МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра «Информационные системы»

**отчет**

по практической работе №4

по дисциплине «Программирование»

Тема: "Текстовые строки как массивы символов"

|  |  |  |
| --- | --- | --- |
| Студент гр. 0323 |  | Голубцов В.В. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

Цель работы.

Практические навыки работы с строками как массивом символов.

Основные теоретические положения.

Текстовые строки представляются с помощью одномерных массивов символов. В языке C++ текстовая строка представляет собой набор символов, обязательно заканчивающийся нулевым символом (‘\0’). Поэтому, если вы хотите создать текстовый массив для хранения 10 (N) символов, нужно выделить память под 11(N+1) символов.

Объявленный таким образом массив может использоваться для хранения текстовых строк, содержащих не более 10 символов. Нулевой символ позволяет определить границу между содержащимся в строке текстом и неиспользованной частью строки.

При определении строковых переменных их можно инициализировать конкретными значениями с помощью строковых литералов:

char S1[15] = “This is text”;

char S2[] = “Пример текста”;

Последние два элемента переменной  просто не используются, а строка  автоматически подстраивается под длину инициализирующего текста.

При работе со строками можно обращаться к отдельным символам строки как в обычном одномерном массиве с помощью индексов:

cout << S1[0]; // На экране будет выведен символ ‘T’

Если строка формируется при помощи цикла (или иного способа), то необходимо в ее конец обязательно записать нулевой символ '\0'.

При выводе строк можно использовать форматирование (манипуляторы или функции потока вывода). Вывод текстовых строк на экран крайне простая задача:

char Str[21] = “Это пример текста”;

cout  <<  Str << endl;

cout  <<  “Это текстовый литерал.” << endl;

Ввод текста с клавиатуры можно осуществлять разными способами, каждый из которых имеет определенные особенности.

Непосредственное чтение текстовых строк из потока вывода осуществляется до первого знака пробела.

Такой способ чтения обеспечивает ввод символов до первого пробельного символа (не до конца строки). Остальные символы введенного с клавиатуры остаются в потоке ввода и могут быть прочитаны из него следующими операторами >>.

Для того чтобы прочесть всю строку полностью, можно воспользоваться одной из функций gets или gets\_s (для этого в программу должен быть включен заголовочный файл <stdio.h>).

Функция gets имеет один параметр, соответствующий массиву символов, в который осуществляется чтение. Вторая функция (gets\_s) имеет второй параметр, задающий максимальную длину массива символов .

Ввод текста, длина которого (вместе с нулевым символом) превышает значение второго параметра (то есть длины символьного массива ), приводит к возникновению ошибки при выполнении программы

Предпочтительно использование функции потока ввода cin.getline:

const int N = 21;

char Str [N];

cin.getline (Str, N);      // Пусть введена строка “Это пример текста”

cout  <<  Str << endl;  // На экран будет выведено “ Это пример текста”

Если длина введенного с клавиатуры текста превышает максимальную длину массива , в него будет записано (в нашем примере) 20 символов вводимого текста и нулевой символ. Остальные символы введенного текста остаются во входном потоке и могут быть взяты из него следующими инструкциями ввода.

Функция cin.getline может иметь третий параметр, задающий символ, при встрече которого чтение строки из потока прекращается:

cin.getline (Str, N,  ‘.’);

Иногда чтение из потока невозможно (например, попытка считать слишком длинный текст). Для того чтобы продолжить чтение из потока, необходимо восстановить его нормальное состояние. Этого можно достигнуть с помощью функции потока cin.clear(), которая сбрасывает состояние потока в нормальное. Если забирать остатки данных из потока ввода не надо, то следует очистить его с помощью функции cin.sync().

При обработке текстовых строк обычно используется  набор типовых операций, к которым можно отнести:

·         определение фактической длины текста записанного в символьный массив;

·         копирование текста из одной строки в другую;

·         объединение двух строк;

·         лексикографическое сравнение строк – в алфавитном порядке (больше, меньше, равно)

и др.

Класс string предназначен для работы со строками типа char, которые представляют собой строчку с завершающим нулем (символ ‘\0’). Класс string был введен как альтернативный вариант для работы со строками типа char.

Чтобы использовать возможности класса string, нужно подключить библиотеку <string> и пространство имен std. Объявление же переменной типа string осуществляется схоже с обычной переменной:

string S1; // Переменная с именем s1 типа string

string S2 = “Пример”; // объявление с инициализацией

Создание нового типа string было обусловлено недостатками работы с строками символов, который показывал тип char. В сравнении с этим типом string имеет ряд основных преимуществ:

·        возможность использования для обработки строк стандартные операторы С++(=,+,<,==,>,+=,!=,<=,>=,[])(=,+,<,==,>,+=,!=,<=,>=,[]). Использование типа char приводило требовало написание чрезмерного программного кода;

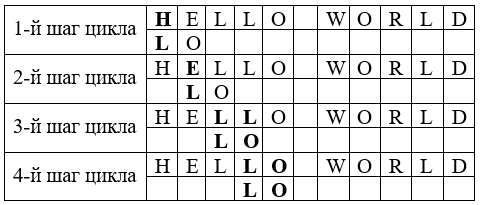
·        обеспечение лучшей надежности программного кода;

·        обеспечение строки, как самостоятельного типа данных.

**Поиск подстроки в строке.**

При работе со строками часто будет возникать потребность в поиске набора символа или слов (поиска подстроки в строке). При условии, что текст может быть крайне большим, хочется, чтобы алгоритм поиска подстроки работал быстро.

Самый простой способ подстроки в строке – Линейный поиск – циклическое сравнение всех символов строки с подстрокой. Действительно, этот способ первый приходит в голову, но очевидно, что он будет самым долгим.



На первых двух итерациях цикла сравниваемые буквы не будут совпадать. На третьей же итерации, совпал символ ‘L’, это означает, что теперь нужно сравнивать следующий символ подстроки со следующим символом строки. Видно, что символы отличаются, поэтому алгоритм продолжает свою работу. На четвертой же итерации подстрока была найдена.

Если представить, что исходная строка непорядок больше и подстрока находится в конце строки (или вовсе отсутствует), то сразу видны минусы данного алгоритма.

Одним из самых популярных алгоритмов, который работает быстрее, чем приведенный выше алгоритм, является алгоритм Кнута-Морриса-Пратта (КМП). Идея заключается в том, что не нужно проходить и сравнивать абсолютно все символы строки, если известны символы, которые есть и в строке, и в подстроке.

Суть алгоритма: дана подстрока S и строка T. Требуется определить индекс, начиная с которого образец S содержится в строке T. Если S не содержится в T, необходимо вернуть индекс, который не может быть интерпретирован как позиция в строке.



Алгоритм Бойера-Мура в отличие от КМП полностью не зависим и не требует заранее проходить по строке. Этот алгоритм считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке.

Преимущество этого алгоритма в том, что ценной некоторого количества предварительных вычислений над подстрокой (но не над исходной строкой, в которой ведётся поиск), подстрока сравнивается с исходным текстом не во всех позициях (пропускаются позиции, которые точно не дадут положительный результат).

Поиск подстроки ускоряется благодаря созданию таблиц сдвигов. Сравнение подстроки со строки начинается с последнего символа подстроки, а затем происходит прыжок, длина которого определяется по таблице сдвигов. Таблица сдвигов строится по подстроке так чтобы перепрыгнуть максимальное количество символов строки и не пропустить вхождение подстроки в строку.

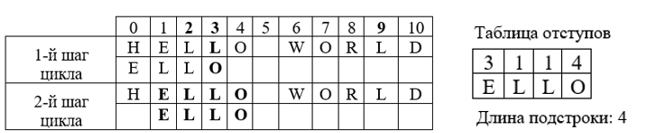
Правила построения таблицы сдвигов:

1)    Значение элемента таблицы равно удаленности соответствующего символа от конца шаблона (подстроки).

2)    Если символ встречается более одного раза, то применятся значение, соответствующее символу, наиболее близкому к концу шаблона.

3)    Если символ в конце шаблона встречается 1 раз, ему соответствует значение, равное длине образа; если более одного раза – значение, соответствующее символу, наиболее близкому к концу образа.

4)    Для символов, отсутствующих в образе, применяется значение, равное длине шаблона.



Сначала была построена таблица отступов и подсчитана длина подстроки. Затем начинается алгоритм поиска подстроки в строке. Сравнивает символ ‘L’ строки и ‘O’ подстроки. Элементы не совпадают, поэтому необходимо определить длину отступа. Символ ‘L’ присутствует в таблице отступа, длина отступа равняется 1. Подстрока смещается на 1 символ вперед. На следующей итерации подстрока найдена.

Постановка задачи.

Необходимо написать программу, которая реализует поставленную задачу:

1)    С клавиатуры или с файла (\*) (пользователь сам может выбрать способ ввода) вводится последовательность, содержащая от 1 до 50 слов, в каждом из которых от 1 до 10 строчных латинских букв и цифр. Между соседними словами произвольное количество пробелов. За последним символом стоит точка.

2)    Необходимо отредактировать входной текст:

·        удалить лишние пробелы;

·        удалить лишние знаки препинания (под «лишними» подразумевается несколько подряд идущих знаков (обратите внимание, что «…» - корректное использование знака) в тексте);

·        исправить регистр букв, если это требуется (пример некорректного использования регистра букв: пРиМЕр);

3) Выполнить задание по варианту:

|  |  |
| --- | --- |
| 1 | Вывести на экран слова последовательности в обратном порядке. |
| 2 | Вывести на экран слова последовательности в алфавитном порядке. |
| 3 | Вывести на экран слова последовательности, не содержащие цифр. |
| 4 | Вывести на экран только те слова последовательности, в которых встречаются одинаковые буквы. |
| 5 | Вывести на экран только те слова последовательности, в которых первая буква слова встречается в этом слове еще раз. |
| 6 | После окончания ввода последовательности вывести на экран сначала все слова, содержащие только буквы, затем слова, содержащие только цифры, а потом слова, содержащие и буквы, и цифры. |

4) Выполнить задание по варианту:

|  |  |
| --- | --- |
| 1 | Вывести на экран ту же последовательность, заменив во всех словах первую букву соответствующей прописной буквой. |
| 2 | Вывести на экран количество символов в каждом слове исходной последовательности. |
| 3 | Вывести на экран ту же последовательность, удалив из всех слов заданный набор букв и (или) цифр. |
| 4 | Вывести на экран ту же последовательность, заменив во всех словах цифры на буквы латинского алфавита, номера которых в алфавите равны заменяемой цифре. |
| 5 | Вывести на экран ту же последовательность, переместив все цифры, содержащиеся в словах, в конец соответствующих слов. |
| 6 | Вывести все слова исходной последовательности на экран вертикально. |
| 7 | Вывести на экран  все слова последовательности в две или три колонки (в зависимости от количества слов) с выравниванием слов по правой границе колонки. |

5)  Необходимо найти подстроку, которую введёт пользователь в имеющейся строке. Реализуйте два алгоритма: первый алгоритма – Линейный поиск, а второй алгоритм согласно вашему номеру в списке. Четные номера должны реализовать алгоритм КНМ, а нечетные – Бойера-Мура. (\*)

Выполнение работы.

В начале программы пользователю дается выбор чтения текста для его последующего редактирования. По задаче должно быть реализовано два способа: ввод текста вручную и чтение текста из файла.

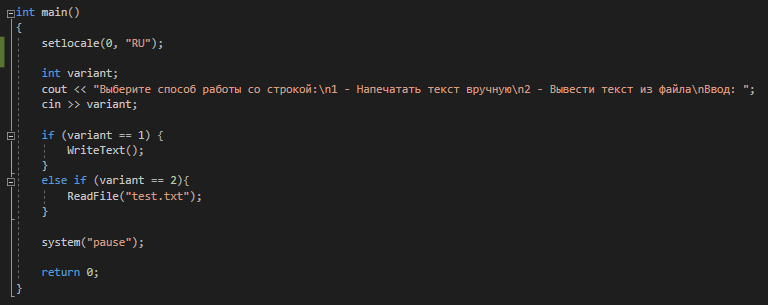
В первом варианте пользователю дается возможность ввести текст вручную с помощью cin.getline().

Рисунок : выбор чтения текста

Во втором варианте программа создает поток файла, открывает файл по его имени, делает проверку на то удается ли открыть и записывает в char наш текст.

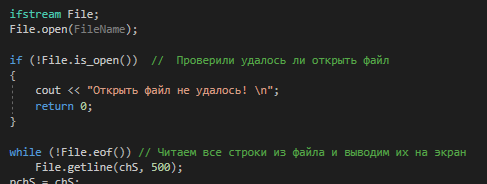


Рисунок :чтение из файла

Далее операции форматирования строки одинаковы для двух способов введенного текста.

Первым делом мы удаляем из строки все повторяющиеся символы (за исключением букв) с помощью функции *RemovingDuplicateCharacters*.

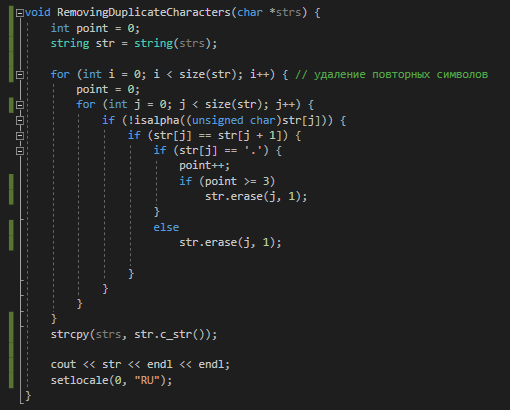


Рисунок 3: удаление повторояющихся элементов

В функцию передается строка. Преобразовываем ее в string. В цикле идет проверка на то является ли элемент буквой, если нет, значит сравниваем его с следующим элементом и если они равны, мы удаляем текущий элемент.

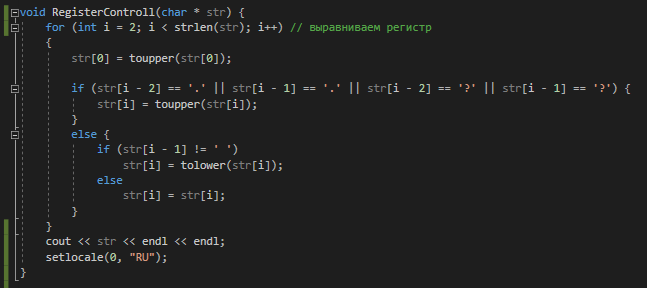
Далее вызывается функция исправления регистра строки, в которую так же передается текст.

Рисунок : исправление регистра букв

Все операции выполняются с помощью встроенной функции *toupper/tolower.* Первый символ всегда является заглавной буквой. Далее в цикле идет проверка предыдущих символов на то является ли текущий символ новым предложением. Если да, то делаем букву заглавной. Если нет, то идет проверка на то, является ли слово изначально с большой буквы, допустим чтобы не допустить ошибку в словах имени, городов и т.д. Все остальные буквы изменяем на строчные.

Т.к. у меня 5 вариант мне необходимо было сделать функцию, которая выводит на экран только те слова, в которых первая буква слова встречается в нем еще минимум раз.

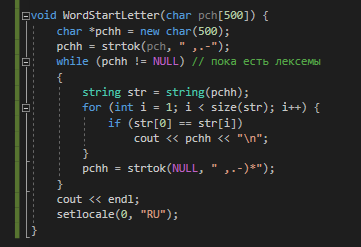


Рисунок : Вывод слов в которых первая буква встречается еще раз

В начале форматируем строку в массив слов(лексемы). Далее для каждого слова идет проверка на то, есть ли в слове повторяющиеся буквы. После завершения проверки удаляем слово из массива и идет новая проверка следующего слова.

Следующая функция ставит цифры в конец слова, если таковые есть и они стоят не в конце.

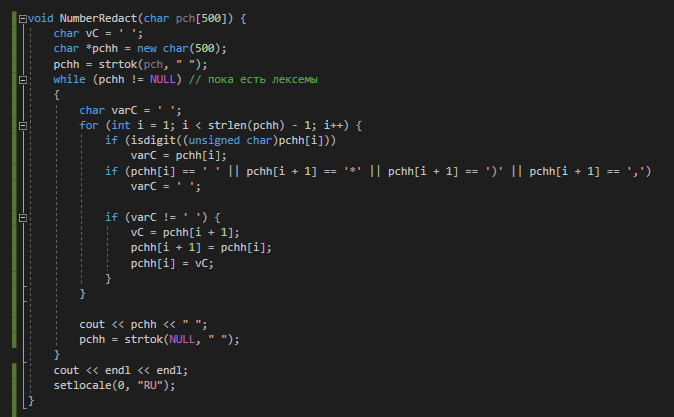
В начале преобразует строку в массив слов. В цикле проверка: если элемент цифра, то varC равна этой цифре. Пока след. элемент не символ ( \*),), элемент не перезаписывается, иначе мы его удаляем. Пока в varC не пустой, заменяем след. элемент на текущий и таким образом доходим до конца слова.

Рисунок : перемещение цифр в конец слова

*В программе используется частая смена кодировки из-за проблем с выводом русского текста в редактор, с английским текстом и без таких присвоений разных кодировок работает хорошо*.

Выводы.

С текстовыми строками, представленными в виде массива символов, достаточно легко выполнять различные преобразования.

Полный код программы

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

#include <vector>

using namespace std;

void RemovingDuplicateCharacters(char \*strs) {

int point = 0;

string str = string(strs);

for (int i = 0; i < size(str); i++) { // удаление повторных символов

point = 0;

for (int j = 0; j < size(str); j++) {

if (!isalpha((unsigned char)str[j])) {

if (str[j] == str[j + 1]) {

if (str[j] == '.') {

point++;

if (point >= 3)

str.erase(j, 1);

}

else

str.erase(j, 1);

}

}

}

}

strcpy(strs, str.c\_str());

cout << str << endl << endl;

setlocale(0, "RU");

}

void RegisterControll(char \* str) {

for (int i = 2; i < strlen(str); i++) // выравниваем регистр

{

str[0] = toupper(str[0]);

if (str[i - 2] == '.' || str[i - 1] == '.' || str[i - 2] == '?' || str[i - 1] == '?') {

str[i] = toupper(str[i]);

}

else {

if (str[i - 1] != ' ')

str[i] = tolower(str[i]);

else

str[i] = str[i];

}

}

cout << str << endl << endl;

setlocale(0, "RU");

}

void WordStartLetter(char pch[500]) {

char \*pchh = new char(500);

pchh = strtok(pch, " ,.-");

while (pchh != NULL) // пока есть лексемы

{

string str = string(pchh);

for (int i = 1; i < size(str); i++) {

if (str[0] == str[i])

cout << pchh << "\n";

}

pchh = strtok(NULL, " ,.-)\*");

}

cout << endl;

setlocale(0, "RU");

}

void NumberRedact(char pch[500]) {

char vC = ' ';

char \*pchh = new char(500);

pchh = strtok(pch, " ");

while (pchh != NULL) // пока есть лексемы

{

char varC = ' ';

for (int i = 1; i < strlen(pchh) - 1; i++) {

if (isdigit((unsigned char)pchh[i]))

varC = pchh[i];

if (pchh[i] == ' ' || pchh[i + 1] == '\*' || pchh[i + 1] == ')' || pchh[i + 1] == ',')

varC = ' ';

if (varC != ' ') {

vC = pchh[i + 1];

pchh[i + 1] = pchh[i];

pchh[i] = vC;

}

}

cout << pchh << " ";

pchh = strtok(NULL, " ");

}

cout << endl << endl;

setlocale(0, "RU");

}

bool ReadFile(const char \*FileName)

{

char chS[400];

char copyStr[500];

char \*pchS;

ifstream File;

File.open(FileName);

if (!File.is\_open()) // Проверили удалось ли открыть файл

{

cout << "Открыть файл не удалось! \n";

return 0;

}

while (!File.eof()) // Читаем все строки из файла и выводим их на экран

File.getline(chS, 500);

pchS = chS;

cout << "Исходный текст:\n" << pchS << endl << endl;

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\n\*\*\*В этой строке удалены все повторные символы\*\*\*\n";

RemovingDuplicateCharacters(pchS); // удаление повторных символов

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\n\*\*\*В этой строке выравнен регистр букв\*\*\*\n";

RegisterControll(pchS); // выравниваем регистр

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\n\*\*\*Слова, в которых есть буквы такие же как и первая буква слова\*\*\*\n";

strcpy(copyStr, pchS);

WordStartLetter(copyStr); // выводим слова, в которых есть буквы такие же как и первая буква слова

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\n\*\*\*В этой все буквы в словах передвинуты в конец слова\*\*\*\n";

NumberRedact(pchS); // перемещаем буквы в словах в конец слова

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

File.close(); // Закрываем файл

return 1;

}

void WriteText() {

cin.get();

char chS[500];

char copyStr[500];

char \*pchS;

cout << "Введите текст содержащий от 1 до 50 слов, в каждом из которых от 1 до 10 строчных латинских букв и цифр. Между соседними словами произвольное количество пробелов. За последним символом стоит точка.\nВвод: ";

cin.getline(chS, 500);

pchS = chS;

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\n\*\*\*В этой строке удалены все повторные символы\*\*\*\n";

setlocale(0, ".866");

RemovingDuplicateCharacters(pchS); // удаление повторных символов

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\n\*\*\*В этой строке выравнен регистр букв\*\*\*\n";

setlocale(0, ".866");

RegisterControll(pchS); // выравниваем регистр

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\n\*\*\*Слова, в которых есть буквы такие же как и первая буква слова\*\*\*\n";

strcpy(copyStr, pchS);

setlocale(0, ".866");

WordStartLetter(copyStr); // выводим слова, в которых есть буквы такие же как и первая буква слова

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\n\*\*\*В этой строке все буквы в словах передвинуты в конец слова\*\*\*\n";

setlocale(0, ".866");

NumberRedact(pchS); // перемещаем буквы в словах в конец слова

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

}

int main()

{

setlocale(0, "RU");

int variant;

cout << "Выберите способ работы со строкой:\n1 - Напечатать текст вручную\n2 - Вывести текст из файла\nВвод: ";

cin >> variant;

if (variant == 1) {

WriteText();

}

else if (variant == 2){

ReadFile("test.txt");

}

system("pause");

return 0;

}

Приложение Б

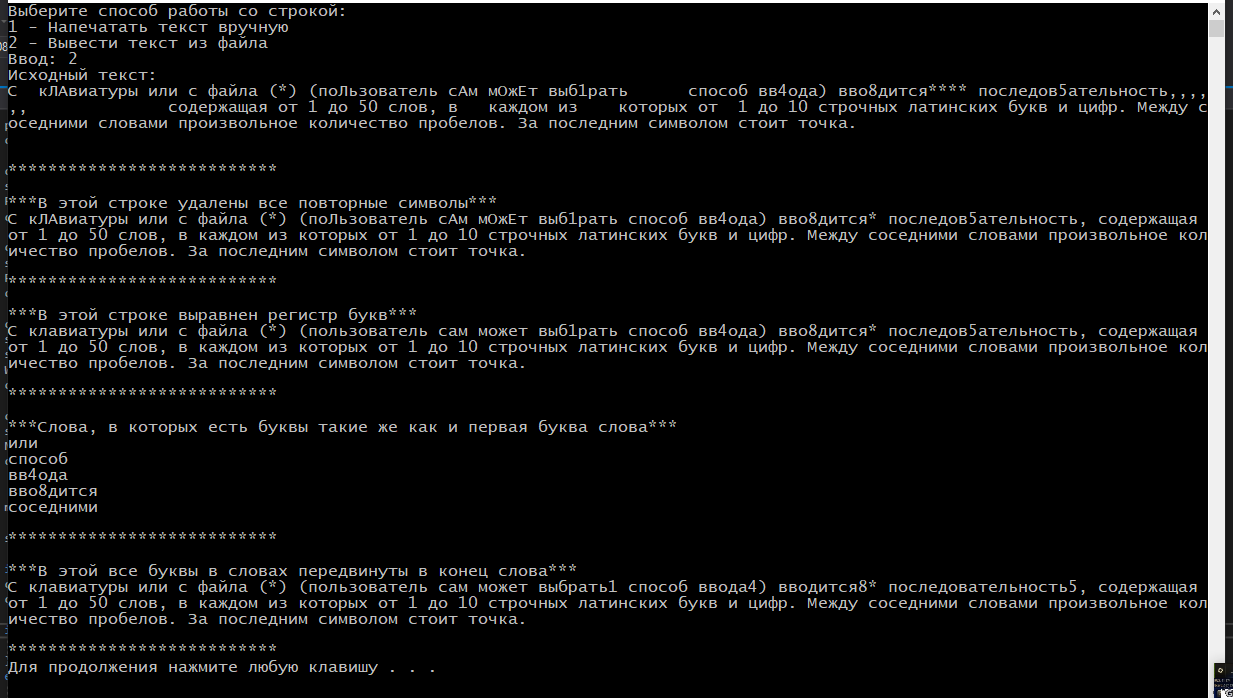
****ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

Рисунок : полный цикл выполнения программы