**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Информационных систем**

отчет

**по лабораторной работе №3**

**по дисциплине «Программирование»**

Тема: Тип данных string. Методы поиска подстроки в строке

|  |  |  |
| --- | --- | --- |
| Студент гр. 0323 |  | Кольцов К.Э |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

**Цель работы.**

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:

Вывести на экран слова последовательности, не содержащие цифр.

4) Выполнить задание по варианту:

Вывести на экран ту же последовательность, удалив из всех слов заданный набор букв и (или) цифр.

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. (\*)

**Основные теоретические положения.**

Массив представляет собой индексированную последовательность однотипных элементов с заранее определенным количеством элементов. Наглядно одномерный массив можно представить, как набор пронумерованных ячеек, в каждой из которых содержится определенное значение.

Все массивы можно разделить на две группы: одномерные и многомерные. Описание массива в программе отличается от объявления обычной переменной наличием размерности массива, которая задается в квадратных скобках после имени.

Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных.

Аналогом одномерного массива из математики может служить последовательность некоторых элементов с одним индексом: a\_i*ai*​ при  i = 0, 1, 2, … n – одномерный вектор. Каждый элемент такой последовательности представляет собой некоторое значение определенного типа данных.

Указатели хранят адрес в памяти компьютера, в котором находятся переменная, от типа переменной зависит размер этой памяти, так для указателя \*int размер 4, и при операции увеличении оператора int \*p = 140, операция p + 1 выдаст 144, а не 140. Для получения указателя используется оператор &, для доступа к данным указателя операция \*. Указатели являются переменными, то есть имеют адрес в памяти, поэтому можно создавать указатели на указатели и т.д.

Тип данных string является классом для работы со строками. Работать с ним можно как с массивом символов, но также имеются собственные методы, упрощающие работу. В отличии от массива имеет свой собственный адрес, поэтому для создания указателя \*char нужно приравнивать его к нулевому элементу simpleString[0], также при передачи строки в метод передается копия строки, поэтому для работы со строкой можно использовать указатель на нее.

**Экспериментальные результаты.**

Программа выполняет команды, которые пишет пользователь, всего команд 5:

h: список команд;

n: заново задать предложение;

d: задать набор символов и вывести предложение без них;

w: найти подстроку в предложении;

e: завершить работу программы.

Команда h выводит список команд, команда e завершает программу, остальные команды вызывают соответствующие методы в коде программы.

Команда n позволяет заново задать предложение тремя способами:

f: из файла;

r: создать случайную

c: ввести с клавиатуры.

Команда d создает список символов, который нельзя печатать и выводит предложение, пропуская в нем эти символы.

Команда w считывает подстроку пользователя ищет ее с помощью двух алгоритмов и выводит скорость их работы.

Команда e заканчивает работу программы.

Работу каждой функции можно просмотреть отдельно (см. рис. 1, 2, 3)

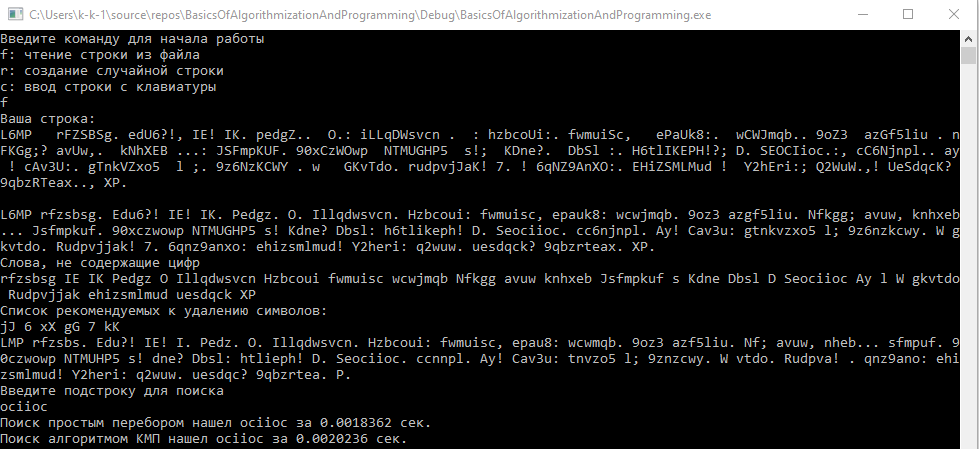


Рисунок 1 – Начало работы программы

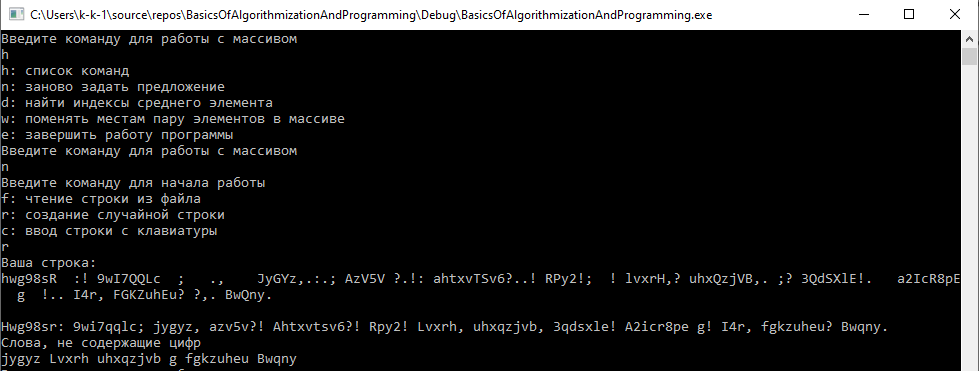


Рисунок 2 – Команды h и n

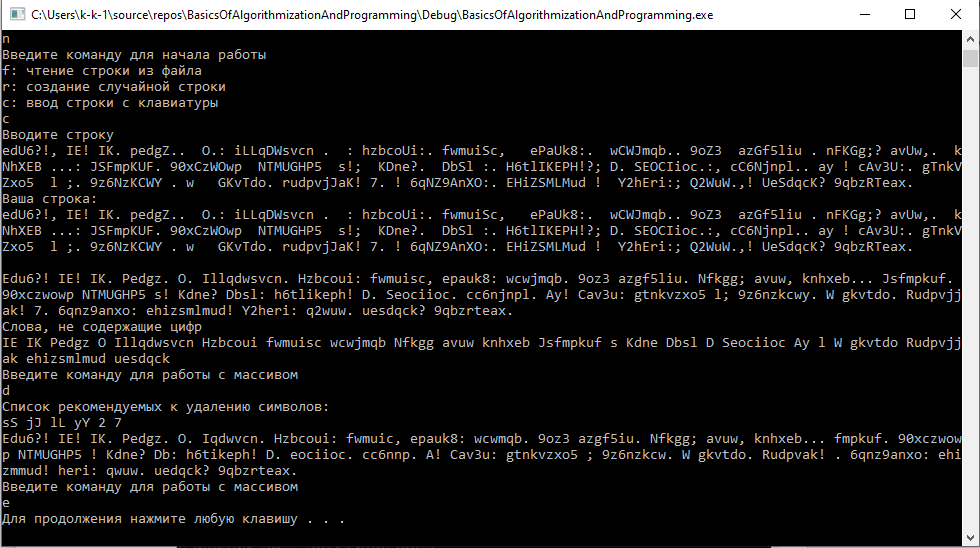


Рисунок 3 – Варианты команды n и выход из программы

**Обработка результатов эксперимента.**

Программа выводит результат корректной обработки строки и правильно находит подстроку в предложении.

**Выводы.**

В ходе данной лабораторной работы я изучил различные способы работы со строчным типом данных в c++.

ПРИЛОЖЕНИЕ А

Листинг программы

﻿#include <iostream>

#include <string>

#include <ctime>

#include <vector>

#include <fstream>

#include <chrono>

std::string makeSentence(int); //принимает указатель на строку и меняет ее на случайную строку по условиям работы

void changeSentence(std::string\*); //удаляет лишние пробелы, приводит в порядок знаки препинания и регистры

void printNotNumberWord(std::string); //печатает слова из предложения, не содержащие цифры

void printDeathNote(std::string); //печатает предложение, исключая из слова набор случайных символов

void userFindWord(std::string); //поиск подстроки пользователя

int main()

{

setlocale(LC\_ALL, "Russian"); //для корректного вывода руских букв;

std::cout.setf(std::ios::fixed); //для кореектного вывода дробных чисел

std::cout.precision(7); //только первые 7 знаков после запятой значащие для вывода временного интервала

std::srand(time(0)); //для случайности генератора предложений и списка символов для Death Note

std::string sentence; //рабочее предложение

char answer, answer1; //для приема ответов пользователя

bool flag = true, flag1;

std::cout << "Введите команду для начала работы\n" \

"f: чтение строки из файла\n" \

"r: создание случайной строки\n" \

"c: ввод строки с клавиатуры\n";

std::ifstream in;

while (flag) {//цикл для первого создания предложения

std::cin >> answer;

std::cin.ignore(INT\_MAX, '\n');

switch (answer)

{

case 'f':

in.open("SimpleText.txt");

if (in.is\_open()) {

std::getline(in, sentence);

flag = false;

}

else

std::cout << "Файл не доступен\n";

in.close();

break;

case 'r':

sentence = makeSentence(50);

flag = false;

break;

case 'c':

std::cout << "Вводите строку\n";

std::getline(std::cin, sentence);

flag = false;

break;

default:

std::cout << "Вы выбрали несуществующую команду\n";

break;

}

}

std::cout << "Ваша строка:\n";

std::cout << sentence << '\n';

changeSentence(&sentence);

std::cout << '\n' << sentence << '\n';

printNotNumberWord(sentence);

printDeathNote(sentence);

userFindWord(sentence);

flag = true;

while (flag) {//цикл для приема команд от пользователя

std::cout << "Введите команду для работы с массивом\n";

std::cin >> answer;

std::cin.ignore(INT\_MAX, '\n');

switch (answer)

{

case 'h':

std::cout << "h: список команд\n" \

"n: заново задать предложение\n" \

"d: задать набор символов и вывести предложение без них\n" \

"w: найти подстроку в предложении\n" \

"e: завершить работу программы\n";

break;

case 'n':

flag1 = true;

std::cout << "Введите команду для начала работы\n" \

"f: чтение строки из файла\n" \

"r: создание случайной строки\n" \

"c: ввод строки с клавиатуры\n";

while (flag1) {//цикл для создания нового предложения

std::cin >> answer1;

std::cin.ignore(INT\_MAX, '\n');

switch (answer1)

{

case 'f':

in.open("SimpleText.txt");

if (in.is\_open()) {

std::getline(in, sentence);

flag1 = false;

}

else

std::cout << "Файл не доступен\n";

in.close();

break;

case 'r':

sentence = makeSentence(50);

flag1 = false;

break;

case 'c':

std::cout << "Вводите строку\n";

std::getline(std::cin, sentence);

flag1 = false;

break;

default:

std::cout << "Вы выбрали несуществующую команду\n";

break;

}

}

std::cout << "Ваша строка:\n";

std::cout << sentence << '\n';

changeSentence(&sentence);

std::cout << '\n' << sentence << '\n';

printNotNumberWord(sentence);

break;

case 'd':

printDeathNote(sentence);;

break;

case 'w':

userFindWord(sentence);

break;

case 'e':

flag=false;

break;

default:

std::cout << "Вы указали несуществующую команду";

break;

}

}

system("pause");

return 0;

}

std::string makeSentence(int N) {

std::string sentence;

int count = std::rand() % (N) + 1;

for (int i = 0;i < count - 1;i++) {

for (int j = std::rand() % 10 + 1;j > 0;j--) {

int k = std::rand() % 62 + 48;

if (k > 57)

k += 7;

if (k > 90)

k += 6;

sentence += (char)k;

}

for (int j = std::rand() % 7;j >= 0;j--) {

int choice = std::rand() % 11;

switch (choice)

{

case 0:

sentence += ' ';

break;

case 1:

sentence += " ";

break;

case 2:

sentence += " ";

break;

case 3:

sentence += '!';

break;

case 4:

sentence += ',';

break;

case 5:

sentence += '.';

break;

case 6:

sentence += ':';

break;

case 7:

sentence += ';';

break;

case 8:

sentence += '?';

break;

case 9:

sentence += '...';

break;

case 10:

sentence += '..';

break;

}

}

sentence += ' ';

}

for (int j = std::rand() % 10 + 1;j > 0;j--) {

int k = std::rand() % 62 + 48;

if (k > 57)

k += 7;

if (k > 90)

k += 6;

sentence += (char)k;

}

sentence += '.';

return sentence;

}

void changeSentence(std::string\* sentence) {

char \*point = &sentence[0][0], \*begin = point, \*end = point;

bool word = true, firstWord = true;

while (\*(point)) {

if (word) {

if (((\*point >= 48) && (\*point <= 57)) || ((\*point >= 65) && (\*point <= 90)) || ((\*point >= 97) && (\*point <= 122))) {

end = point;

}

else {

word = false;

bool caps = true;

for (char \*marker = begin;marker <= end;marker++) {

if (caps) {

if (\*marker >= 97) {

caps = false;

marker = begin - 1;

}

}

else {

if ((\*marker >= 65) && (\*marker <= 90))

\*marker += 32;

}

}

if (firstWord) {

firstWord = false;

if (\*begin >= 97)

\*begin -= 32;

}

begin = end = point;

}

}

else {

if (((\*point >= 48) && (\*point <= 57)) || ((\*point >= 65) && (\*point <= 90)) || ((\*point >= 97) && (\*point <= 122))) {

word = true;

for (char \*marker = begin;marker <= end;marker++) {

switch (\*marker)

{

case ' ':

if (!(((\*(marker + 1) >= 48) && (\*(marker + 1) <= 57)) || ((\*(marker + 1) >= 65) && (\*(marker + 1) <= 90)) || ((\*(marker + 1) >= 97) && (\*(marker + 1) <= 122)))) {

sentence[0].erase(marker - &sentence[0][0], 1);

end--;

point--;

marker--;

}

break;

case '?':

if (marker != begin) {

sentence[0].erase(marker - &sentence[0][0], 1);

end--;

point--;

marker--;

}

else

firstWord = true;

break;

case '!':

if (!((marker == begin) || ((\*begin == '?') && (marker == begin + 1)))) {

sentence[0].erase(marker - &sentence[0][0], 1);

end--;

point--;

marker--;

}

else

firstWord = true;

break;

case ',':

if (!(marker == begin) || ((marker == begin + 1) && (\*begin == '.'))) {

sentence[0].erase(marker - &sentence[0][0], 1);

end--;

point--;

marker--;

if (\*begin == '.')

firstWord = false;

}

break;

case '.':

if (!((\*begin == '.') && ((marker == begin) || ((marker == begin + 1) && (\*(marker + 1) == '.')) || ((marker == begin + 2) && (\*(marker - 1) == '.'))))) {

sentence[0].erase(marker - &sentence[0][0], 1);

end--;

point--;

marker--;

}

else

firstWord = true;

break;

case ';':

case ':':

if (marker != begin) {

sentence[0].erase(marker - &sentence[0][0], 1);

end--;

point--;

marker--;

}

break;

}

}

begin = end = point;

}

else {

end = point;

}

}

point++;

}

if (word) {

bool caps = true;

for (char \*marker = begin;marker <= end;marker++) {

if (caps) {

if (\*marker >= 97) {

caps = false;

marker = begin - 1;

}

}

else {

if ((\*marker >= 65) && (\*marker <= 90))

\*marker += 32;

}

}

if (firstWord) {

firstWord = false;

if (\*begin >= 97)

\*begin -= 32;

}

}

else {

for (char \*marker = begin;marker <= end;marker++) {

switch (\*marker)

{

case ' ':

if (!(((\*(marker + 1) >= 48) && (\*(marker + 1) <= 57)) || ((\*(marker + 1) >= 65) && (\*(marker + 1) <= 90)) || ((\*(marker + 1) >= 97) && (\*(marker + 1) <= 122)))) {

sentence[0].erase(marker - &sentence[0][0], 1);

end--;

marker--;

}

break;

case '?':

if (marker != begin) {

sentence[0].erase(marker - &sentence[0][0], 1);

end--;

marker--;

}

else

firstWord = true;

break;

case '!':

if (!((marker == begin) || ((\*begin == '?') && (marker == begin + 1)))) {

sentence[0].erase(marker - &sentence[0][0], 1);

end--;

marker--;

}

else

firstWord = true;

break;

case ',':

if (!(marker == begin) || ((marker == begin + 1) && (\*begin == '.'))) {

sentence[0].erase(marker - &sentence[0][0], 1);

end--;

marker--;

if (\*begin == '.')

firstWord = false;

}

break;

case '.':

if (!((\*begin == '.') && ((marker == begin) || ((marker == begin + 1) && (\*(marker + 1) == '.')) || ((marker == begin + 2) && (\*(marker - 1) == '.'))))) {

sentence[0].erase(marker - &sentence[0][0], 1);

end--;

marker--;

}

else

firstWord = true;

break;

case ';':

case ':':

if (marker != begin) {

sentence[0].erase(marker - &sentence[0][0], 1);

end--;

marker--;

}

break;

}

}

}

}

void printNotNumberWord(std::string sentence) {

std::cout << "Слова, не содержащие цифр\n";

char \*point = &sentence[0], \*begin = point, \*end = point;

bool word = true, wordPrint = true;

while (\*(point)) {

if (word) {

if (((\*point >= 48) && (\*point <= 57)) || ((\*point >= 65) && (\*point <= 90)) || ((\*point >= 97) && (\*point <= 122))) {

end = point;

if ((\*point >= 48) && (\*point <= 57))

wordPrint = false;

}

else {

word = false;

if (wordPrint) {

for (char \*marker = begin;marker <= end;marker++)

std::cout << \*marker;

std::cout << ' ';

}

wordPrint = false;

}

}

else {

if (((\*point >= 48) && (\*point <= 57)) || ((\*point >= 65) && (\*point <= 90)) || ((\*point >= 97) && (\*point <= 122))) {

begin = end = point;

word = true;

if (!((\*point >= 48) && (\*point <= 57)))

wordPrint = true;

}

}

point++;

}

std::cout << '\n';

}

void printDeathNote(std::string sentence) {

std::string deathNote;

bool unique;

char \*begin = &deathNote[0], \*end = &deathNote[0];

std::cout << "Список рекомендуемых к удалению символов:\n";

for (int j = 6;j > 0;j--) {

unique = true;

int k = std::rand() % 36 + 48;

if (k > 57)

k += 7;

for (char \*marker = begin;marker < end;marker++)

if ((char)k == \*marker)

unique = false;

if (unique) {

if (k >= 65) {

deathNote += (char)(k + 32);

std::cout << \*(end);

end++;

}

deathNote += (char)k;

std::cout << \*(end) << ' ';

end++;

}

else

j++;

}

std::cout << '\n';

char \*point;

point = &sentence[0];

while (\*point) {

if (((\*point >= 48) && (\*point <= 57)) || ((\*point >= 65) && (\*point <= 90)) || ((\*point >= 97) && (\*point <= 122))) {

unique = true;

for (char \*marker = begin;marker < end;marker++)

if (\*point == \*marker)

unique = false;

if (unique)

std::cout << \*point;

}

else

std::cout << \*point;

point++;

}

std::cout << '\n';

}

void userFindWord(std::string sentence) {

std::cout << "Введите подстроку для поиска\n";

std::string userWord;

std::getline(std::cin, userWord);

std::chrono::system\_clock::time\_point start, fin;

std::chrono::duration<double> findTime;

bool find = false;

char \*point = &sentence[0], \*begin = &userWord[0], \*end = &userWord[userWord.length() - 1];

std::cout << "Поиск простым перебором ";

start = std::chrono::system\_clock::now();

while (\*(point + userWord.length())) {

while (\*begin == \*point) {

if (begin == end) {

find = true;

begin = &userWord[0];

std::cout << "нашел ";

for (char \*marker = begin;marker <= end;marker++) {

std::cout << \*marker;

}

point = &sentence[sentence.length() - 1 - userWord.length()];

}

begin++;

point++;

}

begin = &userWord[0];

point++;

}

fin = std::chrono::system\_clock::now();

findTime = fin - start;

if (!find)

std::cout << "не дал результата";

std::cout << " за " << findTime.count() << " сек.\n";

find = false;

std::vector<int> prefixFunxin(userWord.length());

prefixFunxin[0] = 0;

int k = 0;

std::cout << "Поиск алгоритмом КМП ";

start = std::chrono::system\_clock::now();

for (int i = 1;i < userWord.length();i++) {

while ((k > 0) && (userWord[i] != userWord[k]))

k = prefixFunxin[k - 1];

if (userWord[i] == userWord[k])

k++;

prefixFunxin[i] = k;

}

k = 0;

for (int i = 0;i < sentence.length();i++) {

while ((k > 0) && (userWord[k] != sentence[i]))

k = prefixFunxin[k - 1];

if (userWord[k] == sentence[i])

k++;

if (k == userWord.length()) {

find = true;

std::cout << "нашел ";

for (int j = i - k + 1;j <= i;j++)

std::cout << sentence[j];

i = sentence.length();

}

}

fin = std::chrono::system\_clock::now();

findTime = fin - start;

if (!find)

std::cout << "не дал результата";

std::cout << " за " << findTime.count() << " сек.\n";

std::cout << '\n';

}