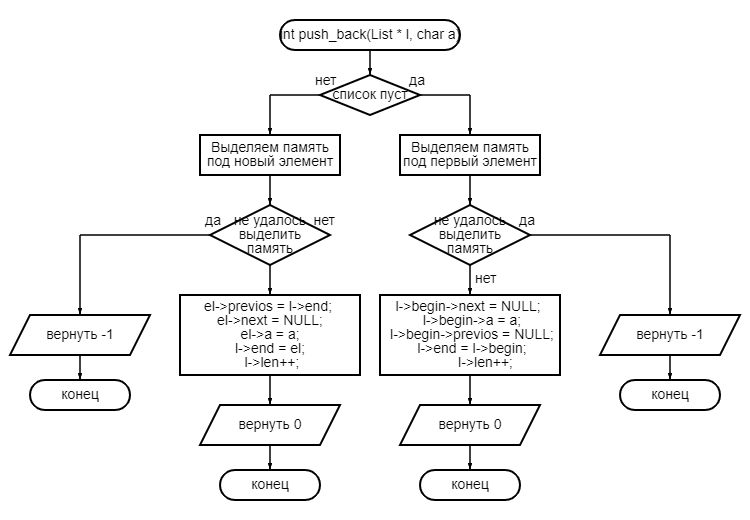
4

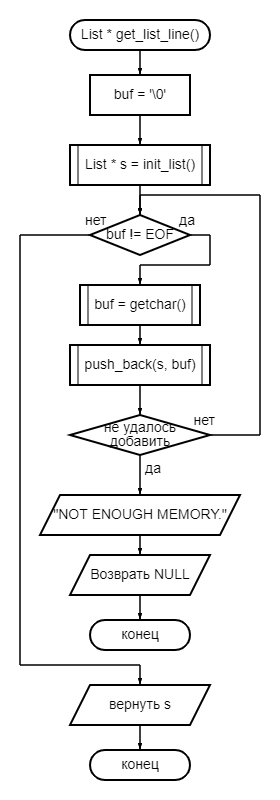
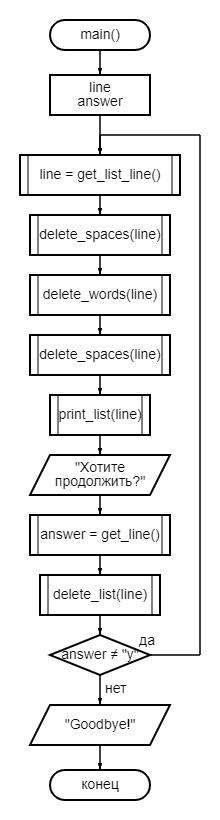
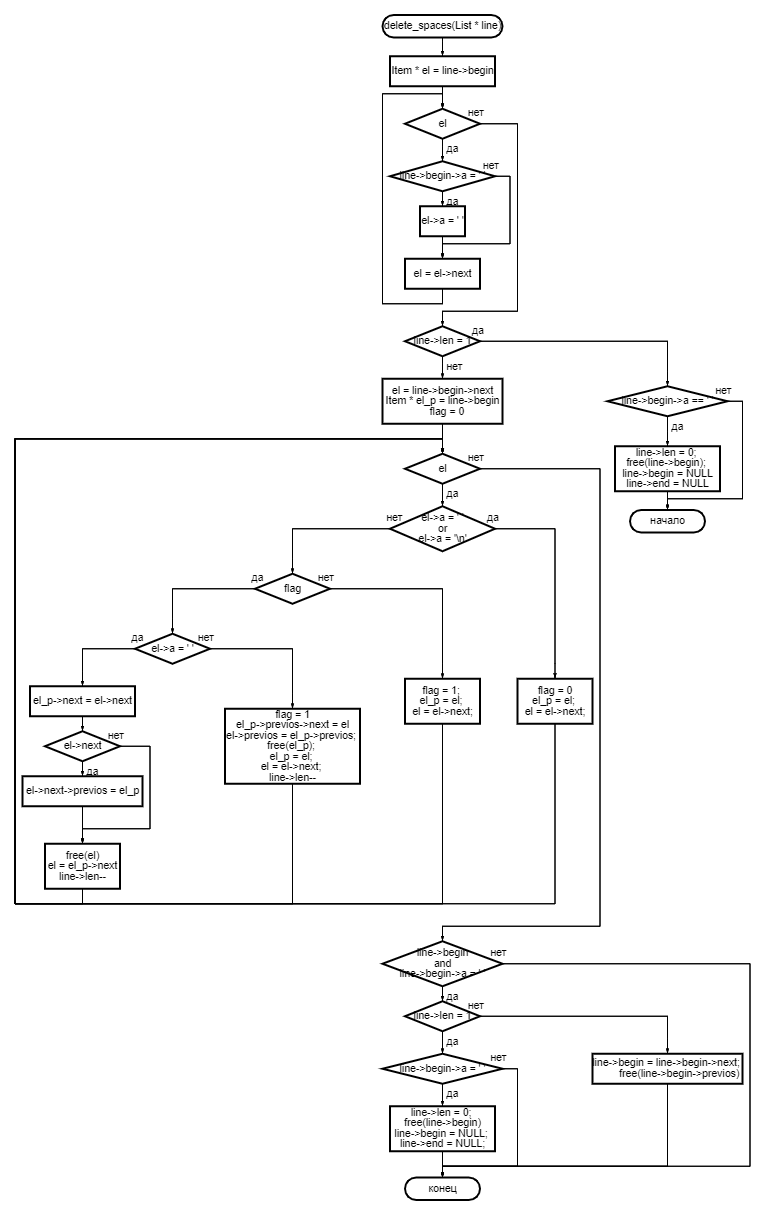
5

6

7

A B C D



1

2

3

4

5

6

7

8

9

10

11

12

13

14

A B

1

2

3

4

5

6

7

8

10

11

12

13

14

15

16

17

18

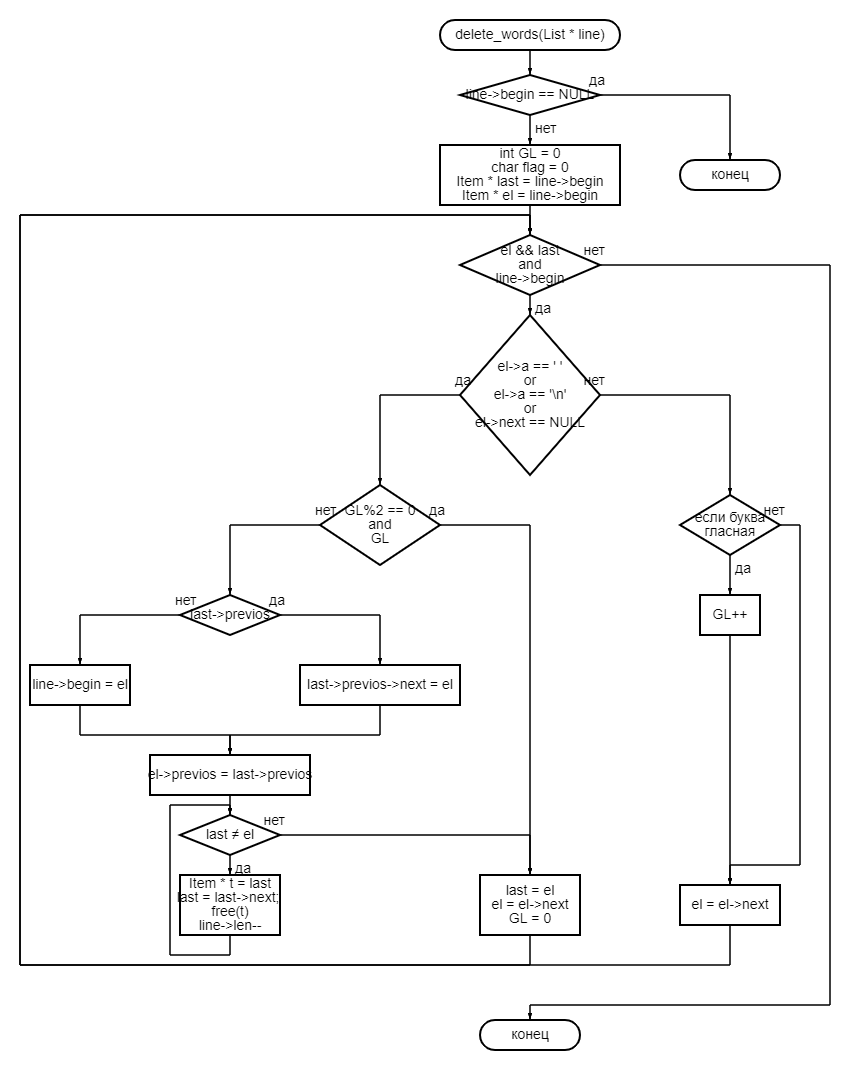
19

20

21

22

A B C D E



1

2

3

4

5

6

7

8

9

10

11

12

A B C D E

**Выводы:**

В данной лабораторной работе я:

* Научился работать со связными списками (в ключе замены классическим строкам).
* Закрепил навыки работы со структурами и указателями.

Из интересного опыта – работа со структурами была практически безболезненной, удобно удалять элементы, написать второй get\_line оказалось куда как проще чем в первый раз.