## МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬ­НИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **КНІТ**

Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 4

**На тему:** *“* *Символьні рядки в C. Ввід та вивід ”*

**З дисципліни:** *“Основи програмування”*

## Лектор:

ст.викл. каф. ПЗ

Муха Т.О.

**Виконав:**

ст. гр. ПЗ-18

Юшкевич А.І.

**Прийняв:**

ст. викл. каф. ПЗ

Кутельмах. Р.К.

« » 2022 р.

∑= .

Львів – 2022

**Тема роботи:** Символьні рядки в C. Ввід та вивід.

**Мета роботи:** здобути практичні навики опрацювання текстової інформації з врахуванням особливостей організації символьних рядків у мові С. Вивчити основні засоби потокового вводу/виводу в С.

**Лабораторне завдання**

**7.** Скласти програму на мовіС, яка обчислюватиме значення виразу з лабораторної роботи №1 Додаток 1 “Обчислення заданих арифметичних виразів”. Значення параметрів X, Y, Z прочитати із заданого бінарного файлу (вводиться користувачем з клавіатури). Результат записати у файл, теж вказаний користувачем. Формат файлу (бінарний чи текстовий) також задається користувачем. При цьому робота програми повинна бути запротокольована. Виконання основних подій, починаючи зі старту програми, має бути відображене у log-файлі з часовими мітками (розміщення вибрати самостійно). Старі записи у log-файлі мають зберігатися. Перелік подій для логування:

* + старт програми
  + відкриття файлу з параметрами
  + обчислення значення виразу
  + запис обчисленого значення у вихідний файл  завершення програми Приклад log-файлу:

**[2016.09.01 8:30.000] Program started.**

**[2016.09.01 8:30.011] Parameters file “d:\data\input.bin” opened. X=3, Y=1, Z=2.4.**

**[2016.09.01 8:30.012] Expression calculated. Result = 4.56.**

**[2016.09.01 8:30.014] Output file “d:\data\output.txt” saved.**

**[2016.09.01 8:30.015] Program ended.**

Для отримання поточного часу можна скористатися функцією time() із заголовного файлу *time.h.*

**Індивідуальне завдання**

**1)** **25.** Порахувати кількість приголосних у введеному реченні.

**Теоретичні відомості**

Мова С не має спеціального типу для оголошення символьних рядків, а розглядає символьний рядок як особливий вид масиву. Елементи масиву, який називають символьним рядком, мають тип char, його значеннями є коди символів, з яких складається цей рядок (ASCII-коди, якщо заданий компілятор застосовує ASCII-таблицю для кодування символів). Останнім символом рядка повинен бути т. зв. нуль-символ ('\0'), код якого дорівнює 0. 3 кожним символьним рядком пов'язується вказівник на початок даного рядка. У всьому іншому – символьні рядки повністю зберігають властивості масивів.

## 2.1 Оголошення та ініціалізація символьних рядків

Рядкові константи (літерали) в мові С записуються як послідовність довільних символів взятих у подвійні лапки: "...". В оперативній пам'яті їм виділяється ділянка, обсяг якої на один байт більший за кількість символів у рядку. В цей додатковий байт автоматично записується нуль-символ (‘\0’), який надалі слугуватиме ознакою кінця рядка.

**Приклад:** char \*pst = "Hello world!";

Для збереження в пам’яті записаного рядка компілятор виділить 13 байтів, з них 12 байтів для символів і останній для '\0' . Адресу початку рядка отримає вказівник pst. Символьні рядки також можуть оголошуватися як звичайні масиви:

char імя\_символьного\_рядка [кількість\_символів];

Оголошений нижче масив str призначений для збереження символьного рядка: char str[150];

У str можна записати довільний символьний рядок, довжина якого не перевищує 149 символів, оскільки останнім записується нуль-символ – для нього треба обов'язково зарезервувати один байт. Слід також пам'ятати, що ***перевищення встановленої в оголошенні кількості символів не контролюється компілятором і може призвести до небезпечних помилок у роботі програми***.

В оголошеннях символьні рядки, як і масиви символів, можна ініціалізувати. Розглянемо декілька характерних прикладів:

char ml[20] = {'a', 'b', 'с', 'd', 'e', 'f'}; char m2[20] = {'a', 'b', 'с', 'd', 'e', ' f', '\0'};

char m3[20] = "abcdef"; char m4[ ] = "abcdef";

Символьні рядки m1, m2 і m3 оголошено однаково – як масиви з 20 елементів, що мають тип *char*, але ініціалізацію їх виконано різними способами. Початкові шість елементів масиву m1 заповнено послідовністю літер, проте без ' \0' у кінці, тому цей масив не буде повноправним символьним рядком, з ним можна буде працювати тільки як із звичайним масивом символів. У масив m2 записано таку ж послідовність літер, а після неї – нуль-символ. Фактично в m2 занесено рядок символів "abcdef" . Такий же рядок записано в масив m3 , тобто результати ініціалізації m2 та m3 збігаються (очевидно, що ініціалізація m3 є простішою у записі).

Незаповнені елементи масивів ml, m2 та m3 містять"сміття" (за умови, що масиви оголошено як локальні, а в разі глобальних чи статичних масивів усі вільні елементи заповнюються нулями). Надлишкові елементи можна використовувати надалі для доповнення і розширення відповідних рядків. В оголошенні масиву m4 не вказано граничну кількість символів, тому розмірність цього масиву (символьного рядка) встановлюється за кількістю елементівініціалізаторів. Для наведеного прикладу розмірність m4 становитиме 7 символів: 6 перших байтів масиву заповнюються кодами літер, а сьомий – кінцевий нуль-символом. Хоча масиви m3 та m4 проініціалізовані константними рядками, елементи цих масивів можна змінювати так само, як елементи масивів m1 та m2.

## 2.2. Звертання до елементів символьних рядків

Процеси опрацювання символьних рядків базуються на двох основних властивостях рядків:

1) ім'я символьного рядка є константним вказівником на його перший символ; 2) кінець рядка задається нуль-символом ' \0'.

Для звертання до символів рядка застосовують як індексну, так і вказівникову систему доступу до елементів масиву

## 2.3. Бібліотечні функції для роботи з символами та символьними рядками

Стандартна бібліотека мови С включає набір різнотипних функцій, що забезпечують швидку реалізацію операцій, які найчастіше зустрічаються у процесах опрацювання символьних і текстових даних.

### **2.3.1. Функції класифікації та перетворення символів**

У заголовному файлі <ctype. h> оголошено групу функцій, призначених для перевірки та класифікації окремих символів. Імена цих функцій починаються префіксом is: is... (). Усі функції мають один параметр з типом int – символ, що перевіряється. Функції перевірки повертають ціле ненульове значення (істина), якщо заданий символ належить до відповідної класифікаційної групи, і нульове значення (хибне), якщо символ не належить до цієї групи. Наприклад, функцію, яка перевіряє, чи заданий символ sym є літерою (до уваги беруться тільки великі та малі латинські літери), оголошено так:

int isalpha (int sym);

У табл.1 наведено найбільш популярні функції класифікації та зміни символів.

Дві останні функції з табл. 1: tolower() та toupper() призначені для зміни регістра символів-літер (на жаль, тільки латинських). Перша з цих функцій повертає відповідну малу літеру, якщо sym велика латинська літера, а друга – повертає велику літеру, якщо sym мала латинська літера. В інших випадках обидві функції повертають значення sym.

Табл. 1. Основні функції класифікації та зміни символів

|  |  |
| --- | --- |
| Функція | Призначення |
| класифікації: | Перевіряє, чи символ sym є: |
| int isalpha(sym) | малою або великою латинською літерою |
| int isdigit(sym) | десятковою цифрою |
| int isalnum(sym) | латинською літерою або десятковою цифрою |
| int isxdigit(sym) | шістнадцятковою цифрою |
| int isspace(sym) | пробільним символом (символом пробілу, нового рядка, горизонтальної чи вертикальної табуляції) |
| int islower(sym) | малою латинською літерою |
| int isupper(sym) | великою латинською літерою |
| перетворення: | Повертає: |
| int tolower(sym) | малу латинську літеру, якщо sym велика латинська літера |
| int toupper(sym) | велику латинську літеру, якщо sym мала латинська літера |

## 2.4. Функції операцій над символьними рядками

Прототипи бібліотечних функцій, призначених для роботи з символьними рядками, оголошені в заголовному файлі <string.h>. Основну групу складають функції, які починаються префіксом str: str...(). У табл. 2 описано функції, які найчастіше використовуються у процесах опрацювання символьних рядків. Щоб скоротити записи прототипів функцій у табл.2, в списку параметрів кожної функції вказано тільки імена параметрів. Типи параметрів в оголошеннях цих функцій є такими:

const char \*s, \*sl, \*s2; char \*sr; unsigned n; int sym;

***Символьні рядки, які опрацьовуються функціями <string.h>, мають обов'язково закінчуватись ' \0'***.У разі звертання до функцій конкатенації(об'єднання)та копіювання рядківтреба забезпечити, щоб розмірність масиву символів sr була достатньою для запису рядка результату, оскільки функції не контролюють довжин рядків. Крім цього, рядки sr і s у функціях конкатенації та копіювання не повинні перекриватись.

Щоб вилучити підрядок, знаходимо його початок (перший символ '#' у str) і кінець (останній символ '#'). Потім переносимо, використовуючи функцію strcpy(), кінцеву частину основного рядка на місце підрядка, який треба видалити.

Табл. 2. Основні функції опрацювання символьних рядків

|  |  |
| --- | --- |
| Функція | Призначення |
| char\* strcpy(sr, s); | Копіює рядок s (з '\0' включно) за адресою, заданою параметром sr. Повертає значення sr - адресу скопійованого рядка. |
| char\* strcat(sr, s); | Додає рядок s (з '\0' включно) у кінець рядка sr. Повертає значення sr – адресу доповненого рядка. |
| int strcmp(s1, s2); | Послідовно порівнює символи рядків s1 і s2 як дані з типом unsigned char. Повертає ціле число, значення якого <0, якщо s1<s2; 0, якщо s1 == s2; > 0, якщо s1 > s2. |
| char\* strncpy(sr, s, n); | Аналог strcpy(), але з s копіюється не більше, ніж n початкових символів; якщо скопійована група символів не закінчується '\0', то нуль-символ у sr не заноситься. |
| char\* strncat (sr, s, n); | Аналог strcat(), але додає до sr тільки n початкових символів з s; у кінець об'єднаного рядка заноситься '\0'. |
| char\* strncmp(s1, s2, n); | Аналог strcmp(), але порівнює тільки n початкових символів рядків s1 та s2. Якщо якийсь із рядків коротший за n, то порівняння припиняється з досягненням ' \0'. |
| unsigned strlen(s); | Повертає довжину рядка s у символах (' \ 0' не враховується). |
| char\* strchr(s, sym); | Перевіряє, чи символ sym входить у рядок s. Повертає вказівник на перше входження sym у s або NULL, якщо sym не зустрічається в рядку s. |
| char\* strrchr(s, sym); | Аналог strchr(). Повертає вказівник на останнє входження символа sym у рядок s або NULL, якщо sym не зустрічається в рядку s. |
| char\* strstr(s1, s2); | Перевіряє, чи рядок s2 входить як підрядок у s1. Повертає вказівник на перший символ рядка s2 у s1 або NULL, якщо рядок s2 не зустрічається у рядку s1. |
| char\* strtok(sr, s); | Виділяє в рядку sr лексеми, обмежені символами з рядка s. Повертає вказівник на виділену лексему або NULL. |
| char\* strdup(s); | Копіює рядок s (з ' \0 ' включно) в динамічну пам'ять, попередньо виділивши там ділянку потрібної довжини.  Повертає адресу рядка в динамічній пам'яті. |

Останньою в табл. 2 записана функція strdup(), яка заносить копію заданого символьного рядка (з нуль-символом включно) у динамічну пам’ять та повертає адресу, за якою записано рядок. Функція strdup() не належить до функцій, що підтримуються стандартом мови С, хоча вона входить до складу більшості С-бібліотек.

**Функція виділення лексем**.Зупинимось детальніше на функціїstrtok(),оголошення якої наступне:

char\* strtok(char\* str, const char\* lim);

Ця функція виконує поділ символьного рядка str на окремі лексеми, записуючи після кожної лексеми '\0'. Рядок lim задає набір символів, якими можуть бути обмежені лексеми рядка str (нуль-символ у переліку обмежувачів вказувати не треба). Для виділення всіх лексем символьного рядка функцію використовують циклічно. У першому звертанні до strtok() вказують адресу початку рядка, а функція повертає адресу першої знайденої лексеми. У наступних звертаннях до strtok() замість першого параметра записують порожній вказівник NULL, а функція повертає адресу наступної лексеми рядка. Коли всі лексеми виділені, функція повертає NULL.

**2.5. Функції перетворення рядків символів у числа та зворотних перетворень**

**Перетворення "символьний рядок => число".** У багатьох задачах необхідно

Перетворювати числові дані, записані у формі текстових рядків, в одну з внутрішніх форм збереження чисел. Такі перетворення реалізують стандартні бібліотечні функції, оголошені в <stdlib.h> (табл.3). Типи параметрів цих функцій наступні:

const char \*st;

char \*\*end; int base;

Табл. 3. Функції перетворення символьних рядків у числа

|  |  |
| --- | --- |
| Функція | Призначення |
| int atoi(st); | Виділяє у рядку st перше ціле десяткове число і перетворює його у дане з типом int. Числу може передувати довільна кількість символів пробілу. Кінцем рядка вважається перший символ, що не належить до цифр. Повертає знайдене числове значення у разі успішного перетворення, а в разі помилки - результат не визначений. |
| long atol(st) ; | Аналог atoi(), але перетворює рядок у число з типом long. |
| double atof(st); | Аналог atoi(), але перетворює рядок st у дане з типом double. Число у рядку може бути записане як ціле чи як дійсне у формі з фіксованою або з плаваючою крапкою. |
| long strtol(st, end, base); | Розширений варіант atol(). Вказівник end (цей параметр є вказівником на вказівник) задає адресу змінної-вказівника, в яку буде записано адресу першого символа, що залишився неперетвореним. Параметр base визначає основу системи числення, в якій записано число і може приймати значення від 2 до 36: цифрами числа можуть бути арабські цифри і послідовні малі або великі латинські літери (для base >10). Якщо base дорівнює 0, то основа числа визначається формою його запису: число, що починається 0, вважається вісімковим, число з префіксом 0X чи 0x – шістнадцятковим, всі інші числа розглядаються як десяткові. |
| unsigned long strtoul(st, end,  base); | Аналог strtol(), але повертає значення, що має тип unsigned long. |
| double strtod(st, end); | Розширений варіант atof(). Перетворює початкову частину st у дане з типом double. Додатково повертає через end адресу першого символа, записаного за числом. |
| float strtof(st, end); | Аналог strtod(), але повертає значення з типом float. |
| long double strtold (st, end); | Аналог strtod(), але повертає значення, що має тип long double. |

**Перетворення "число => рядок"**.Бібліотека мови програмування С додатково містить ряд функцій зворотних перетворень "число => символьний рядок серед яких:

char\* itoa(int num, const char\* str, int base); char\* ltoa(long num, const char\* str, int base); char\* ultoa(unsigned long num, const char\* str, int base);

Ці функції відрізняються між собою тільки типом параметра num – цілого числа, яке потрібно перетворити. Всі три функції формують із числа num рядок символів, що відповідає запису цього числа в системі числення з основою base, і повертають вказівник на перший символ створеного рядка. Сформований рядок записується за адресою str. Функції itoa() та ltoa() записують від'ємні числа зі знаком мінус

## 2.6. Масиви символьних рядків і масиви вказівників

Практичні задачі часто вимагають опрацювання груп символьних рядків. У таких випадках створюють або масиви символьних рядків, або масиви вказівників на перші символи рядків, а самі рядки розташовують в оперативній пам'яті окремо. Розглянемо ці два підходи.

### 2.6.1. Масиви символьних рядків

Поширеним видом багатовимірних масивів є масиви символьних рядків, тобто масиви, кожен елемент яких – окремий символьний рядок. Наприклад, масив найменувань міст, оголошений і проініціалізований наступним чином:

char cities[6][20] = {"Львів", "Хмельницький", "Полтава",

"Рівне", "Івано-Франківськ", "Київ"};

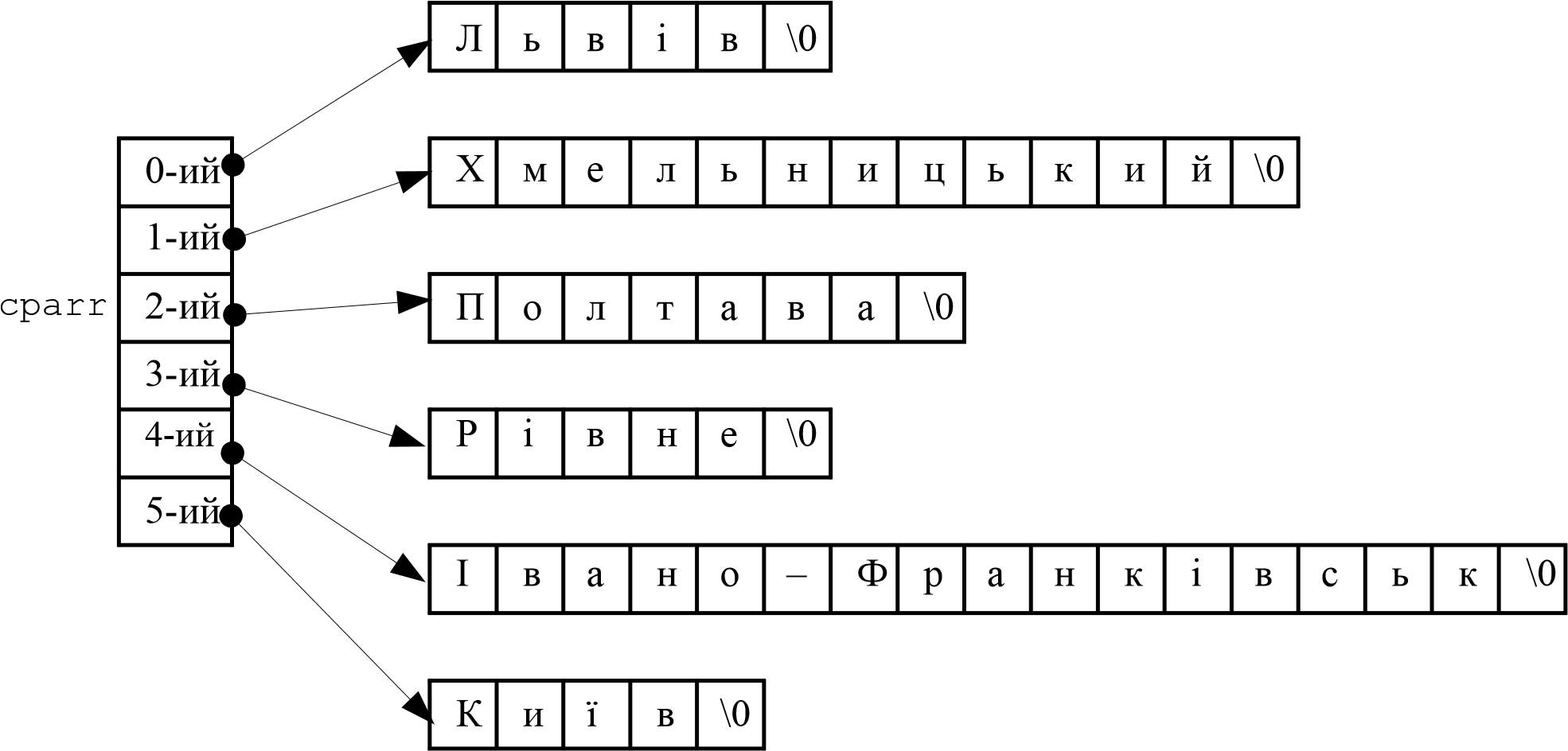
є двовимірним масивом з шести рядків, кожен з яких заповнений найменуванням відповідного міста

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| Л | ь | в | і | в | \0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| X | м | е | л | ь | н | и | ц | ь | к | и | й | \0 |  |  |  |  |  |  |  |
| П | о | л | т | а | в | а | \0 |  |  |  |  |  |  |  |  |  |  |  |  |
| Р | і | в | н | е | \0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| І | в | а | н | о | – | Ф | р | а | н | к | і | в | с | ь | к | \0 |  |  |  |
| К | и | ї | в | \0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Правила звертання до елементів масиву символьних рядків такі ж, як і для всіх інших багатовимірних масивів. Зокрема, перший рядок масиву ("Львів") виділяють вирази \*cities та cities[0]. Cities

**2.6.2. Масиви вказівників на символи рядків**

Створення масиву вказівників на перші символи набору рядків, кожен з яких зберігається в оперативній пам'яті окремо, – це альтернативний підхід, що дає змогу усунути недолік виділення зайвого місця, властивий масивам символьних рядків. Розглянемо програму, яка здійснює випадковий вибір одного міста зі заданого списку міст (імітація жеребкування). В програмі створено масив вказівників cparr, елементи якого мають тип char\*. Масив проініціалізовано стрінговими константами і найменуваннями міст. Кожному рядку, тобто найменуванню міста, компілятор виділяє в пам'яті ділянку, обсяг якої дорівнює довжині рядка плюс нуль-символ. Адреса розміщення рядка заноситься у відповідний елемент масиву cparr.



Щоб забезпечити можливість зміни складу і кількості міст, у програмі в оголошенні масиву cparr не вказано розмірність цього масиву. Тому задана кількість міст обчислюється за допомогою виразу sizeof cparr / sizeof(char \*)

Значення цього виразу використано як максимальне значення генерованого числа за допомогою функції rand() для визначення номера вибраного міста. (Функція rand() генерує псевдовипадкове дійсне число в діапазоні 0.. 32767).

Перевага використання масивів вказівників, які зберігають адреси початків рядків замість масивів символьних рядків особливо відчутна, коли в програмі треба переставляти окремі рядки місцями, наприклад, потрібно впорядкувати набір рядків. У цих випадках самі символьні рядки не переписують (це важливо, адже їх розмір може бути достатньо великим), а тільки міняють місцями вказівники на початки рядків.

**№1**

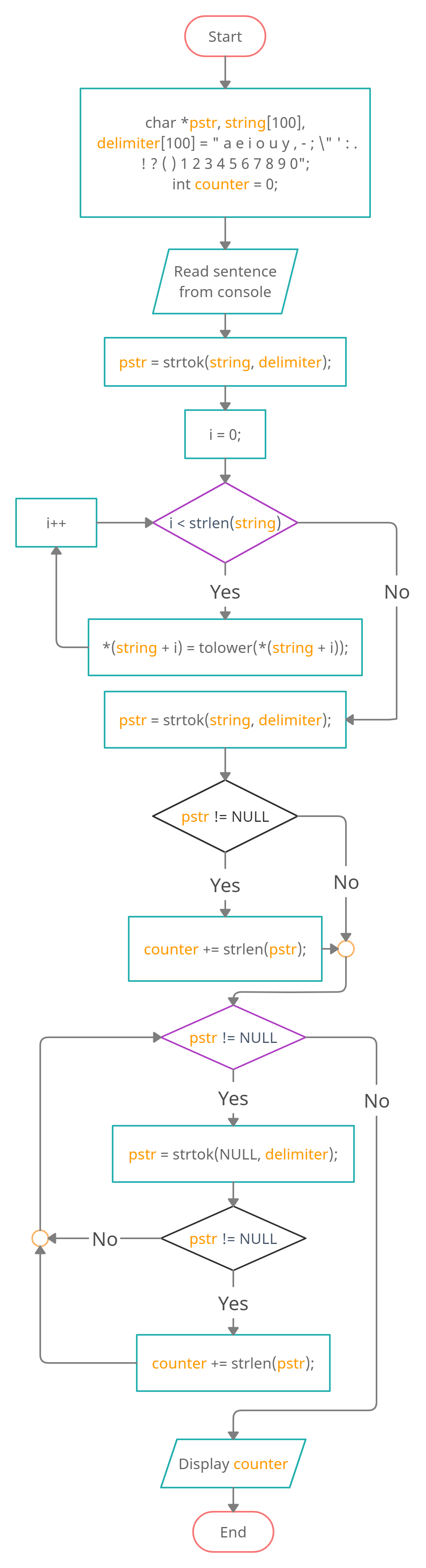


Рис.1 Блок-схема першої програми

# Код програми

Назва файлу: **Lab04.c**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <string.h>

int main(void) {

char \*pstr, string[100], delimiter[100] = " a e i o u y , - ;"

"\" ' : . ! ? ( ) 1 2 3 4 5 6 7 8 9 0";

int counter = 0;

puts("Enter sentence: ");

gets(string);

for (int i = 0; i < strlen(string); i++)

\*(string + i) = tolower(\*(string + i));

pstr = strtok(string, delimiter);

if (pstr != NULL)

counter += strlen(pstr);

while (pstr != NULL) {

pstr = strtok(NULL, delimiter);

if(pstr != NULL)

counter += strlen(pstr);

}

printf("Number of consonants is equal %d", counter);

return 0;

}

# Протокол роботи

1. Користувач вводить речення
2. Програма виводить кількість приголосних у реченні.

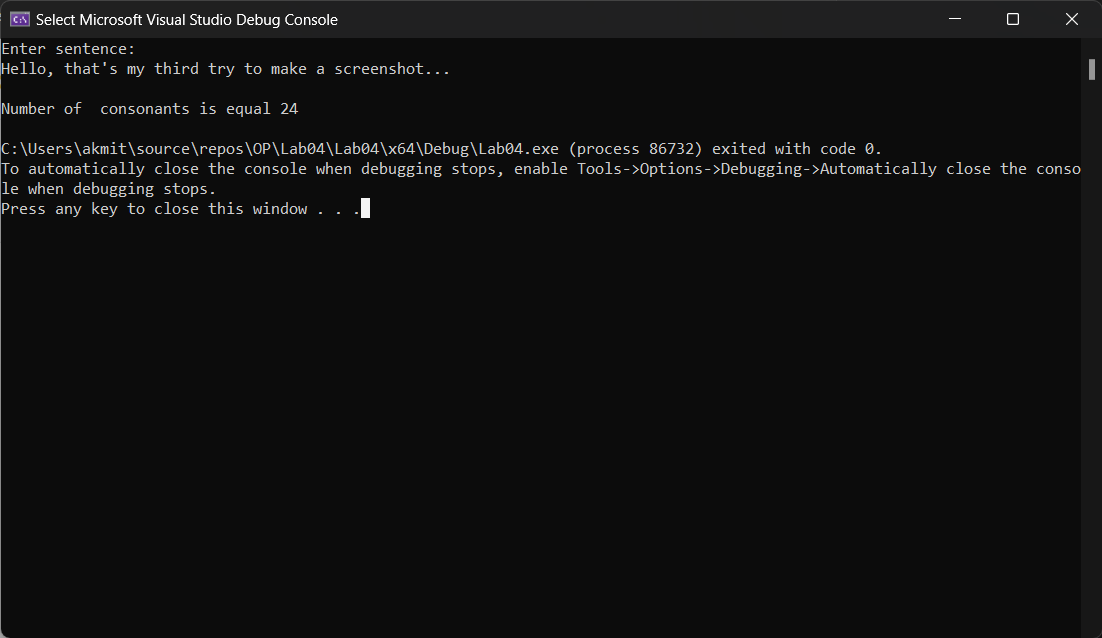


Рис.2 Результат виконання програми з індивідуального завдання.

**Лабораторне завдання. Пункт 7.**

**Код програми**

Назва файлу: **Lab04\_7.c**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <time.h>

int main(void) {

FILE\* file, \* logStream;

char readFrom[1000], writeInto[1000], logPath[] = "Log.txt", logStr[100], tempStr[100];

double xyz[3] = { 0 }, x, y, z, result = 0;

time\_t currentTime;

int onGoing = 0;

do {

onGoing = 0;

logStream = fopen(logPath, "a");

if (logStream != NULL) {

time(&currentTime);

strftime(logStr, 20, "%Y-%m-%d %H:%M:%S", localtime(&currentTime));

strcat(logStr, "]: Program has started");

fprintf(logStream, "[%s\n", logStr);

fclose(logStream);

onGoing = 0;

}

else {

logStream = fopen(logPath, "w");

fclose(logStream);

onGoing = 1;

}

} while (onGoing);

printf("Path of the file with start values: ");

scanf("%s", readFrom);

do {

onGoing = 0;

file = fopen(readFrom, "rb");

if (file != NULL) {

fread(xyz, sizeof(double), 3, file);

fclose(file);

logStream = fopen(logPath, "a");

if (logStream != NULL) {

time(&currentTime);

strftime(logStr, 20, "%Y-%m-%d %H:%M:%S", localtime(&currentTime));

sprintf(tempStr, "]: Values was read: X = %lf, Y = %lf, Z = %lf", xyz[0], xyz[1], xyz[2]);

strcat(logStr, tempStr);

fprintf(logStream, "[%s\n", logStr);

fclose(logStream);

}

onGoing = 0;

}

else {

logStream = fopen(logPath, "a");

if (logStream != NULL) {

time(&currentTime);

strftime(logStr, 20, "%Y-%m-%d %H:%M:%S", localtime(&currentTime));

strcat(logStr, "]: Incorrect input file path");

fprintf(logStream, "[%s\n", logStr);

fclose(logStream);

}

onGoing = 1;

printf("\nSorry, but this file doesn`t exist. Please, try again:\n");

scanf("%s", readFrom);

}

} while (onGoing);

x = xyz[0];

y = xyz[1];

z = xyz[2];

do {

if ((x - y <= 0 && z - (int)z != 0) || pow((x - y), z) + pow(x, 2) == 0) {

printf("\nSorry, but we can't perform this operation, because values of variables lie beyond the domain of a function\n");

logStream = fopen(logPath, "a");

if (logStream != NULL) {

time(&currentTime);

strftime(logStr, 20, "%Y-%m-%d %H:%M:%S", localtime(&currentTime));

strcat(logStr, "]: Error in calculating");

fprintf(logStream, "[%s\n", logStr);

fclose(logStream);

}

break;

}

result = (x + 3 \* (x - y) + pow(x, 2)) / (pow((x - y), z) + pow(x, 2));

printf("\nResult is %lf\n", result);

printf("\nPath of the target file: ");

scanf("%s", writeInto);

logStream = fopen(logPath, "a");

if (logStream != NULL) {

time(&currentTime);

strftime(logStr, 20, "%Y-%m-%d %H:%M:%S", localtime(&currentTime));

sprintf(tempStr, "]: Result was calculated successfully. Result = %lf", result);

strcat(logStr, tempStr);

fprintf(logStream, "[%s\n", logStr);

fclose(logStream);

}

char modeExtention[100], resStr[100];

while (1) {

printf("\nIn which format you want to save result? ('text' or 'binary'): ");

scanf("%s", modeExtention);

if (!(strcmp(modeExtention, "binary") && strcmp(modeExtention, "b"))) {

file = fopen(writeInto, "wb");

if (file != NULL) {

fwrite(&result, sizeof(double), 1, file);

fclose(file);

logStream = fopen(logPath, "a");

if (logStream != NULL) {

time(&currentTime);

strftime(logStr, 20, "%Y-%m-%d %H:%M:%S", localtime(&currentTime));

strcat(logStr, "]: Result was added as binary");

fprintf(logStream, "[%s\n", logStr);

fclose(logStream);

}

puts("\nResult was added as binary");

}

break;

}

else if (!(strcmp(modeExtention, "text") && strcmp(modeExtention, "t"))) {

file = fopen(writeInto, "wt");

if (file != NULL) {

fprintf(file, "%lf", result);

fclose(file);

logStream = fopen(logPath, "a");

if (logStream != NULL) {

time(&currentTime);

strftime(logStr, 20, "%Y-%m-%d %H:%M:%S", localtime(&currentTime));

strcat(logStr, "]: Result was added as text");

fprintf(logStream, "[%s\n", logStr);

fclose(logStream);

}

puts("\nResult was added as text");

}

break;

}

else {

logStream = fopen(logPath, "a");

if (logStream != NULL) {

time(&currentTime);

strftime(logStr, 20, "%Y-%m-%d %H:%M:%S", localtime(&currentTime));

strcat(logStr, "]: Incorrect input value for binary or text mode");

fprintf(logStream, "[%s\n", logStr);

fclose(logStream);

}

}

}

} while (0);

logStream = fopen(logPath, "a");

if (logStream != NULL) {

time(&currentTime);

strftime(logStr, 20, "%Y-%m-%d %H:%M:%S", localtime(&currentTime));

strcat(logStr, "]: End of the program\n");

fprintf(logStream, "[%s\n", logStr);

fclose(logStream);

}

return 0;

}

**Протокол виконання**

1. Користувач вводить шлях до бінарного файлу, з якого будуть взяті значення змінних X, Y, Z.
2. Якщо цього файлу не існує програма запропонує користувачеві ввести шлях знову.
3. Якщо значення з файлу не входять в область визначення функції, програма виведе попередження про помилку і закінчить своє виконання.
4. Якщо значення входять в область визначення, програма запропонує користувачеві ввести шлях до файлу, в який буде записаний результат.
5. Після цього користувачу буде запропоновано ввести формат, в якому буде записаний результат.
6. Програма відзвітує про виконання задачі.

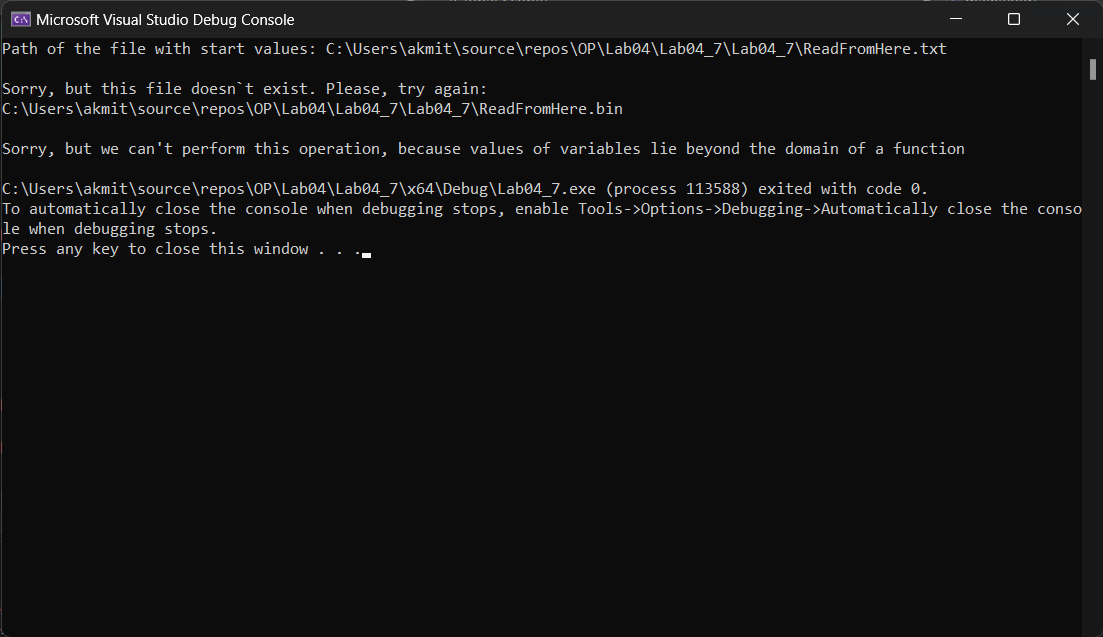


Рис.3 Результат виконання, перший виклик. Значенням змінних відповідають значення,

запропоновані в умові до першого додатку першої лабораторної роботи.

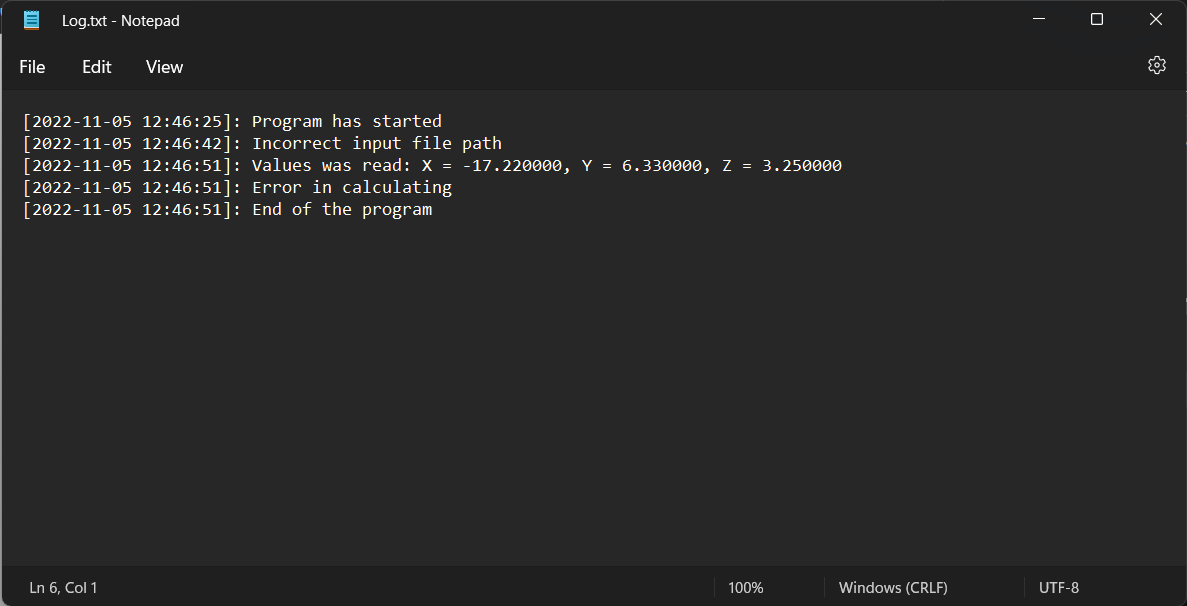


Рис.4 Log файл після першого виклику

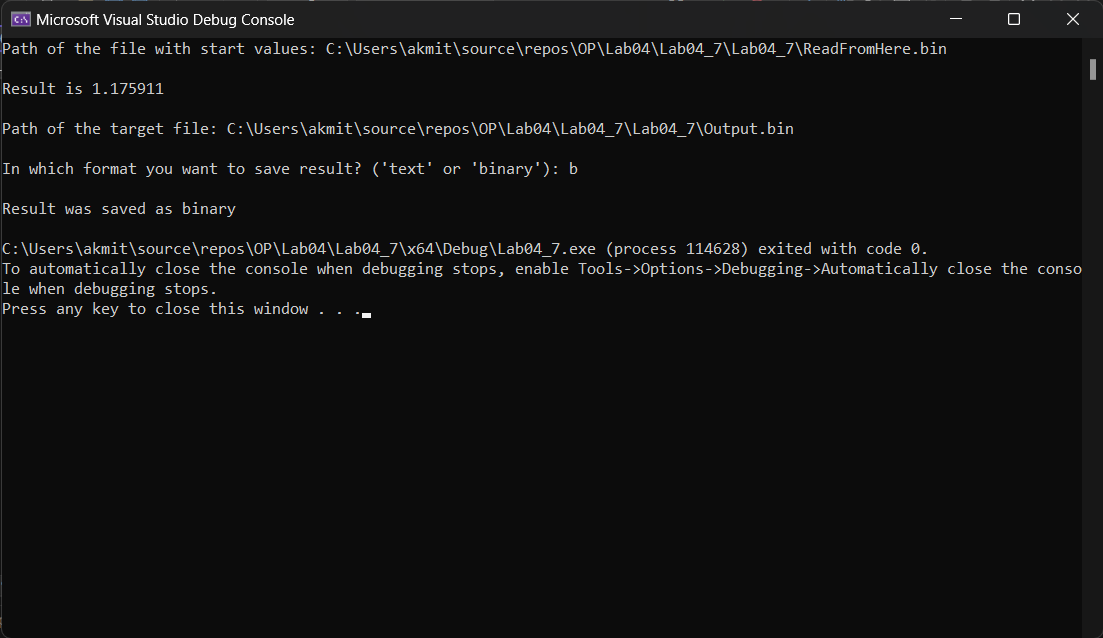


Рис.5 Результат виконання, другий виклик. Значення входять в

область визначення. Результат записано у бінарному форматі.

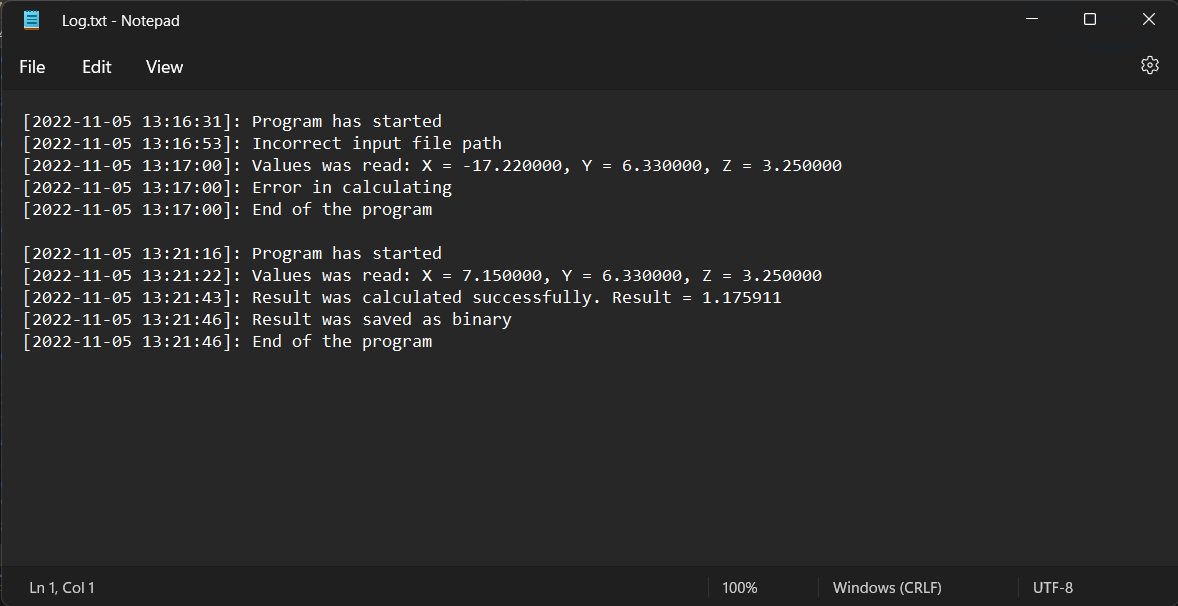
****

Рис.6 Log файл після другого виклику

**Висновок**

На лабораторній роботі я здобув практичні навички опрацювання текстової інформації з врахуванням особливостей організації символьних рядків у мові С. Навчився працювати з файлами, вивчив основні засоби потокового вводу/виводу в С.