**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Информационных Систем и Технологий (ИСИТ)**

отчет

**по практической работе №3**

**по дисциплине «Программирование»**

Тема: «ТЕКСТОВЫЕ СТРОКИ»

|  |  |  |
| --- | --- | --- |
| Студент гр. 0324 |  | Марычева М.Ф. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

**Цель работы.**

Ознакомление с основными особенностями работы с символьными массивами и строками в языке C++, приобретение практических навыков работы со строками, реализация различных алгоритмов, связанных со строками, в том числе и алгоритма поиска подстроки в строке, а также функциями ввода-вывода и обработки строковой информации.

**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'.

Ввод с клавиатуры

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().

Обработка текстовых строк

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)

Массивы текстовых строк

Широкое применение в программировании получила специальная форма двумерных массивов символов, которые можно рассматривать как одномерный массив строк, заканчивающихся нулевым символом. Определяются такие массивы следующим образом:

char arr\_str [30] [21];

Этот массив содержит 30 строк. Каждая строка может содержать текст из максимум 20 символов + нулевой символ.

Для того чтобы обратиться к некоторой строке этого массива достаточно указать только первый индекс.

и др.

String

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.

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;

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.

Хоть алгоритм и работает быстрее, по-прежнему необходимо сначала пройти всю строку, чтобы определить префиксы или суффиксы (вхождение (индексы) символов).

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) 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) Вывести на экран ту же последовательность, заменив во всех словах цифры на буквы латинского алфавита, номера которых в алфавите равны заменяемой цифре.

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.**

Поочередно выполняем поставленные задачи, применяя на практике знания о строках и обработке текстовой информации. Также используем дополнительные источники и курс stepik для ознакомления с теоретическими материалами, необходимыми для выполнения заданий повышенной сложности. Форматируем, добавляем цвет, создаем контекстное меню, внося в него пункты выполненных задач при помощи конструкции switch.

**Выводы.**

Были освоены новые приемы работы со строками, файлами и классом string, изучены незнакомые способы работы с текстом, изучены алгоритмы поиска подстроки в строке.

Приложение А

Полный код программы

#include <iostream>

#include <string>

#include <sstream>

#include <algorithm>

#include <fstream>

using namespace std;

void clean(string str)

{

setlocale(LC\_ALL, "Russian");

char znak[9] = { ',' , ':' , ';' , '!' , '?' , '-' , '"' };

for (int j = 0; j < str.length(); j++)

{

if (str[j] == ' ')

{

while (str[j + 1] == ' ')

str.erase(j + 1, 1);

}

}

if (str[0] == ' ')

str.erase(0, 1);

if (str[str.length() - 1] == ' ')

str.erase(str.length() - 2, 1);

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

{

for (int j = 0; j < 9; j++)

{

if (str[i] == znak[j])

{

while (str[i + 1] == znak[j])

str.erase(i + 1, 1);

}

}

}

for (int j = 0; j < str.length(); j++)

{

if (str[j] == '.' && str[j + 1] == '.' && str[j + 2] == '.')

{

j += 2;

}

else

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

{

while (str[j + 1] == '.')

str.erase(j + 1, 1);

}

}

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

{

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

}

cout << "Отредактированная строка:" << str << "\n\n";

}

bool IsStrContainsNumb(const string str)

{

return str.find\_first\_not\_of(",.:;?!-()'0123456789") == string::npos;

}

bool IsStrContainsLetters(const string str)

{

return str.find\_first\_not\_of(",.:;?!-()'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ") == string::npos;

}

void linear(string line, string substring)

{

int leng2 = substring.length();

int leng1 = line.length();

cout << "Индексы: ";

for (int i = 0; i <= leng1 - leng2; i++)

{

int j;

for (j = 0; j < leng2; j++)

if (line[i + j] != substring[j])

break;

if (j == leng2)

cout << i << " ";

}

cout << endl;

}

int main()

{

setlocale(LC\_ALL, "Russian");

string a;

cout << "Введите строку: ";

getline(cin, a);

bool filter = a.find\_first\_not\_of(",.:;?!-()'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ ") != string::npos;

bool quan = 0;

stringstream buf(a);

string Lett;

int words = 0;

while (buf >> Lett) {

words++;

if ((Lett.size() >= 10) || (Lett.size() < 1)) {

quan = 1;

}

}

int pass = 1;

if ((quan == 1) || (words < 1) || (words >= 50) || (filter == 1)) {

cout << "Введённая строка не соответствует условиям.\n 1)От 1 до 50 слов\n2)В каждом слове от 1 до 10 букв и цифр\n3)Только латинские буквы\n\n";

pass = 0;

}

while (pass == 1)

{

int number;

cout << "Выберите номер задания:\n2)Необходимо отредактировать входной текст: удалить лишние пробелы, удалить лишние знаки препинания, исправить регистр букв\n3)Вариант 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.\n4) Вариант 5: Вывести на экран ту же последовательность, переместив все цифры, содержащиеся в словах, в конец соответствующих слов.\n5)Необходимо найти все подстроки, которую введёт пользователь в имеющейся строке.\n6)Выход\n\n";

cin >> number;

switch (number) {

case 2: {

clean(a);

break;

}

case 3: {

clean(a);

string let;

string num;

string letAndnum;

istringstream a2(a);

while (a2 >> a) {

if (IsStrContainsNumb(a) == 1) {

num += a;

num += " ";

}

if (IsStrContainsLetters(a) == 1) {

let += a;

let += " ";

}

if ((IsStrContainsLetters(a) == 0) && (IsStrContainsNumb(a) == 0)) {

letAndnum += a;

letAndnum += " ";

}

}

cout << "Слова, содержащие только буквы: " << let << endl;

cout << "Слова, содержащие только цифры: " << num << endl;

cout << "Слова, содержащие буквы и цифры: " << letAndnum << endl;

cout << endl;

break;

}

case 4:

{

auto start = a.begin();

auto space = a.begin();

while (true) {

space = find(start, a.end(), ' ');

stable\_partition(start, space, [](char a) {return !isdigit(a); });

if (space == a.end())

break;

start = space + 1;

}

cout << a << endl;

break;

}

case 5:

{

string a2;

cout << "\nВведите подстроку: ";

cin >> a2;

linear(a, a2);

break;

}

case 6:

return 0;

break;

}

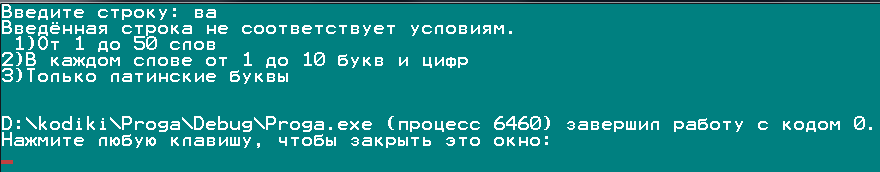
}

}

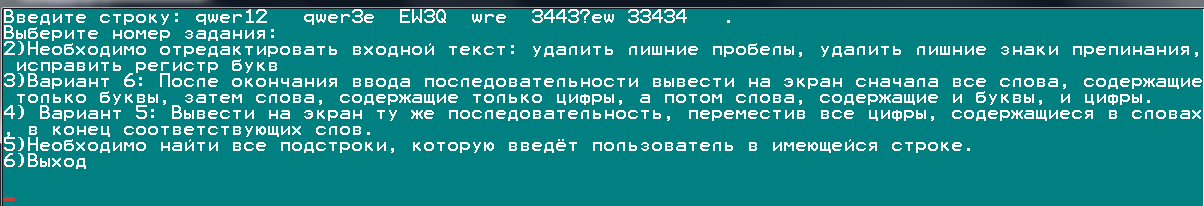
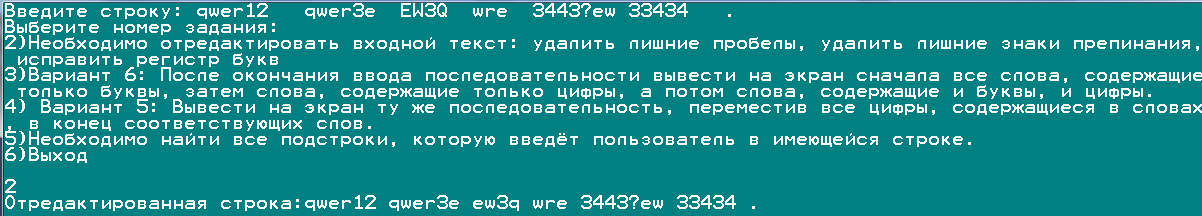
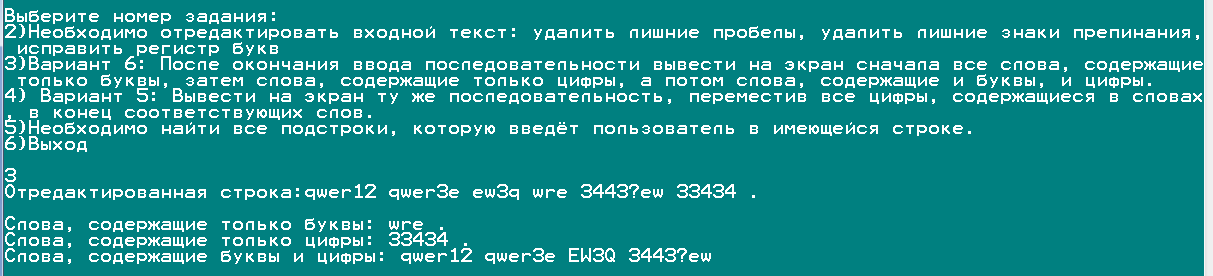
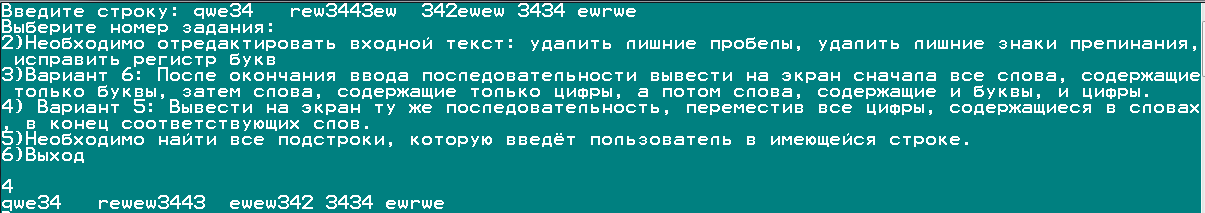
Приложение б

Полученные результаты

0.1)Ошибка ввода



0.2)Меню

1. 
2. 
3. 
4. 
5. 