**Курсовая работа**

**по дисциплине: «Основы программирования на языке C»**

2019 г

# Задание

Разработать приложение на языке C, реализующее следующие функции.

Водится массив строк текста с неизвестным количеством строк.

Ввод текста заканчивается, если повторно введена любая ранее введенная строка.

Затем ввести массив символов разделителей.

Признаком окончания ввода этого массива является ввод символа, код которого больше кодов всех ранее введенных символов разделителей.

Преобразовать исходный массив таким образом, чтобы в каждой строке текста слова были упорядочены в порядке возрастания количества согласных букв.

Символы разделители не переставляется и при необходимости сторону.

Вывести преобразованный текст.

Оглавление

[Задание 2](#_Toc10306425)

[Введение 4](#_Toc10306426)

[1 Исходные данные и способ их ввода 5](#_Toc10306427)

[1.1 Ввод данных из памяти программы. 5](#_Toc10306428)

[1.2 Ввод данных с клавиатуры. 5](#_Toc10306429)

[2 Основные модули программы 6](#_Toc10306430)

[3 Описание функций 7](#_Toc10306431)

[3.1 Функции модуля main 7](#_Toc10306432)

[3.2 Функции модуля utils 7](#_Toc10306433)

[3.3 Функции модуля lines 7](#_Toc10306434)

[3.4 Функции модуля words 7](#_Toc10306435)

[4 Описание порядка работы с программой 9](#_Toc10306436)

[Заключение 10](#_Toc10306437)

[Список литературы 11](#_Toc10306438)

[Приложение А. Исходный код программы 12](#_Toc10306439)

# Введение

В настоящее время, с развитием научно-технического прогресса и информационных технологий, сложность автоматизируемых предметных областей постоянно и неуклонно возрастает. Программист проводит всё своё рабочее время, по сути, в борьбе со сложностью программ, отлаживая свой код в поисках допущенных ошибок.

Ради борьбы со сложностью эволюционируют языки программирования. После машинных кодов и перфокарт появились текстовые редакторы и язык ассемблера, что позволило писать чуть более сложные и объёмные программы при меньших трудозатратах. Далее появились процедурные языки высокого уровня – C, Паскаль и многие другие, что перевело технологии программирования на новый уровень.

Данная работа выполнена в виде программы на языке программирования C. Программа предназначена для работы со списками поступающих на вход строк текста и сортировки их в соотвествии с заданными критериями.

Пользовательский интерфейс выполнен в виде текстовой консоли.

# Исходные данные и способ их ввода

Исходными данными в программе являются две последовательности – набор строк исходного текста и перечень символов-разделителей. Основной режим получения данных для обработки – ввод пользователем с клавиатуры. Также для демонстрационных и отладочных целей предусмотрен ввод данных из внутренней памяти программы, т.е. из предопределённых наборов значений.

## Ввод данных из памяти программы.

В исходном коде для каждого из наборов входных данных – строки и разделители – определён массив предустановленных значений, что позволяет не тратить время на ввод тестовых данных при каждом запуске программы в процессе отладки либо при необходимости оперативно продемонстрировать её работу.

## Ввод данных с клавиатуры.

Во время сеанса работы программы пользователю предлагается сперва ввести последовательность текстовых строк. Ввод каждой строки подтверждается нажатием клавиши Enter. Для того чтобы завершить ввод последовательности строк, необходимо ввести строку, которая уже вводилась ранее.

Далее программа предложит ввести последовательность символов-разделителей. Так же, как и при вводе строк, каждый символ подтверждается нажатием клавиши Enter. При сохранении очередного символа, программа складывает значение его ASCII-кода с кодами предыдущих введённых символов, накапливая таким образом суммарное значение. Для завершения ввода разделителей, необходимо ввести символ, код которого превышает сумму кодов ранее введённых символов.

# Основные модули программы

Программа состоит из 5 модулей, каждый из которых отвечает за свой функционал:

* **main** – основной модуль, управляющий текущим состоянием программы;
* **utils** – в этот модуль вынесена функция consonant\_count(), подсчитывающая количество гласных в тексте;
* **lines** – контейнер для строк текста, организованный в виде односвязного списка;
* **words** – контейнер для отдельных слов, организованный в виде двусвязного списка.

Необходимость выделения отдельных модулей возникала по мере разрастания и усложнения программы, а также появления новых сущностей, наличие которых в едином файле усложняло понимание прогрмаммного кода.

Например, сперва был создан основной модуль с функцией фхода в программу, затем в нём был реализован пользовательский ввод. Вводимые с консоли строки потребовалось хранить в динамической структуре данных, односвязном списке, и для него потребовался отдельный модуль, состоящий из заголовочного и основного файлов. После отладки пользовательского ввода было реализовано переупорядочивание слов в каждой строке, для чего потребовалась ещё одна динамическая структура данных, для хранения уже отдельных слов. Для удобства работы с алгоритмом быстрой сортировки был выбрал двусвязный список. Эта структура данных, вместе с обслуживающими её функциями, была также вынесена в отдельный модуль. И, наконец, функция подсчёта согласных в тексте была выненсена во вспомогательный модуль, поскольку в процессе отладки она поочерёдно была востребована и в основном модуле, и в модуле работы со строками текста.

# Описание функций

## Функции модуля main

**int get\_line( line\_t\* lines, line\_t\*\*\* ppp, char\* text )**

Принимает введённую строку текста и добавляет её в односвязный список.

**int main(int argc, char\* argv[])**

Точка входа программы. Управляет пользовательским вводом, запускает обработку данных и выводит результат на экран.

## Функции модуля utils

**int consonant\_count( const char\* text )**

Находит в строке текста количество согласных.

## Функции модуля lines

**line\_t\* add\_line( line\_t\*\* head, const char\* text )**

Добавляет новую строку в односвязный список.

**void lines\_clear( line\_t\*\* head )**

Удаляет содержимое списка строк и освобождает занимаемую им память.

**void lines\_print( line\_t\* head )**

Выводит содержимое списка на экран.

**int lines\_find( line\_t\* head, const char\* text )**

Ищет строку в списке.

**void lines\_process( line\_t\* head, const char\* delimeters )**

Разделяет слова в каждой строке и сортирует их по возрастанию количества согласных в слове.

## Функции модуля words

**void add\_word( word\_list\_t\* list, const char\* word ).**

Добавляет новое слово в двусвязный список.

**void remove\_words( word\_list\_t\* list )**

Удаляет содержимое списка слов и освобождает занимаемую им память.

**void print\_words( word\_list\_t\* list )**

Выводит содержимое списка на экран.

**void sort\_words( word\_t \*left, word\_t \*right )**

Сортирует содержимое списка согласно количеству согласных в каждом слове.

# Описание порядка работы с программой

В папке с исходными файлами программы находится Makefile, определяющий правила сборки исполняемого файла и содержащий список зависимостей. Для запуска процесса сборки достаточно набрать в терминале команду make, которая запустит компиляцию и компоновку средствами gcc, стандартного компилятора Linux. В результате будет создан исполняемый файл task.

Полный сеанс работы с программой изображён на рисунке 1.

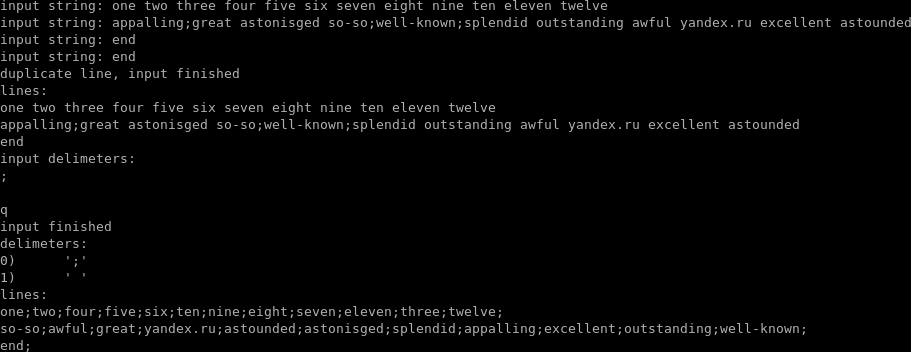


Рисунок 1 – Сеанс работы с программой в консоли

После запуска программы из командной строки, отображается приглашение на ввод строки текста. После ввода строк, описанного в п.1.2, программа предложит ввести список разделителей. В конце ввода данных, программа осуществит их обработку и вывод на экран.

# Заключение

В результате работы была спроектирована и разработана програма, позволяющая обрабатывать входные массивы строк, разделяя их на отдельные слова (токены) и сортируя в соответствии с количеством содержащихся в каждом слове согласных.

В ходе работы над проектом получены практические навыки разработки программ на языке высокого уровня C. Изучено множество аспектов разработки программ на этом языке – в первую очередь, модульная организация сложной программной системы, организация динамических структур данных и управление памятью.

На базе практического применения, закреплены базовые знания, полученные ранее, при изучении теории.

# Список литературы

1. Стивен Прата Язык программирования C. Лекции и упражнения / Стивен Прата. - М.: Вильямс, 2015. - 928 c.

2. Подбельский В.В, Фомин С.С. Программирование на языке C / М.: «Финансы и статистика». 2003.

3. Дейтел Х.М., Дейтел П.Дж. Как программировать на C / М: «БИНОМ», 2000.

4. Шилдт, Г. Справочник по C. Диасофт М.: 2000.

5. ГОСТ 2.105-95 Общие требования к текстовым документам.

# Приложение А. Исходный код программы

**main.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <sys/stat.h>

#include "words.h"

#include "lines.h"

#include "utils.h"

#define MAX\_INPUT\_SIZE 4096

static char input\_buffer[ MAX\_INPUT\_SIZE ];

static char delimeters[ MAX\_INPUT\_SIZE ];

int get\_line( line\_t\* lines, line\_t\*\*\* ppp, char\* text ) {

size\_t length = strlen( text );

// при вводе с консоли, последним символом будет \0', а предпоследним - '\n'

if( length < 1 ) {

fprintf( stderr, "input error\n" );

return 0;

}

text[ --length ] = '\0'; // удаляем '\n' (заменяем '\n' на '\0')

if( lines\_find( lines, text ) ) {

printf( "duplicate line, input finished\n" );

return 0;

}

line\_t\* line = add\_line( \*ppp, text );

\*\*ppp = line;

\*ppp = &line->next;

return 1;

}

// точка входа программы

int main( int argc, char\* argv[] )

{

// список строк

line\_t\* lines = NULL;

// вспомогательные переменные для работы со списком

line\_t\* tail = NULL; // указатель на конец (для добавления элементов)

//line\_t\* line = NULL; // указатель для работы с текущим элементом

line\_t\*\* pp = &lines; //

// количество разделителей

int delimeter\_count = 0;

#if 0

const char\* debug\_lines[] = {

"one two three four\n",

"appalling;great astonisged so-so;well-known;splendid outstanding awful yandex.ru excellent astounded\n",

};

{

int count = sizeof( debug\_lines ) / sizeof( debug\_lines[ 0 ] );

for( int i = 0; i < count; ++i ) {

strcpy( input\_buffer, debug\_lines[ i ] );

if( 0 == get\_line( lines, &pp, input\_buffer ) ) {

break;

}

}

}

#else

while( 1 ) {

printf( "input string: " );

if( fgets( input\_buffer, MAX\_INPUT\_SIZE, stdin ) ) {

if( 0 == get\_line( lines, &pp, input\_buffer ) ) {

break;

}

} else {

printf( "input error\n" );

}

} // while( 1 )

#endif

lines\_print( lines );

#if 0

delimeters[ 0 ] = ' ';

delimeters[ 1 ] = ';';

delimeter\_count = 2;

#else

printf( "input delimeters:\n" );

int sum = 0;

while( 1 ) {

int c = getchar();

if ( c != '\n' && c != '\r' ) {

if( sum > 0 && c > sum ) {

printf( "input finished\n" );

break;

}

sum += c;

delimeters[ delimeter\_count++ ] = c;

}

} // while( 1 )

#endif

delimeters[ delimeter\_count ] = '\0';

printf( "delimeters:\n" );

for( int i = 0; i < delimeter\_count; ++i ) {

printf( "%d)\t'%c'\n", i, delimeters[ i ] );

}

lines\_process( lines, delimeters );

lines\_print( lines );

return 0;

}

**utils.h**

#ifndef \_\_UTILS\_H\_\_

#define \_\_UTILS\_H\_\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// находит в строке количество согласных

int consonant\_count( const char\* text );

#endif // \_\_UTILS\_H\_\_

**utils.c**

#include "utils.h"

#include <ctype.h>

/\* 61 a

65 e

69 i

6F o

75 u

79 y \*/

// находит в строке количество согласных

int consonant\_count( const char\* text ) {

if( NULL == text )

return 0;

int result = 0;

size\_t length = strlen( text );

for( size\_t i = 0; i < length; ++i ) {

int ch = tolower( text[ i ] );

//if( 0x61 == ch || 0x65 == ch || 0x69 == ch || 0x6F == ch || 0x75 == ch || 0x79 == ch ) {

if( 0x62 == ch || 0x63 == ch || 0x64 == ch || 0x66 == ch || 0x67 == ch || 0x68 == ch || 0x6A == ch

|| 0x6B == ch || 0x6C == ch || 0x6D == ch || 0x6E == ch || 0x70 == ch || 0x71 == ch || 0x72 == ch

|| 0x73 == ch || 0x74 == ch || 0x76 == ch || 0x77 == ch || 0x78 == ch || 0x7A == ch ) {

++result;

}

}

return result;

}

**lines.h**

#ifndef \_\_LINES\_H\_\_

#define \_\_LINES\_H\_\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "words.h"

// односвязный список для хранения строк текста

// структура, представляющая строку текста в списке

typedef struct line\_t {

char\* text;

struct line\_t\* next; // указатель на следующий элемент списка

word\_list\_t words;

} line\_t;

// добавление строки в список

line\_t\* add\_line( line\_t\*\* head, const char\* text );

// освобождает список

void lines\_clear( line\_t\*\* head ) ;

// печатает список

void lines\_print( line\_t\* head );

// находит строку в списке строк

int lines\_find( line\_t\* head, const char\* text );

// упорядочение слов в строках

void lines\_process( line\_t\* head, const char\* delimeters );

#endif // \_\_LINES\_H\_\_

**lines.c**

#include "lines.h"

// добавление строки в список

line\_t\* add\_line( line\_t\*\* head, const char\* text ) {

line\_t\* line = ( line\_t\* )malloc( sizeof( line\_t ) );

bzero( line, sizeof( line\_t ) );

line->text = ( char\* )malloc( strlen( text ) + 1 );

strcpy( line->text, text );

return line;

}

// освобождает список

void lines\_clear( line\_t\*\* head ) {

line\_t\* curr = \*head;

while( curr ) {

line\_t\* next = curr->next;

free( curr );

curr = next;

}

\*head = NULL;

}

// печатает список

void lines\_print( line\_t\* head ) {

printf( "lines:\n" );

line\_t\* curr = head;

while( curr ) {

printf( "%s\n", curr->text );

curr = curr->next;

}

}

// находит строку в списке строк

int lines\_find( line\_t\* head, const char\* text ) {

line\_t\* curr = head;

while( curr ) {

if( 0 == strcmp( curr->text, text ) )

return 1;

curr = curr->next;

}

return 0;

}

// разделяет слова в каждой строке и сортирует

// их по возрастанию количества согласных в слове

void lines\_process( line\_t\* head, const char\* delimeters ) {

line\_t\* curr;

// разбиваем каждую строку на слова, имходя из списка разделителей

// отдельные слова помещаем в список для последующей сортировки

curr = head;

while( curr ) {

char\* pch = strtok( curr->text, delimeters );

while( pch != NULL ) {

add\_word( &curr->words, pch );

pch = strtok( NULL, delimeters );

}

curr = curr->next;

}

// сортируем каждый список слов и склеиваем отсортированные слова обратно в строку

curr = head;

while( curr ) {

int index = 0;

sort\_words( curr->words.head, curr->words.tail );

word\_t\* word = curr->words.head;

while( word != NULL ) {

char\* pc = word->text;

while( \*pc != '\0' ) {

curr->text[ index++ ] = \*pc;

pc++;

}

curr->text[ index++ ] = delimeters[0];

curr->text[ index ] = '\0';

word = word->next;

}

curr = curr->next;

}

}

**words.h**

#ifndef \_\_WORDS\_H\_\_

#define \_\_WORDS\_H\_\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// двусвязный список для хранения отдельных слов, встречающихся в строке текста

// элемент двусвязного списка

typedef struct word\_t

{

//int nInteger;

char\* text;

struct word\_t \*prev;

struct word\_t \*next;

} word\_t;

// двусвязный список слов

typedef struct \_word\_list\_t

{

// указатели на первый и последний элементы списка

struct word\_t \*head;

struct word\_t \*tail;

} word\_list\_t;

// добавление в список

void add\_word( word\_list\_t\* list, const char\* p );

// удаление

void remove\_words( word\_list\_t\* list );

// вывод содержимого списка в консоль

void print\_words( word\_list\_t\* list );

// быстрая сортировка (quick sort)

void sort\_words( word\_t \*left, word\_t \*right );

#endif // \_\_WORDS\_H\_\_

**words.c**

#include "words.h"

#include "utils.h"

// добавление в список

void add\_word( word\_list\_t\* list, const char\* word ) {

if( word == NULL )

return;

word\_t \*item = ( word\_t\* )malloc( sizeof( word\_t ) );

item->text = ( char\* )malloc( strlen( word ) );

strcpy( item->text, word );

if ( list->head == NULL ) {

list->head = list->tail = item;

item->next = item->prev = NULL;

} else {

list->tail->next = item;

item->prev = list->tail;

list->tail = item;

item->next = NULL;

}

}

// удаление

void remove\_words( word\_list\_t\* list ) {

word\_t \*temp, \*item = list->head;

while( item != NULL ) {

if( item->text != NULL )

free( item->text );

temp = item;

item = item->next;

free( temp );

}

list->head = list->tail = NULL;

}

// вывод содержимого списка в консоль

void print\_words( word\_list\_t\* list ) {

word\_t \*item = list->head;

while ( item != NULL ) {

printf( "%s ", item->text );

item = item->next;

}

printf( "\n" );

}

// быстрая сортировка (quick sort)

void sort\_words( word\_t \*left, word\_t \*right ) {

word\_t \*start;

word\_t \*curr;

char\* p;

// сортировка окончена - выход

if ( left == right )

return;

// установка двух рабочих указателей - Start и Current

start = left;

curr = start->next;

// итерация по списку слева направо

while ( 1 ) {

// элемент с максимальным значением помещается в начало списка

if ( consonant\_count( start->text ) < consonant\_count( curr->text ) ) {

p = curr->text;

curr->text = start->text;

start->text = p;

}

if ( curr == right )

break;

curr = curr->next;

}

// переключение First и Current - максимум попадает в правый конец списка

p = left->text;

left->text = curr->text;

curr->text = p;

// сохранение Current

word\_t\* temp = curr;

// рекурсия

curr = curr->prev;

if( curr != NULL ) {

if ( ( left->prev != curr ) && ( curr->next != left ) )

sort\_words( left, curr );

}

curr = temp;

curr = curr->next;

if( curr != NULL ) {

if ( ( curr->prev != right ) && ( right->next != curr ) )

sort\_words( curr, right );

}

}