**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №4**

**по дисциплине «Программирование»**

Тема: **ТЕКСТОВЫЕ СТРОКИ. ФАЙЛЫ.**

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. | Цатинян А.А. |  |
| Преподаватель | Глущенко А. Г. |  |

Санкт-Петербург

2022

**Цель работы.**

Изучение навыков работами со строками и файлами.

**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. 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. 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. 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(). 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.

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.

Класс string обладает широким функционалом:

·        функция compare() сравнивает одну часть строки с другой;

·        функция length() определяет длину строки;

·        функции find() и rfind() служат для поиска подстроки в строке (отличаются функции лишь направлением поиска);

·        функция erase() служит для удаления символов;

·        функция replace() выполняет замену символов;

·        функция insert() необходима, чтобы вставить одну строку в заданную позицию другой строки;

Но весь функционал string накладывает и свой негативный отпечаток. Основным недостатком string в сравнении с типом char является замедленная скорость обработки данных.

**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 все подстроки, 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 КМП, а нечетные – Бойера-Мура. (\*)

**Выполнение работы.**

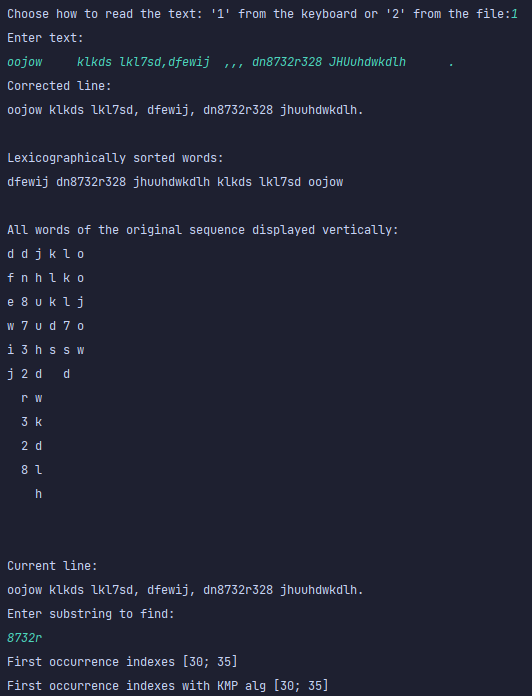
Код программы представлен в приложении А.

Блок описания кода и использованных алгоритмов

Используя библиотеку sstream подстроки пословно выводятся в другую переменную и сохраняются, тем самым мы избавляемся от лишних пробелов. При удалении лишних символов проверяется правильная расстановка пробелов.

Для сортировки слов используется стандартный метод sort, который применяется к объекту типа vector<string>. Так как вектор заполнен объектами типа string, то дополнительных методов сравнения писать не нужно, т.к. в c++ сравнение объектов типа string происходи лексикографически сразу. Для изменения регистра слов используется стандартный метод transform, в который передаётся лямбда функция, которая возвращает символ в нижнем регистре, используя стандартный метод tolower(char).

Блок скриншотов работы программы



**Выводы.**

Изучен навык работы с со строками, объектами типа string. Изучены алгоритмы поиска подстрок используя линейный поиск и алгоритм КМП. Изучен навык работы с файлами, чтением данных из файлов.

Приложение А

рабочий код

#include <iostream>  
#include <fstream>  
#include <sstream>  
#include <vector>  
*/\*  
 \* Test input:  
 \* oojow klkds lkl7sd,dfewij ,,, dn8732r328 JHUuhdwkdlh .  
 \* \*/  
using namespace* std;  
  
*bool* isEmpty(string &);  
  
*bool* isPMark(*char* &);  
  
*int* main() {  
 *int* option;  
 cout << "Choose how to read the text: '1' from the keyboard or '2' from the file:";  
 cin >> option;  
 string text;  
 *if* (option == 1) {  
 cout << "Enter text:" << endl;  
 getline(cin, text);  
 getline(cin, text);  
 } *else* {  
 cout << "Enter path:" << endl;  
 string path;  
 cin >> path;  
 ifstream file(path);  
 *while* (!file.is\_open()) {  
 cout << "Wrong path or such file does not exist, try again" << endl;  
 cin >> path;  
 file.open(path);  
 }  
 getline(file, text);  
 }  
  
 string t, res;  
 vector<string> words;  
 stringstream(text) >> t;  
 *while* (!isEmpty(text)) {  
 text.replace(text.find(t), t.size(), "");  
 transform(t.begin(), t.end(), t.begin(), [](*unsigned char* c) { *return* std::tolower(c); });  
 res += t + " ";  
 stringstream(text) >> t;  
 }  
 *for* (*int* i = 0; i < res.size(); ++i) {  
 *if* (isPMark(res[i])) {  
 *if* (res[i] == '.' && i == res.size() - 3 && res[i + 1] == '.' && res[i + 2] == '.') {  
 *break*;  
 } *else if* (res[i] == '.' && i < res.size() - 3 && res[i + 1] == '.' && res[i + 2] == '.' &&  
 res[i + 3] != '.') {  
 i += 2;  
 *continue*;  
 }  
 *int* j = i + 1;  
 *while* (j != res.size() && isPMark(res[j])) res.erase(res.begin() + j);  
 }  
 }  
 *for* (*int* i = 0; i < res.size(); ++i) {  
 *if* (res[i] == ' ' && isPMark(res[i + 1])) {  
 res.erase(res.begin() + i);  
 i--;  
 }  
 *if* (isPMark(res[i]) && i + 1 != res.size() && res[i + 1] != ' ')  
 res = res.substr(0, i + 1) + " " + res.substr(i + 1, res.size() - i);  
 }  
 text = res;  
 cout << "Corrected line:" << endl << text << endl << endl;  
  
 string textCopy = text;  
 stringstream(textCopy) >> t;  
 *while* (!isEmpty(textCopy)) {  
 textCopy.replace(textCopy.find(t), t.size(), "");  
 transform(t.begin(), t.end(), t.begin(), [](*unsigned char* c) { *return* std::tolower(c); });  
 *if* (!isPMark(t[0])) words.push\_back(isPMark(t[t.size() - 1]) ? t.substr(0, t.size() - 1) : t);  
 stringstream(textCopy) >> t;  
 }  
 sort(words.begin(), words.end());  
 cout << "Lexicographically sorted words:" << endl;  
 *for* (string &str : words) cout << str << " ";  
  
 cout << endl << endl << "All words of the original sequence displayed vertically:" << endl;  
 *int* maxLength = 0;  
 *for* (string &str : words) *if* (str.size() > maxLength) maxLength = str.size();  
 *for* (string &str : words) *while* (str.size() != maxLength) str += ' ';  
 *for* (*int* i = 0; i < maxLength; ++i) {  
 *for* (string &str : words) cout << str[i] << ' ';  
 cout << endl;  
 }  
  
 cout << endl << endl << "Current line:" << endl << text << endl << "Enter substring to find:" << endl;  
 string substring;  
 cin >> substring;  
 *for* (*int* i = 0; i < text.size() - substring.size() + 1; ++i) {  
 *if* (text.substr(i, substring.size()) == substring) {  
 cout << "First occurrence indexes [" << i << "; " << i + substring.size() << "]" << endl;  
 *break*;  
 }  
 }  
 *int* \*prefTable = *new int*[substring.size()];  
 *for* (*int* j = 0; j < substring.size(); ++j) prefTable[j] = 0;  
 *for* (*int* j = 2; j <= substring.size(); ++j) {  
 string subSubstr = substring.substr(0, j);  
 *int* maxPrefLength = 0;  
 *for* (*int* k = 1; k < subSubstr.size(); ++k) {  
 string prefix = subSubstr.substr(0, k), suffix = subSubstr.substr(subSubstr.size() - k, k);  
 *if* (prefix == suffix && k > maxPrefLength) maxPrefLength = k;  
 }  
 prefTable[j - 1] = maxPrefLength;  
 }  
 *int* i = 0;  
 *for* (*int* j = 0; j < substring.size(); ++j) {  
 *if* (substring[j] != text[i + j]) {  
 *if* (j == 0) {  
 i++;  
 j--;  
 *continue*;  
 } *else* {  
 i += j - prefTable[j - 1];  
 j = prefTable[j - 1] - 1;  
 }  
 }  
 }  
 cout << "First occurrence indexes with KMP alg [" << i << "; " << i + substring.size() << "]" << endl;  
}  
  
*bool* isEmpty(string &str) {  
 *for* (*char* &ch: str) *if* (ch != ' ') *return false*;  
 *return true*;  
}  
  
*bool* isPMark(*char* &ch) {  
 *if* ((ch >= 48 && ch <= 57) || (ch >= 65 && ch <= 90) || (ch >= 97 && ch <= 122) || ch == 32) *return false*;  
 *return true*;  
}