**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №3**

**по дисциплине «Программирование»**

Тема: "**ДВУМЕРНЫЕ СТАТИЧЕСКИЕ МАССИВЫ. УКАЗАТЕЛИ"**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3372 |  | Матвиенко А. В. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Научиться работать с текстовыми строками, изучить класс string, научиться проводить операции с подстроками, а также изучить массивы строк.

**Основные теоретические положения.**

### Определение текстовой строки

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)

и др.

Написать соответствующие функции обработки достаточно просто. Вот два примера:

**Пример 1**: Функция определения фактической длины строки:

unsigned my\_StrLen(char \*S)

{

unsigned L = 0;

while (S[L]) // При достижении символа с числовым значением 0 выход из цикла

++L;

return L;

}

**Пример 2**: Функция добавления строки S2 в конец строки S1 (обе строки должны заканчиваться нулевым символом, и строка S1 должна иметь достаточную длину для добавления символов строки S2):

void my\_StrCat(char \*S1, char \*S2)

{

     unsigned j = my\_StrLen(S1);

     for (unsigned i = 0; S2 [i]; ++ i, ++j)

          S1 [j] = S2 [i];

     S1 [++j] = '\0';

}

Все достаточно просто. Тем более, что делать это необходимости нет, так как аналогичные и многие другие функции по обработке строк, завершающихся нулевым символом, уже имеются в библиотеках. Одну из таких библиотек можно подключить к программе с помощью заголовочного файла **<cstring>**.  Вот наиболее распространенные функции из этой библиотеки:

Функция **strlen(char \*s)** – возвращает фактическую длину текстовой строки, хранящейся в символьном массиве **s** (см. аналог **my\_StrLen(char \*S)**).

Функция **strcpy(char \*dest, char \*source)** – копирует содержимое строки **source** в строку **dest**.

Функция **strcat(char \*s1, char \*s2)** – добавляет содержимое строки **s2** в конец строки **s1**.

Функция **strcmp(char \*s1, char \*s2)** – осуществляет лексикографическое сравнение строк **s1**и **s2**. Возвращает значение 0, если строки одинаковы (равны), значение большее 0 при **s1 > s2**и отрицательное значение при **s1 < s2**.

**Пример**. Имеются две строки **S1** и **S2**, содержащие некоторые тексты. Необходимо поменять содержимое этих строк так, чтобы строка S2 содержала “больший” текст. Реализация:

char S1 [41], S2 [41], B [41];

cin >> S1;

cin >> S2;

if (strcmp (S1, S2))

     {

            strcpy(B, S1);

            strcpy(S1, S2);

            strcpy(S2, B);

     }

cout << S1 << endl;

cout << S2 << endl;

В этой же библиотеке содержатся другие варианты упомянутых функций и множество других полезных обработчиков строк.

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.

/\* string.cpp: Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы \*/

#include <iostream>

#include <string>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

string A = "Пример"; // Объявляем и инициализируем строку А

string B; // Объявляем строку В

string C = "текста"; // Объявляем и инициализируем строку С

B = A; // Копируем текст из строки А в строку В

C = B + " " + C;

if(C == "Пример текста") // Проверяем корректно ли произошло присоединение строк

cout << C;

return 0;

}

Результат выполнения программы – вывод в консоль сообщения:

Пример текста

С string можно использовать оператор индексации и получать значения символа. Принцип действия такой же, как и у типа char:

string s1 = “hello!”;

cout << s1[0]; // Будет выведен символ ‘h’

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;

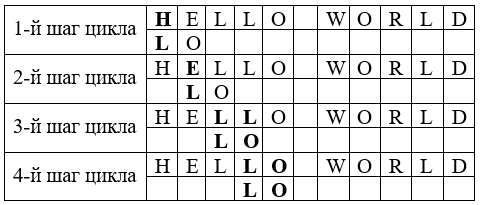
·        функция insert() необходима, чтобы вставить одну строку в заданную позицию другой строки;

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

### Прямой поиск

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.

Одной из вариаций алгоритма полного перебора является реализация таблицы включений. 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 поиска подстроки в строке прямого поиска (полного перебора) на языке C++ может быть реализован следующим образом:

#include <iostream>

#include <string>

using namespace std;

int searchSubstring(string text, string pattern) {

    int n = text.length(); // длина текста

    int m = pattern.length(); // длина подстроки

    for (int i = 0; i <= n - m; i++) {

        int j;

        for (j = 0; j < m; j++) {

            if (text[i + j] != pattern[j])

                break;

        }

        if (j == m) // если все символы совпали

            return i; // вернуть индекс начала подстроки

    }

    return -1; // подстрока не найдена

}

int main() {

    string text = "Hello, World!";

    string pattern = "World";

    int result = searchSubstring(text, pattern);

    if (result == -1)

        cout << "Подстрока не найдена" << endl;

    else

        cout << "Подстрока найдена в позиции: " << result << endl;

    return 0;

}

Алгоритм просто перебирает все возможные подстроки в исходной строке и сравнивает их с искомой подстрокой. Если все символы совпадают, то возвращается индекс начала этой подстроки в исходной строке. Если подстрока не найдена, то функция возвращает -1. В данном примере ищется подстрока "World" в строке "Hello, World!", и будет выведено сообщение "Подстрока найдена в позиции: 7".

**Постановка задачи.**

 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.  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);

Вывести на экран только те слова последовательности, в которых первая буква слова встречается в этом слове еще раз. Вывести на экран ту же последовательность, удалив из всех слов заданный набор букв и (или) цифр.

**Выполнение работы.**

|  |  |
| --- | --- |
| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| Меню | |
| При запуске программы перед пользователем появляется окно с меню, где он вводит строку, а затем может выбрать номер задания последовательности. | Меню:    Ввод строки: |

Продолжение Таблицы

|  |  |
| --- | --- |
| Выполнение 1 задания | |
| При вводе пользователем корректного значения пункта меню программа выполняется. | Результат выполнения программы – вывод строки, введенной пользователем |

|  |  |
| --- | --- |
| Выполнение 2 задания | |
| Результат выполнения программы – отредактированный текст |  |

|  |  |
| --- | --- |
| Выполнение 3 задания | |
| Выводятся элементы, в которых первая буква повторяется несколько раз |  |
| Выполнение 5 задания | |
| Выводится индекс введенной подстроки |  |

**Выводы.**

Я научился работать с текстовыми строками, изучил класс string, научился проводить операции с подстроками, а также изучил массивы строк.

Приложение А

рабочий код

#include <iostream>

#include <windows.h>

#include <conio.h>

//#include <string>

#include <algorithm>

#include <vector>

#include <sstream>

using namespace std;

unsigned getStrLen(char\* S)

{

unsigned L = 0;

while (S[L]) ++L;

return L;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char vvod[1000];

char dot[2] = ".";

cout << "Введите строку" << endl;

cin.getline(vvod, 1000, '.');

strcat(vvod, dot);

vector<char> vvodVector;

string str;

str = vvod;

int len = str.size();

bool flag = false;

for (int i = 0; i < len; i++)

{

if ((vvod[i + 1] == ' ' || (vvod[i] == ',' || vvod[i] == '?' || vvod[i] == ',' || vvod[i] == ':' || vvod[i] == '!' || vvod[i] == ';' || vvod[i] == '-' || vvod[i] == ')' || vvod[i] == '(')) && flag == true)

{

continue;

}

if ((vvod[i + 1] == ' ' || (vvod[i] == ',' || vvod[i] == '?' || vvod[i] == ',' || vvod[i] == ':' || vvod[i] == '!' || vvod[i] == ';' || vvod[i] == '-' || vvod[i] == ')' || vvod[i] == '(')) && flag == false)

{

vvodVector.push\_back(vvod[i]);

flag = true;

}

else

{

vvodVector.push\_back(vvod[i]);

flag = false;

}

}

for (int i = 0; i < vvodVector.size(); i++)

{

if (vvodVector[i] >= 'А' && vvodVector[i] <= 'Я')

{

vvodVector[i] += 'я' - 'Я';

}

}

vvodVector[0] = vvodVector[0] - 32;

for (int i = 2; i < vvodVector.size(); i++)

{

if (vvodVector[i - 2] == '!' || vvodVector[i - 2] == '?')

{

vvodVector[i] = vvodVector[i] - 32;

}

}

int choose = 0;

while (choose != -1)

{

cout << "\nВыберите задание:\n\n";

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

<< "2) Необходимо отредактировать входной текст: \n\t· удалить лишние пробелы; \n\t· удалить лишние знаки препинания(под «лишними» подразумевается несколько подряд идущих знаков \n\t«…» - корректное использование знака в тексте); \n\t· исправить регистр букв, если это требуется(пример некорректного использования регистра букв : пРиМЕр);\n\n"

<< "3) Вывести на экран только те слова последовательности, в которых первая буква слова встречается в этом слове еще раз.(5)\n\n"

<< "4) Вывести на экран ту же последовательность, удалив из всех слов заданный набор букв и (или) цифр.(3)\n"

<< "5) Необходимо найти все подстроки, которую введёт пользователь в имеющейся строке. Линейный поиск.\n";

int input;

cin >> input;

switch (input)

{

case 1:

{

string str;

str = vvod;

cout << "\nВы ввели: " << vvod << "\n" << endl;

cout << "Длина строки - " << str.size() << endl;

break;

}

case 2:

{

for (int i = 0; i < vvodVector.size(); i++)

{

cout << vvodVector[i];

}

break;

}

case 3:

{

vector<char> vvodVectorDownReg = vvodVector;

for (int i = 0; i < vvodVector.size(); i++)

{

if (vvodVector[i] >= 'А' && vvodVector[i] <= 'Я')

{

vvodVectorDownReg[i] += 'я' - 'Я';

}

}

char words[1000];

char str[2];

int count = 0;

string word = "";

for (char x : vvodVectorDownReg)

{

if (x == ' ')

{

//cout << word << endl;

word = "";

count++;

}

else

{

word = word + x;

}

}

//cout << count;

char vvodClean[1000];

for (int i = 0; i < vvodVector.size(); i++)

{

vvodClean[i] = vvodVectorDownReg[i];

//cout << vvodClean[i];

}

string line = vvodClean;

int wordsCount = count;

string arr[1000];

int i = 0;

stringstream ssin(line);

while (ssin.good() && i < wordsCount)

{

ssin >> arr[i];

i++;

}

cout << "Элементы, в которых первая буква повторяется несколько раз:\n";

for (i = 0; i < wordsCount; i++)

{

bool f = false;

for (int j = 0; arr[i][j] != '\0' ; j++)

{

if (arr[i][0] == arr[i][j+ 1])

{

f = true;

}

}

if (f == true)

{

cout << arr[i] << " ";

}

//cout << arr[i];

}

cout << endl;

break;

}

case 4:

{

break;

}

case 5:

{

cout << "Введите подстроку:\n";

char supString[100];

cin >> supString;

cout << "Индекс введенной подстроки в осортированной строке:\n";

for (int i = 0; i < vvodVector.size(); i++)

{

bool flag = false;

int counter = 0;

for (int j = 0; supString[j] != '\0'; j++)

{

if (supString[j] != vvodVector[i + counter])

{

flag = true;

break;

}

counter++;

}

if (flag == false)

{

cout << i << " ";

}

}

break;

}

default:

cout << "Доступно 4 задания.\n\n";

break;

}

\_getch();

}

return 0;

}