Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

ЗВІТ

ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 10

з навчальної дисципліни

“Базові методології та технології програмування”

РЕАЛІЗАЦІЯ ПРОГРАМНИХ МОДУЛІВ ОБРОБЛЕННЯ ДАНИХ СКЛАДОВИХ ТИПІВ З ФАЙЛОВИМ ВВЕДЕННЯМ/ВИВЕДЕННЯМ

ЗАВДАННЯ ВИДАВ

доцент кафедри кібербезпеки та програмного забезпечення

Доренський О. П.

[https://github.com/odorenskyi/](https://github.com/odorenskyi/Dmytro-Parkhomenko-KB18)

ВИКОНАВ

студент академічної групи КБ-21

Іванов М.Л.

ПЕРЕВІРИВ

ст. викладач кафедри кібербезпеки   
та програмного забезпечення

Усік П.С.

Кропивницький – 2022

**Лабораторна робота №10**

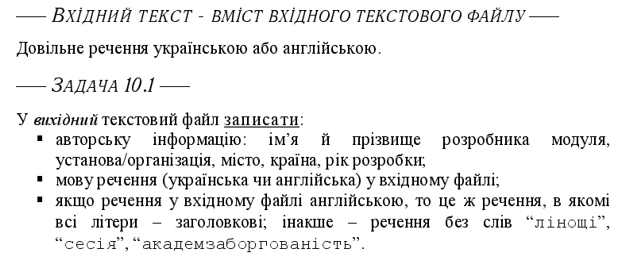
**Тема:** реалізація програмних модулів оброблення даних складових типів з файловим введенням/виведенням

**Мета роботи:** полягає у набутті ґрунтовних вмінь і практичних навичок реалізації у Code::Blocks IDE мовою програмування C++ програминх модулів створення й оброблення даних типів масив, структур, об'єднання, множина, перелік, перетворення типів даних, використання файлових потоків та функцій стандартних бібліотек для оброблення символьнох інформації.

**Завдання:**

1. Реалізувати програмні модулі розв'язування задач 10.1-10.3 як складових статичної бібліотеки libModulesПрізвище.a (проект ModulesПрізвище, створений під час виконання лабораторних робіт №8-9).
2. Реалізувати тестовий драйвер автоматизованої перевірки програмних модулів розв'язування задач 10.1-10.3.

**Варіант** **12**

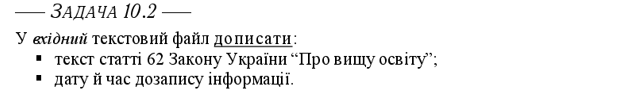


Малюнок 1 – Умова задачі 10.1

**Строга постановка задачі:**

Вхідні дані: path, a – рядки.

Вихідні дані:

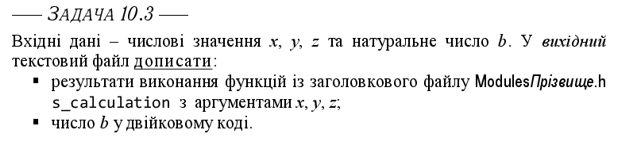


Малюнок 2 – Умова задачі 9.2

**Строга постановка задачі:**

Вхідні дані: path – рядок

Вихідні дані: –.



Малюнок 3 – Умова задачі 9.3

**Строга постановка задачі:**

Вхідні дані: path – рядок,

x, y, z – дійсні числа,

b – ціле число.

Вихідні дані:–.

**Проектування програмного модуля:**

Модуль ModulesIvanov, що вже містить функцію s\_calculation, розширюється шляхом додавання нових функцій та структур даних, призначених для реалізації повернення декількох змінних функціями.

Функція addSmth приймає два рядки, що позначають назву вхіного та вихідного файлів. Із вхідного файлу за допомогою ifstream зчитується зміст (речення). Після цього за допомогою ofsteam відкривається вихідний файл, увесь зміт якого знищуються (якщо файлу не існує – віг створюється). У нього записується інформація про розробника, речення із вхідного файлу з визначеними символами заміненими на «?», кількість літер у рядку зі змістом вхідного файлу та в залежності віл того, парна вона чи ні – відповідний вірш.

Функція PrintRule62 також приймає два рядки, що позначають назву вхіного та вихідного файлів. Із вхідного файлу зчитується зміст, після чого у кінець вихідного файлу дописуються непарні символи (кожний другий, починаючи з найпершого). Після цього дописується дата дозапису.

Функція CalcBin приймає рядок з назвою вихідного файлу, 3 дійсеих числа та ціле число. Ця функція дописує у кінець вихідного файлу результат функції s\_calculations зі змінними x, y та z в якості аргументів та число b у двійковому вигляді.

**Результати тестування модулів для задач 10.1-10.3:**

**TestDriver of ModulesIvanov.a**

**====================================**

**|Тестовий драйвер для завдання 10.1|**

**====================================**

**Визначення мови... Done**

**Привітання, Написання... Done**

**Зміна тексту... Done**

**TC-1 passed**

**Визначення мови... Done**

**Привітання, Написання... Done**

**Зміна тексту... Done**

**TC-2 passed**

**Визначення мови... Done**

**Привітання, Написання... Done**

**Зміна тексту... Done**

**TC-3 passed**

**====================================**

**|Тестовий драйвер для завдання 10.2|**

**====================================**

**Написання закону... Done**

**TC-1 passed**

**Написання закону... Done**

**TC-2 passed**

**Написання закону... Done**

**TC-3 passed**

**====================================**

**|Тестовий драйвер для завдання 10.3|**

**====================================**

**Підрахунок... Done**

**Підрахунок символів... Done**

**Написання кількості символів... Done**

**TC-1 passed**

**Підрахунок... Done**

**Підрахунок символів... Done**

**Написання кількості символів... Done**

**TC-2 passed**

**Підрахунок... Done**

**Підрахунок символів... Done**

**Написання кількості символів... Done**

**TC-3 passed**

**Висновок:** дана лабораторна робота була націлена на набуття навичок

у реалізації технології модульного програмування та оброблення даних складових типыв з файловим введенням/виведенням під час розроблення статичних бібліотек, заголовкових файлів та програмних засобів у кросплатформовому середовищі Code::Blocks.

До вихідного коду статичної бібліотеки, створеної під час виконання лабораторної роботи №8 та розширеної під час виконання лабораторної роботи №9, було додано реалізації функцій для розв’язування задач 10.1-10.3 та створено допоміжні структури для більш зручної розробки. Після компіляції проекту бібліотеки в теці \obj оновився .а файл, тобто файл статичної бібліотеки. Наступним кроком було додавання до заголовкового файлу бібліотеки прототипів функцій та нових структур, які повертаються функціями.

Під час розв'язування задач було створено декілька допоміжних функцій, які по суті є частинами розв'язків, та загальні функції, які роблять попередні налагодження та викликають інші функції. Таким чином, до заголовкового файлу було додано прототипи тільки загальних функцій-розв’язків, з чого слідує, що допоміжні функції не матимуть змоги бути викликаними з будь-якого місця окрім самих загальних функцій. Це було зроблено тільки для того, щоб через цей заголовковий файл можна було користуватись тільки функціями, потрібними для розвя’зування відповідних задач.

Усі задачі вимагали реалізації роботи з вхідними та вихідними файлами, що було зроблено за допомогою потоків ifstream та ofstream зі стандартної бібліотеки <fstream>. Саме робота з файлами була реалізована у загальних функціях статичної бібліотеки, про які йшлося вище. Кожне відкриття файлів та назначеня їх відповідним потокам супроводжувалося перевіркою на те, чи успішно відкрито файл, та у випадку функція повідомляла про помилку та повертала керування у місце, з якого була викликана. В іншому випадку виконувалися потрібні операції з файлом (зчитування/запис) та файл закривався. Закриття файлу забезпечує надійність збереження змісту файлу та вважається гарними тоном. Також загальним функціям, які призначені для безпосереднього використання у подальших ПЗ, були забезпечені інтерфейсом у вигляді параметрів типу string, які призначені для передачі у функцію назв вхідного та/або вихідного файлів.

Перша задача вимагала запису до вихідного файлу анотацій розробника, речення з вхідного файлу з певними літерами заміненими на “?” та запису відповідного вірша в залежності від кількості літер (без знаків пунктуації, цифр та ін.) у вхідному файлі. <fstream> дозволяє одночасне відкриття декількох файлових потоків, тож спочатку для запису відкривається вихідний файл (створюється потік типу ofstream). За вимогами, він має бути перезаписаним або створеним, якщо він відсутній – для цього використовується флаг ios::out (аналог “w”). Якщо відкриття відбулося успішно, в нього записується анотація, після чого у паралельному файловому потоці для читання відкриважться вхідний файл (створюється потік типу ofstream) – це відбувається за допомогою флагу ios::in (анагог “r”). Після перевікрки на відкритя з файлу построково зчитується інформація, яка зберігається у змінну типу string. Важливо відмітити, що вхідні файли створюються користувачем, тож для коректної роботи застосунку вони мають бути збережені у кодуванні ANSI. Після зчитуання файл закривається, і зміст, збережений заздалегідь, обробляється шляхом заміни символів, зазначених у завданні, на “?”. Це було реалізовано за допомогою типу set<тип> вбудованої бібліотеки <set>. Тип даних set реалізує роботу з множинами (непроіндексований набір унікальних значень), що допомагає більш зручно перевіряти, чи є символ в переліку. У циклі посимвольно перевіряються символи рядка, та якщо символ присутній у множині, від перезаписується. Після виведення зміненого рядка у вихідний файл повторно відбувається процедура зі зчитуванням вхідного файлу, далі додаткова функція приймає рядок в якості аргументу та підраховує кількість літер у рядку – це значення повертається в складену функцію, береться за модулем 2 та у файл виводиться один з двох віршів. Після цих дій файл закривається.

Друга задача полягала в тому, щоб у вихідний файл дописати непарні символи з вхідного речення та дату й час дозапису. Для того, щоб дописати інформацію в кінець файлк, його потрібно відкрити для запису з флагом ios::app. Далі вхідне речення зчитується та кожний непарний символ записується у файл. Непарністю в даному випадку вважається позиція символа у тексті з точки зору корисувача (1 символ, 3 символ...), тож на програмному рівні непарні символи рядка – ті, що маєть парний індекс. Дата та час дозапису були реалізовані за допомогою бібліотеки <ctime>, що дає змогу створити змінну типу time\_t, зберігти в неї поточно дату та час у комп'ютерному вигляді (кількість секунд, що пройшла з початку так званої «цифрової ери») за допомогою функції time() та перевести значення цієї змінної у зрозумілий людині вигляд за допомогою функції ctime().

У третій задачі потрібно було у вихідний файл дописати результат функції s\_calculation() з аргументами x, y, z, та записати число b у двійковому вигляді. Загальна функція для розв’язку третьої задачі має відповідні параметри, що дозволяють через неї передавати аргументи функції s\_calculation(). Число b було переведено у двійковий вигляд за допомогою типу даних bitset, що міститься в однойменній бібліотеці. Робота з файлом аналогічна такій, що була у задачі 2.

Далі було створено проект консольного додатка С++ під назвою TestDriver, метою створення якого є реалізація модульного тестування. Для кожної функції було розраховано еталонні вхідні дані та відповідні еталонні результати, які порівнювалися з результатами роботи функцій. Через те, що 2 та 3 задачі випагають дозапису у файл, було прийняте рішення розробити 9 файлів з еталонними результатами, відповідно кожний вихідний файл (3 тести – 3 файли) порівнювався з відповідним файлом очікуваного результату для поточного етапу роботи застосунку (умовно поділемо роботу застосунку на три етапи: перший – перезапис у файл розв'язку першої задачі, другий – додання результату роботи другої функцій, третій – додання результату роботи реалізації третьої задачі). Застосунок було спроектовано для автоматиного тестування, тож на посатку роботи тестового драйвера створюються вхідні файли, в які автоматично записуються еталонні вхідні дані.

Для першої функції перевірялася повна відповідність змісту у файі, що є результатом роботи програми, та очікуваним результатом.

Для другої функції перевірялося те саме, що й у першій функціїї, за виключенням того, що частина рядка з датою та часом перевірялися на відповіднсть формату часу (Www Mmm DD HH:MM:SS YYYY). Через те, що довжина рядка з датою та часом є костантною, вона була зазначена за допомогою макросу #define.

Для третьої функціїї увесь зміст, який знаходиться до дописаних даних, перевіряється так само, як у для другої функції, а дописані дані – так само, як для першої. Проблемою є те, що для різних вхідних даних для першої та другої функцій результат займає різну кількість пам’яті, тож для перевірки формату дати та часу довелось реалізувати пошук цього підрядка у файлі.

Через те, що у вихідному коді міститься велика кількість кириличних символів, усі файли проекту було переформатовано та змінено для роботи у кодуванні UTF-8 для коректного відображення на платформі GitHub. Проте для того, щоб полегшити роботу з обробкою та выдображенням тексту в консолы та файлах, у додаткових налаштуваннях компілятору для певного проекту було введено команду для того, щоб при білдингу коду усі потрібні файли з UTF-8 конвертувалися у CP1251.

Усі налаштування, перераховані у висновку, були застосовані до проектів, а не до середовища, для того, щоб проект успішно виконувався в IDE Code::Blocks на різних пристроях, незалежно від налаштувань певного середовища.