МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра «Информационные системы»

отчет

по лабораторной работе №1

по дисциплине «Программирование»

Тема: "Текстовые строки как массивы символов"

|  |  |  |
| --- | --- | --- |
| Студент гр. 0323 |  | Землянский Д.А. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

Цель работы.

Научиться определять и работать с текстовыми строками.

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

**Ввод с клавиатуры**

При выводе строк можно использовать форматирование (манипуляторы или функции потока вывода).

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

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.

Постановка задачи.

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 КНМ, а нечетные – Бойера-Мура. (\*)

Приложение А

Полный код программы

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <list>

#include <vector>

#include <functional>

#include <cctype>

#include<string>

std::list<std::string> ReadStream(std::istream& source)

{

std::list<std::string> ret;

while (!source.eof())

{

std::string word;

source >> word;

if (!word.empty())

{

ret.push\_back(word);

if (word.c\_str()[word.length() - 1] == '.')

{

break;

}

}

}

return ret;

}

void PrintWords(const std::string& title, const std::list<std::string>& words)

{

std::cout << std::endl << "-------" << title << "-------" << std::endl;

for (const auto& word : words)

{

std::cout << word << ' ';

}

std::cout << std::endl;

}

std::string InputSubString()

{

std::cout << std::endl << "Enter a substring to search" << std::endl;

std::string ret;

std::cin >> ret;

return ret;

}

void Task2EditText(std::list<std::string>& words)

{

for (auto& word : words)

{

//масив символов, составляющих исследуемое слово

char\* wordBuffer = new char[word.size() + 1];

std::strcpy(wordBuffer, word.c\_str());

//знак препинания, который мы обнаружили в предыдущем символе

char punktMark{ '\0' };

//количество подряд идущих точек

int dotCount{ 0 };

//просмотрим каждую букву

for (char\* letter = wordBuffer; \*letter != '\0'; letter++)

{

if (std::isalnum(\*letter))

{

//это буква или цифра, а не знак препинания

//если это не первый символ, то сделаем букву строчной

if (letter != wordBuffer && std::isalpha(\*letter))

{

\*letter = std::tolower(\*letter);

}

//посмотрим, не было ли перед этой буквой неправильного числа точек

if (punktMark == '.')

{

if (dotCount ==2)

{

//убираем одну лишнюю точку

strcpy(letter, letter + 1);

//чтобы не пропустить новый символ в проверке

--letter;

}

else if(dotCount > 3)

{

//убираем commaCount - 3 лишних точек

strcpy(letter, letter + dotCount - 3);

--letter;

}

}

//забыли про предыдущий знак препианния

punktMark = '\0';

dotCount = 0;

}

else

{

//это не буква и не цифра, а значит - знак препинания

//посмотрим, не было ли перед ним другого знака

if (punktMark == '\0')

{

//первый знак препинания - запомним его

punktMark = \*letter;

if (punktMark == '.')

{

++dotCount;

}

}

else

{

//знак препинания не первый - посмотрим, не с точкой ли мы имеем дело

if (punktMark == '.' && \*letter == '.')

{

//если это точка, то просто посчитаем ее

++dotCount;

}

else

{

//текущий знак - точно лишний, удалим

strcpy(letter, letter + 1);

--letter;

}

}

}

}

//если все закончилось точками, то нужно снова проверить, верно ли их количество

if (punktMark == '.')

{

if (dotCount == 2)

{

//убираем одну лишнюю точку в конце

wordBuffer[std::strlen(wordBuffer) - 1] = '\0';

}

else if (dotCount > 3)

{

//убираем commaCount - 3 лишних точек в конце

wordBuffer[std::strlen(wordBuffer) - (dotCount - 3)] = '\0';

}

}

//запомним отредактированное слово

word = wordBuffer;

//удалим не нужный более временный буфер

delete[] wordBuffer;

}

}

void Task3Variant5(const std::list<std::string>& words)

{

std::cout << std::endl << "--------Words, where the first letter is repetitive again-------" << std::endl;

int count{ 0 };

for (auto& word : words)

{

bool shallBePrinted{ false };

auto buffer{ word.c\_str() };

auto first{ \*buffer };

for (auto letter = buffer + 1; \*letter != '\0'; letter++)

{

if (\*letter == first)

{

shallBePrinted = true;

break;

}

}

if (shallBePrinted)

{

std::cout << word << std::endl;

++count;

}

}

std::cout << "------Total " << count << " words found-----" << std::endl;

}

void Task4Variant4(std::list<std::string> words)

{

for (auto& word : words)

{

//масив символов, составляющих исследуемое слово

char\* wordBuffer = new char[word.size() + 1];

std::strcpy(wordBuffer, word.c\_str());

for (char\* letter = wordBuffer; \*letter != '\0'; letter++)

{

//не цифру ли мы повстречали?

if (std::isdigit(\*letter))

{

//заменаем цифру буквой

//будем считать, что буквы нум еруются с нуля, тогда 0 заменяем на A, 1 - на B, и так далее

\*letter = 'A' + \*letter - '0';

}

}

//запомним отредактированное слово

word = wordBuffer;

//удалим не нужный более временный буфер

delete[] wordBuffer;

}

//выведем отредактированную последовательность

PrintWords("Words, where digits are replaced with letters", words);

}

void Task5Search(const std::list<std::string>& words, std::function<bool(const std::string& source, const std::string& key)> searchAlg)

{

const auto key = InputSubString();

std::list<std::string> found;

for (const auto& word : words)

{

if(word.size() >= key.size())

{

if(searchAlg(word, key))

{

found.push\_back(word);

}

}

}

PrintWords(std::string("Words containing key '") + key + "'", found);

}

bool LineSearch(const std::string& source, const std::string& key)

{

for (const char\* s = source.c\_str(); \*s != '\0'; s++)

{

for (int j = 0; ; j++)

{

if (!key[j])

return true;

if (\*(s + j) == '\0' || \*(s + j) != key[j])

break;

}

}

return false;

}

std::vector<size\_t> PrefixFunction(std::string s)

{

auto n = s.length();

std::vector<decltype(n)> prefixLen(n);

for (decltype(n) i = 1; i < n; ++i)

{

auto j = prefixLen[i - 1];

while ((j > 0) && (s[i] != s[j]))

j = prefixLen[j - 1];

if (s[i] == s[j])

++j;

prefixLen[i] = j;

}

return prefixLen;

}

bool KMPSearch(const std::string& source, const std::string& key)

{

auto prefixLen = PrefixFunction(key + ' ' + source);

for (auto p : prefixLen)

{

if (p == key.size())

return true;

}

return false;

}

void Laba4(std::list<std::string> text)

{

PrintWords("Source text", text);

Task2EditText(text);

PrintWords("Extra Punctuation mark are removed; register is fixed", text);

Task3Variant5(text);

Task4Variant4(text);

Task5Search(text, &LineSearch);

Task5Search(text, &KMPSearch);

}

Приложение Б

ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ



