**ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ  
«ОПТИКО-МЕХАНІЧНИЙ ФАХОВИЙ КОЛЕДЖ КИЇВСЬКОГО  
НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ІМЕНІ ТАРАСА ШЕВЧЕНКА»**Циклова комісія програмування та інформаційних технологій

**ЗВІТ**

**З НАВЧАЛЬНОЇ ПРАКТИКИ**

спеціальність *121 Інженерія програмного забезпечення*освітньо-кваліфікаційний рівень *«Фаховий молодший бакалавр»*

|  |  |
| --- | --- |
| ОЦІНКА  (оцінка цифрою та прописом)  Керівники практики від коледжу:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис) | Виконав:  студент 2-го курсу групи ІПЗ-21  *Капічніков Єгор Сергійович*  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис виконавця)  « \_» 20 року |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис) |  |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис) |  |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис) |  |

« \_»\_ 2024 року

КИЇВ — 2024

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| на період | *навчальної* | | практики | |
|  | (навчальної, технологічної, виробничої) | |  | |
| *Капічнікова Єгора Сергійовича* | | | | |
| (П.І.Б студента.) | | | | |
| Спеціальність | *121 Інженерія програмного забезпечення* | Група | | *ІПЗ-23* |

|  | **Питання для вивчення** | **Відмітки про виконання** |
| --- | --- | --- |
| 4 | **Розділ " Практична робота з С++"** |  |
| 4.1 | **Тема 1. Введення та виведення у базових консольних програмах** |  |
| 4.2 | **Тема 2. Структури даних для управління станом програми** |  |
| 4.3 | **Тема 3. Робота з файлами: запис та читання** |  |
| 4.4 | **Тема 4. Формати файлів та їх обробка** |  |
| 4.5 | **Тема 5. Системи контролю версій (VCS)** |  |
| 4.6 | **Тема 6. Мова Markdown для опису проекті** |  |
| 4.7 | **Розроблення ігрової програми** |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Голова циклової комісії |  |  | Савєльєва І.В. |  |  |
|  | (підпис) |  | (ініціали, прізвище) |  | (дата) |
| Керівники практики від коледжу |  |  |  |  |  |
|  |  |  | Нікітенко А.С. |  | 17.05.2024р. |
|  | (підпис) |  | (ініціали, прізвище) |  | (дата) |
|  |  |  | Лумпова Т.І. |  | 17.05.2024р. |
|  | (підпис) |  | (ініціали, прізвище) |  | (дата) |
|  |  |  | Нікітенко А.С. |  | 17.05.2024р. |
|  | (підпис) |  | (ініціали, прізвище) |  | (дата) |
|  |  |  | Лумпова Т.І. |  | 17.05.2024р. |
|  | (підпис) |  | (ініціали, прізвище) |  | (дата) |

**ВСТУП**

**Мова програмування С++** розроблена Б’ярном Страуструпом в підрозділі Bell Laboratories компанії AT&T в 1979 році. Метою створення С++ було доповнення мови програмування С можливостями, зручними для більш глобальної розробки ПО, зі збереженням гнучкості, швидкості і портативності мови C. Разом з тим творці C++ прагнули зберегти сумісність цієї мови програмування з мовою C.

Зараз мова C++ широко використовується для розробки програмного забезпечення, будучи однією з найпопулярніших мов програмування. З її допомогою створюють операційні системи, різноманітні прикладні програми, драйвери пристроїв, ігри тощо.

Серед найбільш популярного ПО, написаного на мові C++ (або з її використанням), є популярна СУБД MySQL, інтернет-браузер Mozilla Firefox, більша частина програмного забезпечення від Microsoft: операційні системи сімейства Windows, IDE Visual Studio, Internet Explorer, Microsoft Office і, звичайно ж, Adobe Photoshop, Adobe Illustrator і Adobe Premiere Pro повністю написані на C++. Також С++ лежить в основі ігрового движка Unity.

**Теоретична частина**

**4.1 Тема 1. Введення та виведення у базових консольних програмах**

У мові C++ дії, що пов’язані з операціями введення і виве­дення, виконуються за допомогою функцій бібліотек. Функції ведення і виведення бібліотек мови дозволяють читати дані з файлів та пристроїв і писати дані у файли і на пристрої.

Бібліотека мови C++ підтримує три рівня введення-виведення даних:

* введення-виведення потоку;
* введення-виведення нижнього рівня;
* введення-виведення для консолі і порту.

При введенні-виведенні потоку всі дані розглядаються як потік окремих байтів. Для користувача потік — це файл на дис­ку або фізичний пристрій, наприклад, дисплей чи клавіатура, або пристрій для друку, з якого чи на який направляється потік даних. Операції введення-виведення для потоку дозволяють об­робляти дані різних розмірів і форматів від одиночного символу до великих структур даних. Програміст може використовувати функції бібліотеки, розробляти власні і включати їх у бібліоте­ку. Для доступу до бібліотеки цих класів треба включити в програму відповідні заголовні файли.

За замовчуванням стандартні введення і виведення повідомлень про помилки відносяться до консолі користувача (клавіатури та екрана). Це означає, що завжди, коли програма очікує введення зі стандартного потоку, дані повинні надходити з клавіатури, а якщо програма виводить дані — то на екран.

У мові C++ існує декілька бібліотек, які містять засоби введення-виведення, наприклад:**stdio.h, iostream.h.** Найчастіше застосовують потокове введення-виведення даних, операції яко­го включені до складу класів **istream** або **iostream**. Доступ до бібліотеки цих класів здійснюється за допомогою використання у програмі директиви компілятора **#include <iostream.h>**.

**Для потокового введення даних** вказується операція «**>>**» («читати з»). Це перевантажена операція, визначена для всіх простих типів і покажчика на **char**. Стандартним потоком введення є **cin**.

Формат запису операції введення має вигляд:

**сin [>> values];**

де ***values*** — змінна.

Так, для введення значень змінних**х і у** можна записати:

**сin >> х >> у;**

Кожна операція «>>» передбачає введення одного значення. При такому введенні даних необхідно дотримуватись конкретних вимог:

* для послідовного введення декількох чисел їх слід розділяти символом пропуску (« ») або **Enter** (дані типу **char** роздiляти пропуском необов’язково);
* якщо послідовно вводиться символ і число (або навпаки), пропуск треба записувати тільки в тому випадку, коли символ (типу **char**) є цифрою;
* потік введення ігнорує пропуски;
* для введення великої кількості даних одним оператором їх можна розташовувати в декількох рядках (використовуючи Enter);
* операція введення з потоку припиняє свою роботу тоді, коли всі включені до нього змінні одержують значення. Наприклад, для операції введення**х і у**, що вказана вище, можна ввести значення **х та у** таким чином:

**2.345   789**  
або   **2.345**  
**789.**

Оскільки в цьому прикладі пропуск є роздільником між значеннями, що вводяться, то при введенні рядків, котрі мiстять пропуски у своєму складі, цей оператор не використовується. У такому випадку треба застосовувати функції **getline(), get()** тощо (див.*розділ 7*). У мові C++ бажано здійснювати потокові введення-виведення даних.

**Для потокового виведення даних** необхідна операція **«<<»** («записати в»), що використовується разом з ім’ям вихідного потоку **cout**. Наприклад, вираз

**cout << х;**

означає виведення значення змінної **х**(або запис у потік). Ця операція вибирає необхідну функцію перетворення даних у потік байтів.

Формат запису операції виведення представляється як:

**cout << data [<< data1];**,

де **data, data1** — це змінні, константи, вирази тощо.

Потокова операція виведення може мати вигляд:  
**cout << “у =” << x + а – sin(x) << “\n“;.**

Застосовуючи логічні операції, вирази треба брати в дужки:  
**cout << “р =” << (а && b || с) << “\n“;.**

Символ переведення на наступний рядок записується як рядкова константа, тобто **“\n”**, інакше він розглядається не як символ керуючої послідовності, а як число 10 (код символу). Таких помилок можна уникнути шляхом присвоювання значення ке­руючих символів змінним, тобто:

**#define << sp " "**

**#define << ht "\t"**

**#define << hl "\n".**

Тепер операцію виведення можна здійснити так:  
**cout <<** **“у =” << x** **+ а – sin(x)<<** **hl;** .

Слід пам’ятати, що ***при виведенні даних з використанням «cout <<» не виконується автоматичний перехід на наступ­ний рядок, для реалізації такого переходу застосовується так переведення рядка*** ***“\n” або операція*** ***endl***

**4.2 Структури даних для управління станом програми**

Масив — СД, згруповані за місцем розташування у пам'яті величини, що мають одну назву (ідентифікатор) і різні порядкові номери (індекси). Це поняття програмування відповідає математичним поняттям послідовності й таблиці (матриці).

Елемент масиву — одна з величин, що утворюють масив. Це поняття програмування відповідає математичному поняттю елемента послідовності чи матриці.

Індекс масиву — величина перелічуваного (зазвичай цілого) типу, яка (сукупність яких) вказує на конкретний елемент масиву. Це поняття програмування відповідає математичному поняттю номера елемента послідовності чи номеру рядка/стовпчика таблиці (матриці). У мові C++ найменше значення індексу — 0.

Масив має такі властивості:

* Всі елементи мають однаковий тип.
* Масив містить фіксовану (незмінну протягом роботи програми) кількість елементів.
* Елементи масиву розташовуються в пам’яті один за одним.
* Всі елементи зберігаються під одним іменем.

У мові C++ властивості елементів масиву не можна змінити протягом виконання програми. В деяких інших мовах це можливо.

Контейнер list (список) у мові С++ задає двонаправлений список. У ці списки можна швидко вставляти, а також видаляти елементи з них. Доступ до елементів списку (як і всіх наступних структур) здійснюють за допомогою вказівників (ітераторів).

#include <list> — замовлення необхідної бібліотеки;

list <тип\_даних> назва\_списку; — опис списку;

list <тип\_даних> :: iterator назва\_ітератора; — опис ітератора (вказівника);

l.begin() — вказівник на початок списку l;

l.end() — вказівник на кінець списку l;

l.rbegin() — реверсивний вказівник на кінець списку l;

l.rend() — реверсивний вказівник на початок списку l;

l.empty() — істинність висловлювання: «Список l порожній»;

l.push\_back(e) — додати елемент e в кінець списку l;

l.pop\_back() — видалити останній елемент списку l;

l.front() — перший елемент списку l;

l.back() — останній елемент списку l;

l.insert(i,e) — вставлення елемента e в позицію i списку l;

l.erase(і, у) — видалення зі списку l послідовності елементів від позиції і до позиції у.

## **4.3 Тема 3. Робота з файлами: запис та читання**

**Класи файлового вводу/виводу**

Є три основні класи файлового вводу/виводу в мові C++:

**ifstream** (є дочірнім класу istream);

**ofstream** (є дочірнім класу ostream);

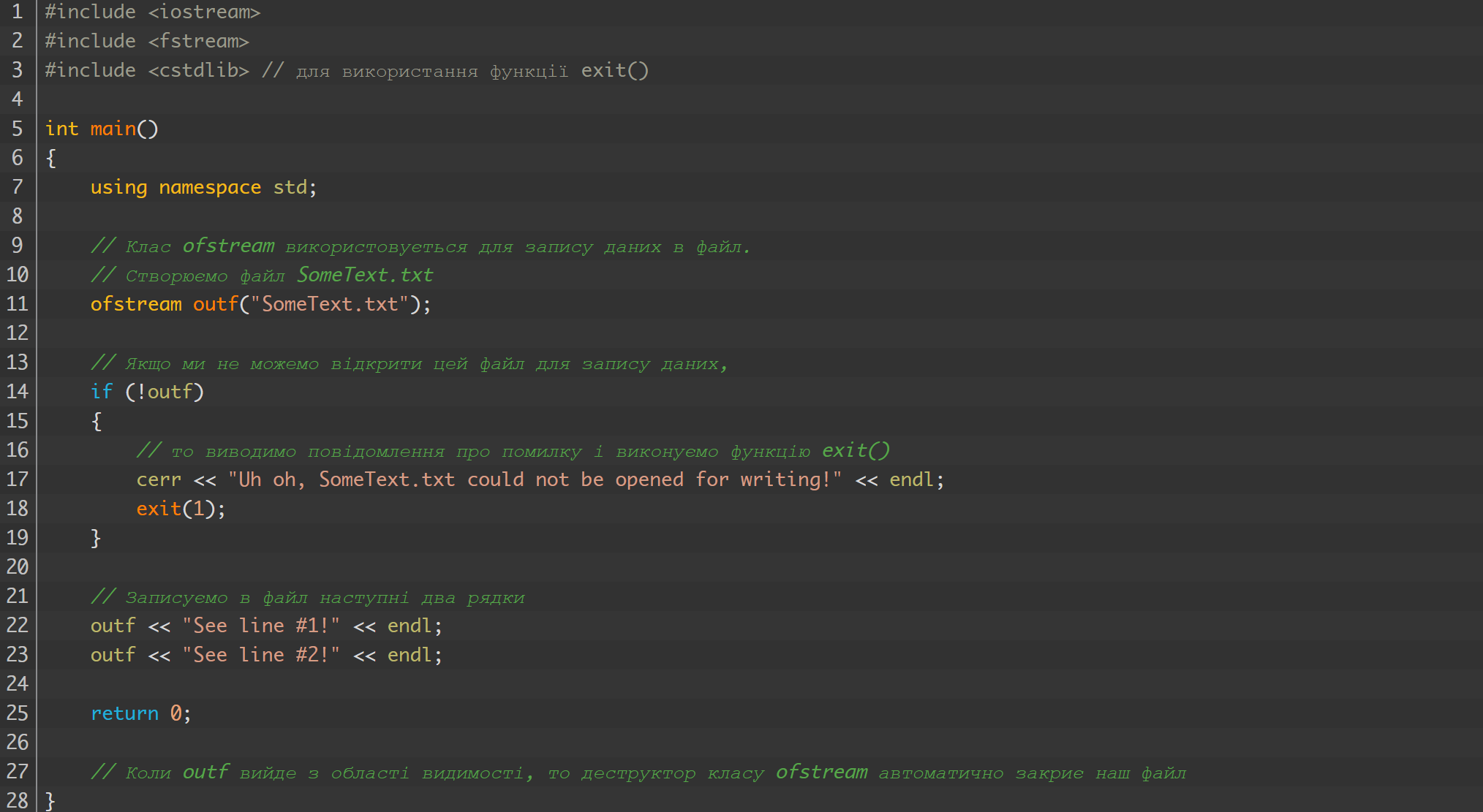
**fstream** (є дочірнім класу iostream).

За допомогою цих класів можна виконувати однонаправлений файловий ввід, однонаправлений файловий вивід і двонаправлений файловий ввід/вивід. Для їх використання потрібно всього лише підключити заголовок fstream.

На відміну від потоків cout, cin, cerr і clog, які відразу ж можна використовувати, файлові потоки повинні бути явно встановлені програмістом. Тобто, щоб відкрити файл для читання і/або запису, потрібно створити об’єкт відповідного класу файлового вводу/виводу, вказавши ім’я файлу в якості параметра. Потім, за допомогою оператора вставки (<<) або оператора вилучення (>>), можна записувати дані в файл або зчитувати вміст файлу. Після виконання даних дій потрібно закрити файл — явно викликати метод close() або просто дозволити файловій змінній вводу/виводу вийти з області видимості (деструктор файлового класу вводу/виводу закриє цей файл автоматично замість нас).

## **Файловий вивід**

Для запису в файл використовується клас ofstream. Наприклад:



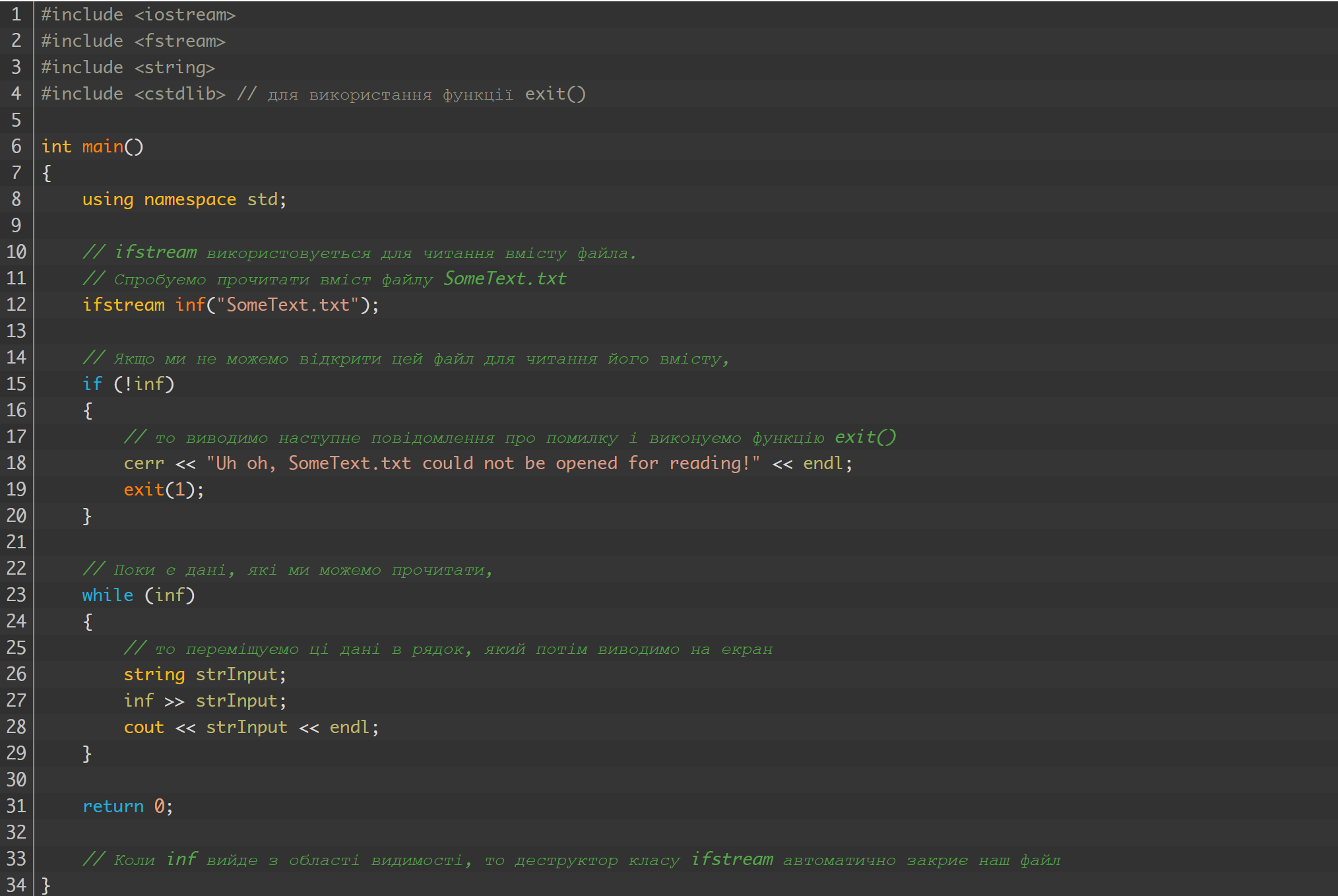
Якщо зайти в каталог проекту (ПКМ по вкладці з назвою .cpp-файлу в Visual Studio > “Открыть содержащую папку”), то можна побачити файл з ім’ям SomeText.txt, в якому знаходяться наступні рядки:

**See line #1!  
See line #2!**

Також можемо використати **метод put()** для запису одного символу в файл.

## **Файловий ввід**

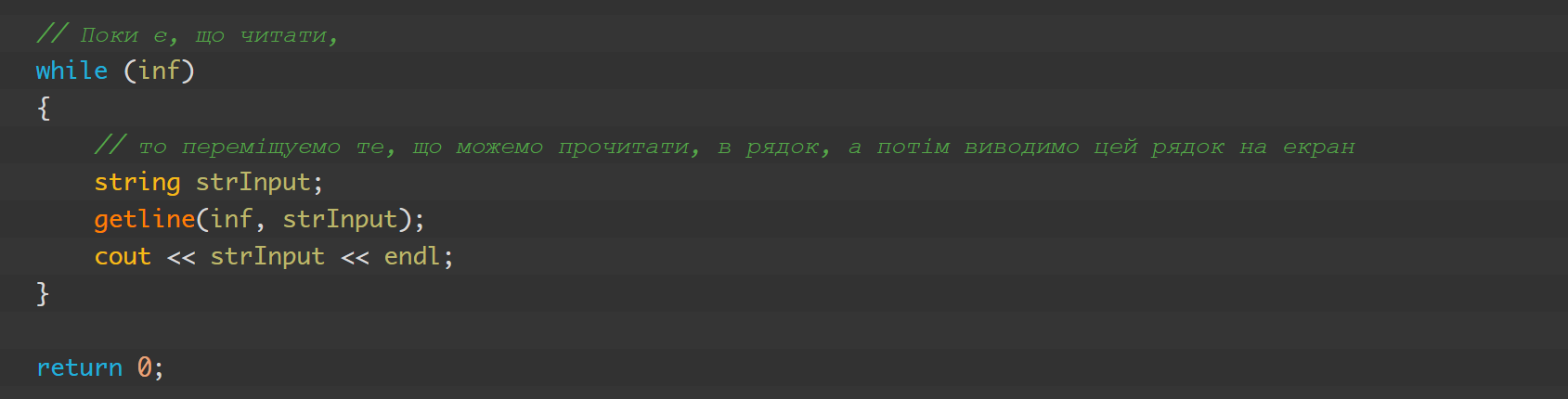
Тепер ми спробуємо прочитати вміст файлу, який створили в попередньому прикладі. Зверніть увагу, ifstream поверне 0, якщо ми досягли кінця файлу (це зручно для визначення «довжини» вмісту файлу). Наприклад:



Результат виконання програми:

**See  
line  
#1!  
See  
line  
#2!**

Оператор вилучення працює з «відформатованими даними», тобто він ігнорує всі пробіли, символи табуляції і символ нового рядка. Щоб прочитати весь вміст як є, без його ділення на частини (як у вищенаведеному прикладі), нам потрібно використати метод **getline():**



Результат виконання програми:

**See line #1!  
See line #2!**

## **Буферизований вивід**

Вивід в мові C++ може бути буферизований. Це означає, що все, що виводиться в файловий потік, не може відразу ж бути записано на диск (в конкретний файл). Це зроблено, в першу чергу, з міркувань продуктивності. Коли дані буфера записуються на диск, то це називається очищенням буфера. Одним із способів очищення буфера є закриття файлу. У такому випадку весь вміст буфера буде переміщено на диск, а потім файл буде закрито.

Буферизація виводу зазвичай не є проблемою, але при певних обставинах вона може викликати проблеми у необережних новачків. Наприклад, коли в буфері зберігаються дані, а програма передчасно завершує своє виконання (або в результаті збою, або шляхом виклику функції exit()). У таких випадках деструктори класів файлового вводу/виводу не виконуються, файли ніколи не закриваються, буфери не очищаються і наші дані губляться назавжди. Ось чому гарною ідеєю є явне закриття всіх відкритих файлів перед викликом функції exit().

Також буфер можна очистити вручну, використовуючи метод ostream::flush() або відправивши std::flush в вихідний потік. Будь-який з цих способів може бути корисний для забезпечення негайного запису вмісту буфера на диск в разі збою програми.

**Режими відкриття файлів**

Що відбудеться, якщо ми спробуємо записати дані в уже існуючий файл? Повторний запуск вищенаведеної програми (найперша) показує, що вихідний файл повністю перезаписується при повторному запуску програми. А що, якщо нам потрібно додати дані в кінець файлу? Виявляється, [конструктори](https://acode.com.ua/urok-124-konstruktory/) файлового потоку приймають необов’язковий другий параметр, який дозволяє вказати програмісту спосіб відкриття файлу. В якості цього параметра можна передавати наступні флаги (які знаходяться в класі ios):

**app**— відкриває файл в режимі додавання;

**ate** — переходить в кінець файлу перед читанням/записом;

**binary** — відкриває файл в бінарному режимі (замість текстового режиму);

**in** — відкриває файл в режимі читання (за замовчуванням для ifstream);

**out**— відкриває файл в режимі запису (за замовчуванням для ofstream);

**trunc** — видаляє файл, якщо він вже існує.

Можна вказати відразу кілька флагів шляхом використання побітового АБО ().

**ifstream** за замовчуванням працює в режимі ios::in;

**ofstream** за замовчуванням працює в режимі ios::out;

**fstream** за замовчуванням працює в режимі ios::in АБО ios::out, що означає, що ви можете виконувати як читання вмісту файлу, так і запис даних в файл.

## **Явне відкриття файлів за допомогою функції open()**

Точно так же, як ми явно закриваємо файл за допомогою методу close(), ми можемо явно відкривати файл за допомогою **функції open()**. Функція open() працює аналогічно конструкторам класу файлового вводу/виводу: приймає ім’я файлу і режим (необов’язково), в якому потрібно відкрити файл, в якості параметрів.

**4.4 Формати файлів та їх обробка**

Формати файлів та їх обробка є ключовими аспектами роботи з інформацією у різних сферах діяльності. Ось деякі з найпоширеніших форматів файлів та способи їх обробки:

Текстові файли

1. TXT: простий текстовий файл без форматування. Використовується для зберігання необробленого тексту.

- Обробка: редагування у будь-якому текстовому редакторі (Notepad, Sublime Text), програмна обробка мовами Python, Java, C++ тощо.

2. JSON (JavaScript Object Notation): легкий формат обміну даними, який легко читається і записується людиною.

- Обробка: програмна обробка мовами JavaScript, Python (json бібліотека), Java тощо.

3. CSV (Comma-Separated Values): формат для зберігання табличних даних, де значення розділені комами.

- Обробка: відкриття в електронних таблицях (Excel, Google Sheets), програмна обробка мовами Python (pandas), R тощо.

4. XML (eXtensible Markup Language): формат для зберігання та передачі структурованих даних.

- Обробка: використання парсерів (lxml в Python, DOM у Java), редагування спеціалізованими редакторами (XMLSpy).

**JSON**

JSON - текстовий формат обміну даними, що базується на JavaScript. Як і багато інших текстових форматів, JSON легко читається людьми.

Незважаючи на походження від JavaScript (точніше, від підмножини мови стандарту ECMA-262 1999), формат вважається незалежним від мови і може використовуватися практично з будь-якою мовою програмування. Для багатьох мов існує готовий код для створення та обробки даних у форматі JSON.

За рахунок своєї лаконічності, порівняно з XML, формат JSON може бути більш підходящим для серіалізації складних структур. Застосовується у веб-додатках як для обміну даними між браузером та сервером (AJAX), так і між серверами (програмні зв'язки HTTP).

Оскільки формат JSON є підмножиною синтаксису мови JavaScript, він може бути швидко десеріалізований вбудованою функцією JSON.parse().

**CSV**

CSV - текстовий формат, призначений для представлення табличних даних. Рядок таблиці відповідає рядку тексту, який містить одне або кілька полів, розділених комами.

Кожен рядок файлу – це один рядок таблиці.

Розділювачем значень колонок є символ комою (,). Проте практично часто використовуються інші роздільники, тобто формат плутають з DSV і TSV.

Значення, що містять зарезервовані символи (подвійна лапка, кома, крапка з комою, новий рядок) обрамляються подвійними лапками ("). Якщо у значенні зустрічаються лапки - вони подаються у файлі у вигляді двох лапок поспіль.

**Читання/запис JSON файлів**

**Читання**

Читання даних з формату JSON і перетворення їх в об’єкти JavaScript здійснюється за допомогою методу JSON.parse(). Цей метод приймає як аргумент рядок JSON і повертає відповідний JavaScript об’єкт.

Приклад:

*const jsonString = '{"name":"John","age":30,"email":"john@example.com"}';*

*const user = JSON.parse(jsonString);*

*console.log(user.name); // Виведе: "John"*

*console.log(user.age); // Виведе: 30*

*console.log(user.email); // Виведе: "john@example.com"*

**Запис**

Запис даних у формат JSON виконується за допомогою методу JSON.stringify(). Цей метод приймає об’єкт JavaScript і перетворює його на рядок JSON.

Приклад:

*const user = {*

*name: "John",*

*age: 30,*

*email: "john@example.com"*

*};*

*const jsonString = JSON.stringify(user);*

*console.log(jsonString);*

*// Виведе: '{"name":"John","age":30,"email":"john@example.com"}'*

**Читання/запис CSV файлів**

Для роботи з CSV файлами можна використовувати бібліотеку rapidcsv.

Для читання таких файлів підходить така команда:

*rapidcsv::Document doc("example.csv");*

**Читання/запис XML файлів**

Для роботи з CSV файлами можна використовувати бібліотеку pugixml.

Для читання таких файлів підходить така команда:

pugi::xml\_parse\_result result = doc.load\_file("example.xml");

## **4.5 Системи контролю версій (VCS)**

**VCS** – місце зберігання коду. Вона уточнена саме на розробку продуктів. Тобто зберігання коду, синхронізацію роботи кількох людина, створення релізів (бранчів).

Як працює **VCS:**

**1. Створити репозиторій**

Вихідно потрібно створити місце, де лежатиме код. Воно називається репозиторій. Створюється один раз адміністратором.

**2. Встановити проект з репозиторія**

Тепер потрібно отримати проект із репозиторію. Забрати код таким чином потрібно рівно один раз

**Що з допомогою треба робити:**

**1. Оновити проект, забрати останню версію з репозиторію**

**2. Вносити зміну у репозиторій**

Основні типи VCS

1. Локальні VCS:

- Зберігають версії файлів на локальному комп'ютері.

- Простий, але має обмеження у можливостях командної роботи.

2. Централізовані VCS (CVCS):

- Використовують центральний сервер для зберігання всіх версій файлів.

- Забезпечують спільну роботу над проектами, але є залежність від центрального сервера.

3. Розподілені VCS (DVCS):

- Кожен розробник має повну копію історії змін проекту на своєму локальному комп'ютері.

- Більш гнучкі і стійкі до збоїв центрального сервера.

**4.6 Мова Markdown для опису проекті**

Markdown – мова розмітки, яку використовують для створення нотаток, документів, вебсайтів, технічної документації тощо. Цікаво те, що файли Markdown можна перетворювати на презентації, які зручно переглядати в браузері.

За допомогою **Markdown** можна робити таку розмітку:

1. Заголовки 6 рівнів

# Заголовок 1

## Заголовок 2

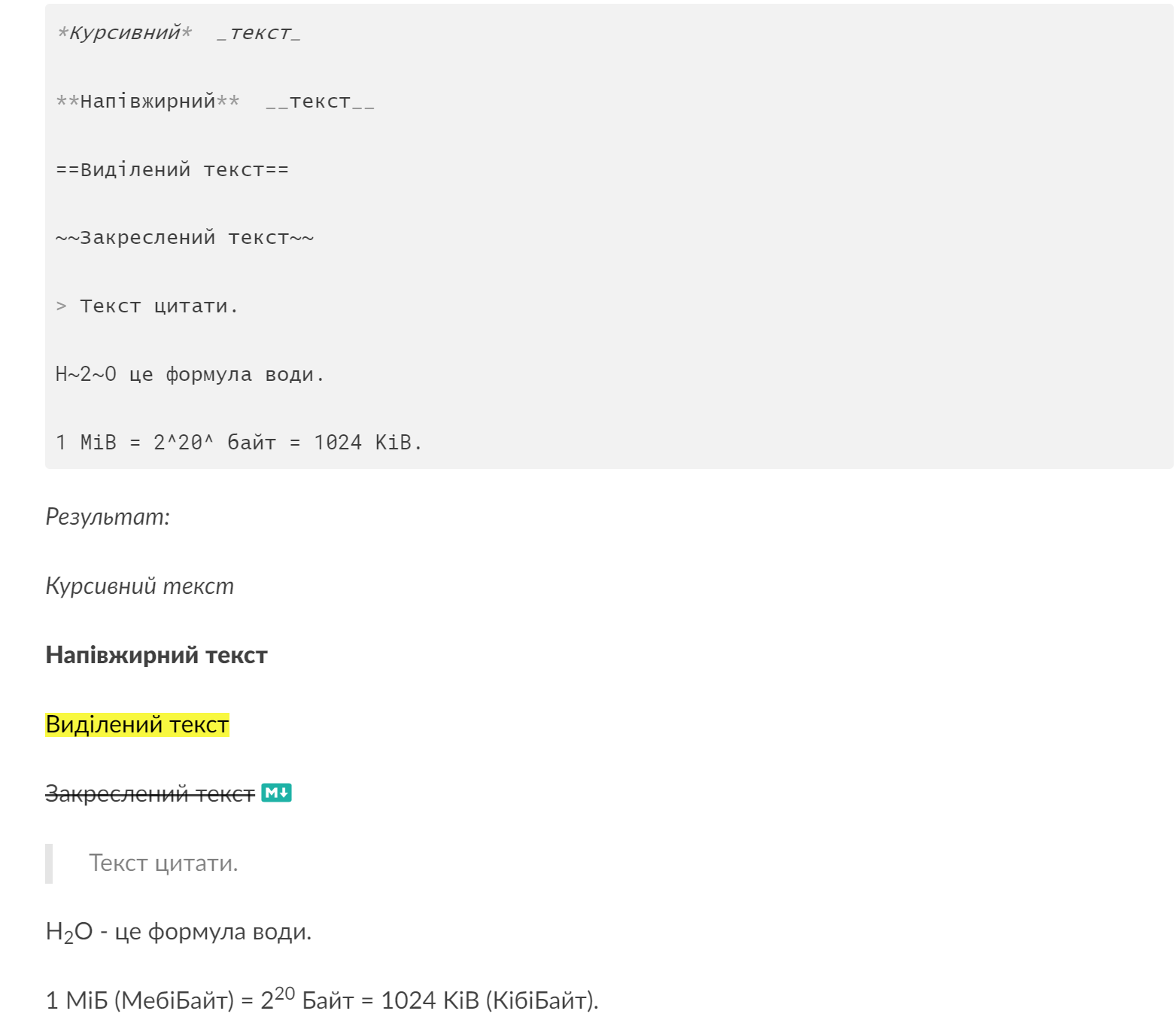
### Заголовок 3

#### Заголовок 4

##### Заголовок 5

###### Заголовок 6

2. Форматування тексту



3. Створювати списки

- елемент 1

- елемент 2

- елемент 3

Також у списках можуть бути вкладені елементи

- елемент 1

- елемент 2

\* вкладений елемент 2.1

\* вкладений елемент 2.2

- елемент 3

4. Посилання

Це [посилання](https://example.com).

Це [приклад][1] [кількох][2] [посилань][id] з розміткою як у виносок в тексті. Можна використовувати [короткий запис][] без `id`.

[1]: https://example.com "Назва посилання 1"

[2]: https://example.com/some

[id]: https://example.com/links (Назва посилання id)

[короткий запис]: https://example.com/short

5. Ідентифікатори заголовків

### Горизонтальна лінія

[Горизонтальна лінія](#горизонтальна-лінія)

Результат:

Горизонтальна лінія

6. Зображення

![alt](https://bit.ly/33B4VxM "shih-tzu")

Результат:



Також можна змінювати розмір зображення

![alt](https://bit.ly/33B4VxM "shih-tzu" = 80х120)

Результат:



## **4.7 Розроблення ігрової програми**

**4.7.1 Код програми**

// Kapychnykov Yegor, ІПЗ-23

#include <iostream>

using namespace std;

// Ініціалізація поля

char board[3][3] = { {'1', '2', '3'}, {'4', '5', '6'}, {'7', '8', '9'} };

char current\_marker;

int current\_player;

void drawBoard() {

cout << "-------------\n";

for (int i = 0; i < 3; i++) {

cout << "|";

for (int j = 0; j < 3; j++) {

cout << " " << board[i][j] << " |";

}

cout << "\n-------------\n";

}

}

bool placeMarker(int slot) {

int row = (slot - 1) / 3;

int col = (slot - 1) % 3;

if (board[row][col] != 'X' && board[row][col] != 'O') {

board[row][col] = current\_marker;

return true;

}

return false;

}

int winner() {

// Перевірка строк

for (int i = 0; i < 3; i++) {

if (board[i][0] == board[i][1] && board[i][1] == board[i][2])

return current\_player;

}

// Перевірка стовбців

for (int i = 0; i < 3; i++) {

if (board[0][i] == board[1][i] && board[1][i] == board[2][i])

return current\_player;

}

// Перевірка діагоналей

if (board[0][0] == board[1][1] && board[1][1] == board[2][2])

return current\_player;

if (board[0][2] == board[1][1] && board[1][1] == board[2][0])

return current\_player;

return 0;

}

void swapPlayerAndMarker() {

if (current\_marker == 'X')

current\_marker = 'O';

else

current\_marker = 'X';

if (current\_player == 1)

current\_player = 2;

else

current\_player = 1;

}

void game() {

cout << "Player 1, choose your option (X or O): ";

char marker\_p1;

cin >> marker\_p1;

current\_player = 1;

current\_marker = marker\_p1;

drawBoard();

int player\_won;

for (int i = 0; i < 9; i++) {

cout << "Player " << current\_player << ", choose number of cell (1-9): ";

int slot;

cin >> slot;

if (slot < 1 || slot > 9) {

cout << "Incorrect number of cell. You can choose numbers only from 1 to 9.\n";

i--;

continue;

}

if (!placeMarker(slot)) {

cout << "Cell;s already chosen. Try another one.\n";

i--;

continue;

}

drawBoard();

player\_won = winner();

if (player\_won == 1) {

cout << "Player 1 won!\n";

break;

}

if (player\_won == 2) {

cout << "Player 2 won!\n";

break;

}

swapPlayerAndMarker();

}

if (player\_won == 0)

cout << "It's a draw\n";

}

int main() {

game();

return 0;

}

**4.7.2 Скріншоти виконання программи**

