# Лабораторна робота № 8

Тема. Файловий ввід-вивід. Робота з текстовими файлами. Функції обробки символьних рядків.

Мета. Формування навиків роботи:

1. з текстовими файлами засобами мови С++;
2. з функціями обробки символьних рядків засобами стандартної бібліотеки.

## Зміст роботи:

1. ознайомитися з теоретичними відомостями до лабораторної роботи;
2. проаналізувати завдання і розробити зовнішні специфікації до програми (вхідні та вихідні дані, функціональні вимоги до програми);
3. розробити модульну структуру програми;
4. розробити набори тестів для перевірки правильності виконання всіх модулів програми;
5. скласти алгоритми для розв’язання завдання всіх модулів програми;
6. розробити і налагодити програму, переконатися в правильності реалізації завдання за допомогою набору тестів;
7. скласти звіт з лабораторної роботи.

## Зміст звіту:

1. титульний аркуш (вид та номер роботи, дисципліна, П.І.Б. та група виконавця);
2. тема і мета лабораторної роботи;
3. постановка завдання (завдання до лабораторної роботи, індивідуальне завдання);
4. зовнішні специфікації програми (вхідні та вихідні дані, функціональні вимоги до програми);
5. модульна структура програми;
6. алгоритми для розв’язання завдання всіх модулів програми;
7. набори тестів для перевірки правильності виконання всіх модулів програми;
8. текст програми (всі файли проекту);
9. результати тестування програми та їх аналіз;
10. висновки щодо роботи з текстовими файлами і обробки символьних рядків.

## Теоретичні відомості

### Загальні відомості про файли

Файл – це іменований блок інформації, який зберігається на носії інформації і має такі ознаки: фіксоване ім'я, певне логічне представлення і відповідні йому операції читання/запису. В програмі файли пов’язані с потоками. Потоком називається програмний інтерфейс, який забезпечує доступ до файлу. Кожна комп'ютерна програма відкриває принаймні три файли стандартних потоків: вхідний файл (stdin), вихідний файл (stdout), файл виводу повідомлень про помилки (stderr). Вхідний файл зазвичай асоціюється із клавіатурою, вихідний файл та файл виводу помилок – із екраном терміналу, проте кожен із цих файлів може бути перенаправлений на інший носій інформації, наприклад, на дисковий файл, на мережу, яка також є файлом, на контролер будь-якого периферійного пристрою комп'ютера, зчитувача інформації тощо. Файлом може бути також ділянка оперативної пам'яті програми.

Під доступом до файлу розуміють запис і читання даних з файлу. Перш ніж використовувати потік для доступу до файлу, його необхідно з'єднати з цим файлом, т. е. забезпечити потік інформацією про файл. Ця інформація зберігається в структурі типу FILE. Тому вважається, що потік має тип FILE\*, т. е. є вказівником. Коли потік з'єднують з файлом, то кажуть, що файл відкривають. Коли потік від'єднують від файлу, то кажуть, що файл закривають.

Кожен потік може працювати в двох режимах: текстовому і бінарному. Режим роботи потоку задається при його поєднанні з файлом.

Текстові файли призначено для зберігання текстів, тобто сукупності символьних рядків змінної довжини. У текстовому режимі потік записує і читає з файлу текстові рядки, які закінчуються керуючою послідовністю. Різні операційні системи дотримуються свого подання кінця рядка і кінця файлу. В UNIX кінець рядка складається з одного символу \n (перехід на новий рядок, код 0xA), в Mac OS – з символу \r (повернення каретки, код 0xD), а в DOS і Windows кінець рядка кодується послідовністю двох символів: \r\n. При записі в текстовий потік комбінації символів \r (повернення каретки) і \n (перехід на новий рядок) в файл записується тільки символ \n. А при читанні з текстового потоку символ \n навпаки перетворюється в комбінацію символів \r\n. Розділювачами слів та чисел у рядку є пробіли й символи табуляції. Оскільки вся інформація текстового файлу є символьною, програмне опрацювання такого файлу полягає в читанні рядків, виокремленні з рядка слів і, за потреби, перетворюванні цифрових символьних послідовностей на числа відповідними функціями перетворювання. За стандартом потік повинен забезпечувати обробку рядків довжиною не менше 254 символу, включаючи символ \n. Стандартом допускається, що при читанні і запису даних текстовим потоком може відбуватися їх перетворення. Створювати, редагувати текстові файли можна не лише в програмі, а й у якому завгодно текстовому редакторі.

При роботі в бінарному режимі потік записує на диск точні копії даних з оперативної пам'яті і зчитує в оперативну пам'ять точні копії даних з диска, тобто записує і читає дані з файлу в тому вигляді, в якому вони зберігаються в оперативній пам’яті.

Робота з файлами в С++ може виконуватися кількома різними способами:

* робота з файлами як з потоками у стилі С;
* робота з файлами як з потоками у стилі С++.

### Робота з файлами у стилі С

### Оголошення файлової змінної

Інформація про файл заноситься до змінної типу FILE\*. Цей тип оголошує вказівник потоку, який використовується надалі у всіх операціях з цим файлом. Тип FILE означено у бібліотеці <stdio.h>. Тому, якщо в програмі передбачається робота з файлами, цей заголовний файл слід долучити до програми: #include <stdio.h>

Тепер можна оголосити файлову змінну – вказівник потоку, який в подальшому буде передаватися у функції введення-виведення у якості параметра:

FILE\* stream;

### Відкриття файлу

Для з’єднання потоку з файлом використовується функція fopen, яка має прототип:

errno\_t fopen\_s( FILE\*\* pFile, const char \*filename, const char \*mode );

Ця функція відкриває файл, ім’я якого задається параметром filename, у режимі, який визначається параметром mode (табл. 1). Параметр pFile – вказівник на файловий вказівник, який отримає вказівник на відкритий файл.

Функція повертає нуль у разі успіху або код помилки у разі невдачі. Коди помилок можна знайти у документації.

Таблиця 1

Специфікатори режиму відкривання файлів у текстовому режимі

|  |  |
| --- | --- |
| Параметр | Опис |
| r  r+  а  a+  w  w+ | відкрити існуючий файл для зчитування даних  відкрити існуючий файл для зчитування й записування даних  створити новий чи відкрити існуючий файл для записування даних у кінець файлу  створити новий чи відкрити існуючий файл для зчитування й записування даних у кінець файлу  створити новий файл для записування даних  створити новий файл для зчитування й записування даних |

При відкритті текстового файлу в режимах "r" або "r+" (для бінарних "rb", "rb+" або "r+b") його індикатор позиції встановлюється на початок файлу. У випадку якщо відкривається неіснуючий файл, то функція fopen\_s закінчується невдачею.

При відкритті текстового файлу в режимах "a" або "a+" (для бінарних "ab", "ab+" або "a+b") створюється новий файл. Якщо файл із заданим ім'ям існує, то він відкривається, і індикатор позиції встановлюється на кінець файлу.

При відкритті текстового файлу в режимах "w" або "w+" (для бінарних "wb", "wb+" або "w+b") створюється новий файл. Якщо файл із заданим ім'ям існує, то його вміст стирається, а індикатор позиції встановлюється на початок файлу.

При відкритому файлі в режимі оновлення ('+' в якості другого або третього символу аргументу позначення режиму) і введення і виведення можуть виконуватися в одному потоці. Проте, запис бути не може слідувати за читанням без проміжного виклику функції fflush (функція роботи з буфером вводу-виводу) або функції позиціонування в файлі (fseek, fsetpos або rewind), а читання не може слідувати за записом без проміжного виклику функції позиціонування в файлі.

### Закриття файлу

Для від'єднання потоку від файлу використовують функцію fclose. Для закриття всіх файлів одночасно використовують функцію \_fcloseall. Прототипи цих функцій:

int fclose( FILE\* stream );

int \_fclose( void );

При успішному завершенні функція fclose повертає 0, а функція \_fclose – кількість закритих файлів. У разі невдачі обидві повертають значення EOF (End Of File, кінець файлу).

### Індикатор кінця файлу

Структура FILE містить індикатор кінця файлу, який встановлюється в ненульове значення функцією читання з файлу досягши цією функцією кінця файлу. Стан кінця файлу зчитується функцією feof, яка має прототип:

int feof( FILE\* stream );

Ця функція повертає ненульове значення, якщо індикатор кінця файлу встановлений, інакше функція повертає 0.

### Вказівник позиції файлу

Для кожного файлу, після його відкриття, визначається вказівник позиції, який вказує на зміщення від початку файлу у байтах. Тип вказівника позиції зазвичай визначається як

typedef long fpos\_t;

Для роботи з вказівник позиції призначені функції fseek, fsetpos, ftell і fgetpos.

Функція fseek має прототип:

int fseek( FILE\* stream, long offset, int mode );

Ця функція пересуває вказівник позиції файлу на offset байтів. У разі успішного завершення функція повертає 0, інакше – ненульове значення. Параметр mode вказує на режим пересування і може набувати значень, визначених такими символічними константами:

SEEK \_ SET – зміщення від початку файлу;

SEEK \_ CUR – зміщення від поточної позиції;

SEEK \_ END – зміщення від кінця файлу.

При роботі з текстовим потоком повинні використовуватися тільки наступні комбінації значень параметрів:

mode = SEEK \_ SET, а offset дорівнює 0 або значенню, поверненому функцією ftell;

mode = SEEK \_ CUR і offset = 0;

mode = SEEK \_ END і offset = 0.

Функція fsetpos має прототип:

int fsetpos( FILE \* stream, const fpos\_t \* pos);

Ця функція встановлює індикатор позиції файлу stream в позицію, на яку вказує параметр pos. Індикатор кінця файлу скидається. У разі успіху функція повертає 0, а в випадку невдачі – нульове значення і встановлює змінну errno.

Функція ftell має прототип:

long ftell( FILE\* stream );

Ця функція у разі успішного завершення повертає поточну позицію файлу stream, а у разі невдачі – повертає значення -1L і встановлює значення змінної errno. Для бінарного потоку позиція дорівнює зміщенню у байтах від початку файлу, а у разі текстового потоку – значенню, яке може використовуватися функцією fseek.

Функція fgetpos має прототип:

int fgetpos( FILE\* stream, fpos\_t\* pos );

Ця функція записує поточну позицію файлу stream за адресою pos. У разі успіху функція повертає 0, а у разі невдачі – ненульове значення і встановлює значення змінної errno.

### Читання з файлу і запис у файл

### Блочний ввід-вивід

Блоком називається область оперативної пам'яті, вміст якої записується у файл. Введення/виведення блоками використовується головним чином при роботі з бінарними потоками. Для запису блоку у файл використовується функція fwrite, яка має прототип:

size\_t fwrite( const void\* ptr, size\_t size, size\_t nitems, FILE\* stream );

Ця функція записує вміст блоку пам'яті, на який вказує параметр ptr, у файл stream. Довжина записуваного блоку визначається як добуток size \* nitems. Функція повертає кількість записаних одиниць пам'яті. У разі удачі це число має бути рівне nitems. У разі невдачі функція fwrite встановлює індикатор помилки.

Для читання блоку з файлу використовується функція fread, яка має прототип:

size\_t fread( const void\* ptr, size\_t size, size\_t nitems, FILE\* stream );

Параметри цієї функції мають той же сенс, що і параметри функції fwrite. Функція повертає кількість прочитаних одиниць пам'яті. У разі удачі це число має бути рівне nitems. У разі невдачі функція fwrite встановлює індикатор помилки або індикатор кінця файлу. У разі помилки стан індикатора позиції після роботи функцій fwrite і fread не визначений. Інакше індикатор позиції пересувається на кількість записаних і прочитаних байтів.

### Ввід-вивід символів

Для запису і читання символів (байтів) з файлу використовуються функції fputc і fgetc. Функція fputc має прототип:

int fputc( int c, FILE\* stream );

Ця функція записує символ, заданий параметром c, в потік stream і пересуває вказівник позиції на наступний символ. У разі успіху функція повертає символ c, а у разі невдачі – значення EOF і встановлює індикатор помилки.

Функція fgetc має прототип:

int fgetc( FILE\* stream );

Ця функція читає символ з потоку stream і пересуває вказівник позиції на наступний символ. У разі успішного завершення функція повертає прочитаний символ. У разі досягнення кінця файлу функція повертає значення EOF і встановлює індикатор кінця файлу. У разі помилки функція повертає значення EOF і встановлює індикатор помилки.

### Запис/читання рядків з файлу

Для запису/читання рядка з файлу в мові С призначені функції fputs і fgets. Ці функції використовуються головним чином при роботі з символьними потоками. Слід звернути увагу на те, що тип FILE\* перейшов в C++ зі стандартного С, і такий файл «розуміє» лише С-рядки, тобто рядки типу char\*.

Функція fputs має прототип:

int fputs( const char\* str, FILE\* stream );

Ця функція записує рядок str у файл stream, не включаючи завершальний нульовий байт (\0). У разі успішного завершення функція повертає ненульове число, а у разі невдачі – EOF.

Функція fgets має прототип:

int fgets( char\* str, int n, FILE\* stream );

Ця функція читає рядок з потоку stream в рядок str. Зупиняється функція у разі, якщо прочитаний (n - 1) символ, або зустрівся символ \n, або виявлений кінець файлу. У будь-якому з цих випадків в кінець рядка поміщається символ \n, за яким записується порожній символ \0. У разі успіху функція повертає вказівник str, а у разі невдачі – NULL. Якщо не прочитаний жоден символ і виявлений кінець файлу, то рядок str не змінюється, а функція повертає значення NULL.

### Форматований ввід-вивід

Для форматованого введення даних використовується функція fscanf, яка має прототип:

int fscanf( FILE\* stream, const char\* format ... );

Ця функція виконує читання даних з файлу stream відповідно до форматного рядка format. Параметр format повинен вказувати на рядок, що форматується, який містить специфікації форматів введення. Кількість і типи аргументів, які слідують після рядка форматування, повинні відповідати кількості і типам форматів введення, заданим в рядку форматування. Якщо ця умова не виконується, то поведінка функції не визначена. Крім того, аргументи повинні задавати адреси змінних, в які вводитимуться дані.

Для форматованого виведення даних у файл призначена функція fprintf, яка має прототип:

int fprintf ( FILE\* stream, const char\* format ... );

Ця функція виконує запис даних у файл stream відповідно до форматного рядка format. Параметр format є рядком, що форматується, який містить специфікації форматів виведення. Кількість і типи аргументів, які слідують після рядка форматування, повинні відповідати кількості і типам специфікацій формату виведення, заданим в рядку форматування. Якщо ці умови не виконуються, то поведінка функції не визначена.

Детальніше про функції fscanf і fscanf та специфікації форматів введення-виведення читайте в документації.

Хоча fprintf () u fscanf () часто є найбільш простим способом запису і читання даних з файлу, вони далеко не завжди є найбільш ефективним рішенням. Оскільки відформатовані ASCII-дані зберігаються так само, як вони виглядають на екрані (а не в двійковому поданні в оперативної пам’яті), з'являються додаткові витрати при кожному виклику. Якщо швидкість роботи програми або розмір створюваних файлів грають важливу роль, то, можливо, краще використовувати fread () і fwrite ().

### Видалення файлу

Для видалення файлу призначена функція remove, яка має прототип:

int remove( const char\* filename );

Ця функція видаляє файл з ім'ям filename. Якщо файл відкритий, то робота функції залежить від реалізації. У разі успішного завершення функція повертає 0, а у разі невдачі – ненульове значення.

### Перейменування файлу

Для перейменування файлу призначена функція rename, яка має прототип:

int rename( const char\* old\_name, const char\* new\_name);

Ця функція перейменовує файл з ім'ям old\_name у файл з ім'ям new\_name. Якщо файл з ім'ям new\_name вже існує, то робота функції залежить від реалізації. У разі успішного завершення функція повертає 0, а у разі невдачі – ненульове значення.

### Робота з тимчасовими файлами

Для створення тимчасового файлу призначена функція tmpfile, яка має прототип:

FILE\* tmpfile( void );

Ця функція створює тимчасовий файл і відкриває його в режимі "w+b". Після закриття потоку тимчасовий файл віддаляється. У разі успішного завершення функція повертає покажчик на файл, а у разі невдачі – NULL.

### Символьні рядки у стилі С

Символьні рядки можуть зберігати яку завгодно символьну інформацію, приміром: імена файлів, назви книг, імена працівників та інші символьні сполучення. С++ підтримує кілька типів рядків – масиви символів, що перейшли від С (так звані С-рядки), і клас string стандартної бібліотеки С++. Останній є доволі зручний у застосуванні, однак на практиці нерідкі є ситуації, коли виникає потреба у користуванні вбудованим типом (С-рядками).

Майже всі різновиди рядків у С являють собою послідовність (масив) символів із завершальним нуль-символом. Нуль-символ (нуль-термінатор) – це символ з кодом 0, який записується у вигляді керуючої послідовності '\0'. За розташуванням нуль-символу визначається фактична довжина рядка.

Відмінною рисою символьного масиву є те, що в ньому насправді може бути менше символів, аніж зазначено при оголошенні. Окрім того, з цими масивами можна виконувати певні специфічні дії, які не можна здійснювати з числовими масивами (наприклад перевіряти наявність у масиві літери чи послідовності літер, копіювати масив як одне ціле, порівнювати масиви за алфавітом, дописувати один масив наприкінці іншого тощо).

Пам’ять під розміщення рядків, як і для будь-яких масивів, може виділятися як компілятором, так і динамічно – при виконуванні програми. Довжина динамічного рядка може задаватися змінною з визначеним заздалегідь значенням, а довжина статичного рядка має задаватися лише константою.

Рядок може бути оголошеним в один з нижче наведених способів:

1) char \*s; // Оголошення вказівника на перший символ рядка;

// пам’ять під сам рядок не виділяється

2) char ss[15]; // Оголошення рядка ss з 14-ти символів;

// пам’ять виділяється компілятором

3) const int n = 10;

char st[n]; // Оголошення рядка st з n-1 (тобто 9-ти) символів;

// пам’ять виділяється компілятором

4) int n = 10;

char \*str = new char[n]; // Оголошення рядка str з n-1 (тобто 9) символів;

// пам’ять виділяється динамічно

При зазначенні довжини рядка слід враховувати завершальний нуль-символ. Наприклад, у вищенаведеному рядку str можна зберігати не 10, а лише 9 символів.

Зауважимо, що при оголошенні рядка першим способом пам’ять під рядок не виділяється і це може бути дуже небезпечним, оскільки до тієї самої ділянки пам’яті може бути розміщено інші змінні й рядок буде втрачено.

При оголошенні рядок можна ініціалізувати рядковою константою, при цьому нуль-символ формується автоматично після останнього символу:

char str[10] = "Hello!";

При цьому виділиться пам’ять під масив з 9-ти елементів та 10-й – нуль-символ (всього 10 байт) і перші 6 символів рядка записуються в перші 6 байт цієї пам’яті (str[0]='H', str[1]='e', str[2]='l', str[3]='l', str[4]='o', str[5]='!'), а в сьомий елемент str[6] записується нуль-символ. Якщо рядок при оголошенні ініціалізується, його розмірність можна опускати (компілятор сам виділить потрібну кількість байтів):

char str[] = "Hello!"; // Виділено й заповнено 7 байтів

Рядки у лапках завжди неявно містять нуль-символ, тому при ініціалізації прописувати його немає потреби. Окрім того, засоби введення символьних масивів автоматично долучають нуль-символ у кінець масиву.

При оголошенні й ініціалізації масиву слід бути впевненим, що розмір масиву є достатній, щоб умістити всі символи рядка з нуль-символом. Річ у тім, що функції, які опрацьовують рядки, керуються позицією нуль-символу, а не розміром рядка. С++ не накладає жодних обмежень на довжину рядка.

Звернімо увагу на те, що рядкова константа (у подвійних лапках) і символьна константа (в одинарних лапках) не є взаємозамінними. Це константи різних типів. Символ у одинарних лапках, наприклад, 's' є символьною константою. Для зберігання такої константи компілятор C++ виділяє лише один байт пам’яті. Символ у подвійних лапках, наприклад, "s" є рядковою константою, що окрім символу 's' містить символ '\0', який долучається компілятором. Більш того, "s" фактично являє собою адресу пам’яті, в якій зберігається рядок.

Як і числові масиви, символьні масиви опрацьовуються поелементно у циклі. Операція присвоювання одного рядка іншому є невизначена (оскільки рядок є масивом) і може виконуватися за допомогою циклу чи за допомогою функцій стандартної бібліотеки.

Поширеним засобом доступу до символів рядка є вказівники типу char\*. У прикладі

char \*st = "Комп’ютерна програма";

компілятор записує всі символи рядка до масиву і присвоює змінній st адресу першого елемента масиву.

Рядок може вважатися за порожній у двох випадках: якщо вказівник на рядок має нульове значення nullptr (немає взагалі жодного рядка) чи коли вказівник вказує на масив, який складається з одного нульового символу (не містить жодного значущого символу).

char\* pc1 = nullptr; // pc1 не адресує жодного масиву символів,

const char\* pc2 = ""; // pc2 адресує нульовий символ

### Функції стандартної бібліотеки для роботи з рядками

С++ має багату колекцію функцій опрацювання рядків із завершальним нулем. Якщо в рядку відсутній нуль-термінатор, опрацювання рядка може тривати скільки завгодно, допоки в пам’яті не зустрінеться '\0'. У якості аргументів до функцій переважно передаються вказівники на рядки. Якщо при виконуванні функції здійснюється перенесення символів рядка з місця-джерела до місця-призначення, для рядка-призначення слід завчасно зарезервувати місце в пам’яті. Копіювання рядків з використанням просто вказівника, а не адреси початку завчасно оголошеного масиву – одна з найпоширеніших помилок програмування, навіть у досвідчених програмістів. При виділенні місця для рядка-призначення слід виділяти місце і для нуль-термінатора.

Для обробки символьних типів даних бібліотека функцій string.h має велику кількість вбудованих функцій, що збіль­шують продуктивність праці програмістів та скорочують час на розробку програм, наприклад:

функції перевірки символів;

функції перетворення символів;

функції перевірки рядків;

функції маніпулювання рядками.

В таблиці 2 і таблиці 3 функції наводяться у вигляді списків, що згруповані за їх розташуванням у заголовних файлах. Ці функції можна знайти в документації та ознайомитись з правилами їх використання на прикладах, які там наведені.

Таблиця 2

Функції класифікації і перетворення символів (заголовний файл <ctype.h> (<cctype>))

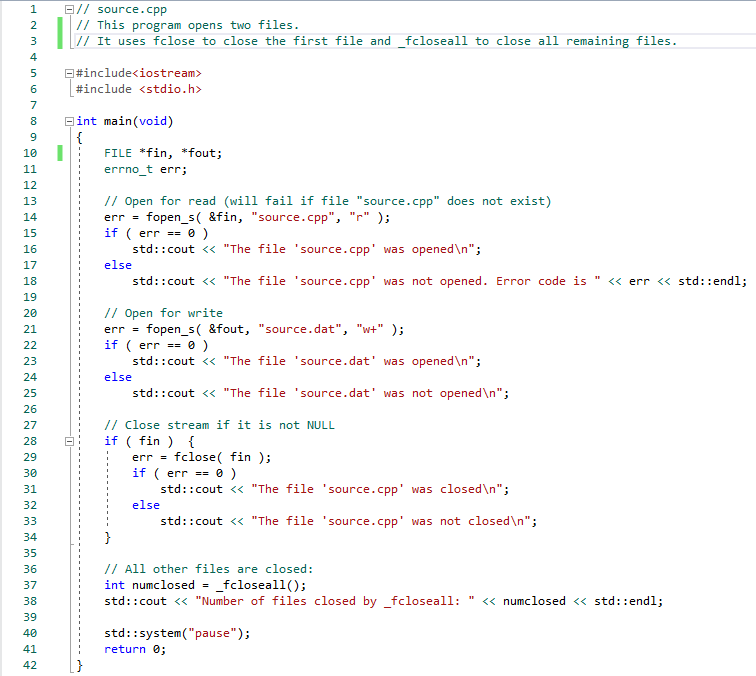
|  |
| --- |
|  |

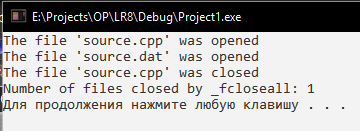
Таблиця 3

Функції роботи з рядками у стилі С (заголовний файл <string.h> (<cstring>))

|  |
| --- |
|  |

Приклад програми.





## Завдання

Розробити багатофайлову програму для роботи з текстовими файлами для заданого індивідуального завдання і обраного рівня складності.

Вимоги до програми:

* для обробки файлів використати рядки в стилі С;
* максимально задіяти функції стандартної бібліотеки (забороняється дублювати дії функцій стандартної бібліотеки власними функціями);
* імена файлів ввести з клавіатури;
* передбачити перевірку всіх операцій роботи з файлами (відкриття, читання, запису);
* управління виконанням програми здійснити на основі текстового меню користувача;
* передбачити обробку необмеженої кількості файлів;
* результати роботи програми вивести у файл та продублювати на екрані.

Вимоги до тексту програми:

* специфікації програми і її функцій;
* коментарі щодо призначення блоків програми, дій окремих операторів для пояснення алгоритму;
* самодокументованість коду: всі ідентифікатори повинні мати назви, що відповідають суті змінних.

### Індивідуальні завдання

Рівень I (60-74 балів):

* + 1. Для кожного рядка текстового файлу підрахувати скільки разів в ньому зустрічається символ А і скільки разів – символ Б, замінюючи їх один одним. Результати обробки кожного рядка записати у файл у вигляді значень двох лічильників і оновленого рядка. Символи А і Б задаються користувачем.
    2. Для кожного рядку текстового файлу сформувати новий рядок, в якому замінити крапки на багатокрапки, та підрахувати кількість замін. Результати обробки кожного рядка записати у файл у вигляді значення лічильника замін і нового рядка.
    3. Для кожного рядка текстового файлу підрахувати скільки разів в ньому зустрічається пара символів А і Б, що стоять поруч. Результати обробки кожного рядка записати у файл у вигляді значень двох лічильників и обробленого рядка. Символи А і Б задаються користувачем.
    4. Кожен рядок текстового файлу обробити за правилом: сформувати новий рядок, видаливши всі символи, які не є цифрами, і підрахувати кількість видалених символів. Результати обробки кожного рядка записати у файл у вигляди кількості видалених символів і нового рядка.
    5. Для кожного рядка текстового файлу сформувати новий рядок, в якому подвоїти кількість пробілів між словами. Підрахувати кількість пробілів в новому рядку. Результати обробки кожного рядка записати у файл у вигляді значення лічильника пробілів і нового рядка.
    6. Кожен рядок текстового файлу обробити за правилом: сформувати новий рядок, в який записати символи, які містяться між двома двокрапками; якщо другої двокрапки немає, то записати всі символи, які містяться після єдиної наявної двокрапки; якщо в рядку двокрапок немає, залишити новий рядок порожнім. Для кожного нового рядка підрахувати кількість символів. Результати обробки кожного рядка записати у файл у вигляді лічильника символів і нового рядка.
    7. В кожному рядку текстового файлу підрахувати найбільшу кількість пробілів, що йдуть підряд. Результати обробки кожного рядка записати у файл у вигляді значення найбільшої кількості пробілів і обробленого рядка.
    8. З кожного рядка текстового файлу переписати в новий рядок групи символів, які містяться між двома дужками (,). Якщо парних дужок в рядку немає, то новій рядок залишається порожнім. Результати обробки кожного рядка записати у файл у вигляди кількості символів, що залишились не переписаними, і нового рядка.
    9. В кожному рядку текстового файлу, якій містить цифри і англійськи символи замінити прописні символи на рядкові і навпаки та підрахувати кількість замін. Результати обробки кожного рядка записати у файл у вигляді значення лічильника замін і оновленого рядка.
    10. Кожен рядок текстового файлу обробити за правилом: сформувати новий рядок, видаливши всі символи, які не є літерами, і підрахувати кількість видалених символів. Результати обробки кожного рядка записати у файл у вигляди кількості видалених символів і нового рядка.

Рівень II (75-89 балів):

1. Для кожного рядка текстового файлу підрахувати скільки разів в ньому зустрічається задане користувачем слово. Результати обробки кожного рядка записати у файл у вигляді значення лічильника, шуканого слова і обробленого рядка.
2. Для кожного рядка текстового файлу підрахувати скільки разів в ньому зустрічаються слова, які починаються і закінчуються однією і тією ж літерою. Результати обробки кожного рядка записати у файл у вигляді значення лічильника і шуканих слів.
3. Для кожного рядка текстового файлу знайти найкоротше слово. Результати обробки кожного рядка записати у файл у вигляді шуканого слова і його довжини.
4. Для кожного рядка текстового файлу сформувати новий рядок, в якому видалити всі пробіли (їх між словами може бути декілька) і між словами поставити крапку. Наприкінці рядка теж поставити крапку. Підрахувати кількість видалених пробілів в рядку. Результати обробки кожного рядка записати у файл у вигляді значення кількості видалених пробілів і оновленого рядку.
5. Для кожного рядка текстового файлу підрахувати скільки разів в ньому зустрічаються слова, які містять рівно три літери «А» (без врахування регістру). Результати обробки кожного рядка записати у файл у вигляді значення лічильника і шуканих слів.
6. Для кожного рядка текстового файлу сформувати новий рядок із слів, які розділяються одним пробілом і розташовані в у зворотному порядку. Підрахувати кількість видалених пробілів. Результати обробки кожного рядка записати у файл у вигляді значення лічильника видалених пробілів і нового рядка.
7. Кожен рядок текстового файлу перетворити так, щоби всі слова в ньому починалися зі рядкової літери і містили між собою один пробіл. Результати обробки кожного рядка записати у файл у вигляді значення довжини обробленого рядка і нового рядка.
8. Для кожного рядка текстового файлу сформувати новий рядок із пар слів, які поміняти місцями. Останнє слово в новий рядок не записувати, якщо йому немає пари. Підрахувати кількість пар слів в новому рядку. Результати обробки кожного рядка записати у файл у вигляді значення лічильника пар слів і нового рядка.
9. Для кожного рядка текстового файлу сформувати новий рядок із слів-паліндромів. Підрахувати кількість слів в новому рядку. Результати обробки кожного рядка записати у файл у вигляді значення лічильника слів і нового рядка.
10. Для кожного рядка текстового файлу сформувати новий рядок, в якому циклічно зсунути слова зліва направо на одне слово і видалити зайві пробіли. Результати обробки кожного рядка записати у файл у вигляді значення довжини нового рядка і самого нового рядка.

Рівень III (90-100 балів):

Завдання. Програма обробки текстового файлу по реченням. Реченням вважається довільна послідовність слів, яка обмежена крапкою, після якої йде рядкова літера або кінець файлу. Попередньо кожне речення слід переформатувати так, щоб воно починалося з окремого рядка і розташовувалося в рядках розміром не більше 100 символів. Виконати один з варіантів завдань роботи з текстовими файлами.

1. Програма  порівняння двох  файлів по реченнями з виводом груп речень, які додані або видалені з другого файлу відносно першого. (+бонус)
2. Програма складає словник термінів. Кожен термін – слово, яке записане рядковими літерами. Крім того, програма дозволяє переглядати текст в обох напрямках по рядках і при виборі рядка, шукає к ньому термін і печатає його. (+бонус)
3. Програма складає словник слів тексту. Для обраного слова в новий файл записуються речення, в яких це слово міститься. (+бонус)
4. Редактор тексту з командами додавання і видалення речень, з прокруткою тексту в обох напрямках. (+бонус)
5. Програма складає для файлів словники з частотою включення слів. Після обробки файлу в новий файл записуються речення з трьома словами, що зустрічаються найчастіше у файлі. (+бонус)
6. Програма розташовує в реченні слова по збільшенню їх довжини.
7. Програма сортує в реченні слова за алфавітом.
8. Програма перевіряє чи правильно записані в реченнях числа. Речення, в яких знайдена помилка у запису числа, виключаються з файлу.
9. Програма виводить на екран і записує в файл будь-який блок з n-го по m-е речення. Значення n і m вказує користувач.
10. Програма перегляду текстового файлу по абзацах. Абзацом вважається будь-яка послідовність речень обмежена порожнім рядком. Програма виводить на екран і записує в файл будь-який абзац за вказаним номером.