**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:  студентка 2-го курсу групи ІПЗ-23  *Стешенко Анастасія Сергіївна*  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис виконавця)  « \_» 20 року |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис) |  |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис) |  |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис) |  |

« \_»\_ 2024 року

КИЇВ — 2024

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на період *Навчальної* практики

Стешенко Анастасії Сергіївни

Спеціальність *121 Інженерія програмного забезпечення* Група *ІПЗ-23*

|  |  |  |
| --- | --- | --- |
| **4** | Розділ "Практична робота з С++" |  |
| 4.1 | Введення та виведення у базових консольних програмах |  |
| 4.2 | Робота з файлами: запис та читання |  |
| 4.3 | Формати файлів та їх обробка |  |
| 4.4 | Системи контролю версій (VCS) |  |
| 4.5 | Мова Markdown для опису проектів |  |
| 4.6 | Створення ігрової програми |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Голова циклової комісії |  |  |  |
|  | (підпис) | (ініціали, прізвище) | (дата) |
| Керівники практики від коледжу |  |  |  |
|  | (підпис) | (ініціали, прізвище) | (дата) |
|  |  |  |  |
|  | (підпис) | (ініціали, прізвище) | (дата) |
|  |  |  |  |
|  | (підпис) | (ініціали, прізвище) | (дата) |

**Зміст**

**ОСНОВНА ЧАСТИНА**

**Теоретичні відомості**

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

Введення та виведення у базових консольних програмах є одним із фундаментальних аспектів програмування. Ці операції дозволяють користувачам спілкуватися з програмою та отримувати результати її роботи через консольний інтерфейс.

***1. Введення даних:*** У мові C++ для цього використовується об'єкт std::cin, який дозволяє отримати дані з клавіатури. Наприклад:

#include <iostream>

int main() {

int number;

std::cout << "Enter a number: ";

std::cin >> number;

std::cout << "You entered: " << number << std::endl;

return 0;

}

У цьому прикладі користувач вводить число з клавіатури, а програма виводить його назад у консоль.

***2. Виведення даних:*** Для виведення даних у консоль у мові C++ використовується об'єкт std::cout. Наприклад:

#include <iostream>

int main() {

std::cout << "Hello, world!" << std::endl;

return 0;

}

У цьому прикладі рядок "Hello, world!" виводиться у консоль за допомогою об'єкта std::cout.

Ці прості операції введення та виведення даних у консоль дозволяють створювати взаємодію між користувачем та програмою, а також надають можливість відображення результатів обчислень та роботи програми.

***Тема 2. Робота з файлами: запис та читання***

JSON (JavaScript Object Notation) - це легкий формат обміну даними, який використовується для зберігання та передачі структурованих даних між різними системами. Він базується на синтаксисі JavaScript, але може бути використаний в багатьох інших мовах програмування. JSON використовується для представлення об'єктів та масивів даних у формі, зрозумілій як людям, так і програмам.

Основні складові JSON - це пари "ключ-значення", які можуть бути вкладені одна в одну для представлення складних структур даних. JSON широко використовується для обміну даними між клієнтськими та серверними додатками, а також для зберігання налаштувань, конфігурацій та інших структурованих даних.

***Робота з файлами JSON:***

*Плюси:*

1. Легкість читання та редагування: Формат JSON має просту та легку для сприйняття структуру, що робить файли JSON легкими для читання та редагування як людиною, так і машинами.
2. Підтримка мов програмування: Більшість сучасних мов програмування мають бібліотеки або пакети, які дозволяють легко читати та записувати дані у форматі JSON, що робить його досить універсальним для використання.
3. Гнучкість: JSON дозволяє представляти різноманітні дані, включаючи тексти, числа, масиви, об'єкти та вкладені структури даних.
4. Популярність: JSON є одним з найбільш поширених форматів обміну даними в сучасному програмуванні, що означає, що для нього існують багато засобів підтримки та бібліотек для різних мов програмування.

*Мінуси:*

1. Великий обсяг даних: JSON може виявитися неефективним для великих обсягів даних, оскільки він зазвичай використовує багато зайвих символів для форматування.
2. Брак сховищ даних: JSON не надає вбудованих механізмів для роботи з великими обсягами даних, такими як індексація або пошук, що може ускладнити обробку великих файлів.

XML (Extensible Markup Language) - це розширювана мова розмітки, яка використовується для представлення та обміну структурованими даними між комп'ютерними системами. XML базується на тезі, що документ має бути структурований та зрозумілий як комп'ютерам, так і людям.

В основі XML лежить синтаксис тегів, подібний до HTML, але він не має заздалегідь визначених тегів та може бути розширений для використання в будь-якому контексті. XML використовується для створення структурованих документів, таких як конфігураційні файли, налаштування програм, обміну даними між веб-серверами та клієнтами, а також для зберігання даних в базах даних. Він є одним з основних форматів для обміну даними в сучасному програмуванні та веб-розробці.

***Робота з файлами XML:***

*Плюси:*

1. Структурованість: XML має строго визначену структуру, що робить його ідеальним для представлення складних структур даних та документів з вкладеними елементами.
2. Можливість валідації: XML може бути легко валідований за допомогою схем XSD (XML Schema Definition), що дозволяє перевіряти правильність структури та типів даних у файлі.
3. Підтримка стандартів: XML підтримується багатьма стандартами та протоколами, що робить його корисним для обміну даними між різними системами.

*Мінуси:*

1. Великий обсяг даних: XML-файли можуть бути досить об'ємними через велику кількість тегів та атрибутів, що може призвести до збільшення розміру файлу та ускладнення його обробки.
2. Складність читання та редагування: Велика кількість тегів та вкладеності може зробити XML-файли складними для читання та редагування людиною, особливо у випадку великих обсягів даних.
3. Обмежена підтримка: Хоча більшість мов програмування підтримують роботу з XML, цей формат може виявитися менш зручним у порівнянні з іншими форматами, такими як JSON, зокрема у веб-розробці та обміні даними між веб-серверами та клієнтами.

***Тема 3. Формати файлів та їх обробка***

У мові програмування C++ є кілька типових форматів файлів, які використовуються для зберігання та обробки даних. Основні з них:

1. ***Текстові файли:*** Це файли, які зберігають дані у звичайному текстовому форматі. Їхній зміст може бути прочитаний та змінений звичайним текстовим редактором. Для роботи з текстовими файлами у C++ використовуються потоки введення/виведення (ifstream та ofstream).

* Плюси: Легко зрозумілі та зручні для редагування людиною, можуть бути читані та оброблятися звичайними текстовими редакторами. Це зроблює їх ідеальним вибором для зберігання конфігураційних файлів, логів, текстових даних тощо.
* Мінуси: Можуть бути менш ефективними для зберігання великих обсягів даних через потребу у просторі для зберігання текстового представлення. Також, обробка текстових файлів може бути менш ефективною у порівнянні з бінарними файлами через необхідність конвертації даних з/у текстовий формат.

1. ***Бінарні файли:*** Ці файли містять дані у вигляді двійкового коду, що дозволяє зберігати дані більш компактно та ефективно. Для роботи з бінарними файлами у C++ також використовуються потоки введення/виведення (ifstream та ofstream), але оперування даними відбувається за допомогою функцій читання та запису в бінарному форматі.

* Плюси: Бінарні файли можуть бути більш ефективними для зберігання великих обсягів даних, оскільки вони не потребують такого великого обсягу простору для зберігання. Також, обробка бінарних даних може бути швидшою, оскільки вони не потребують конвертації в текстовий формат.
* Мінуси: Бінарні файли не придатні для читання та редагування людиною, оскільки їхній зміст представлений у вигляді двійкового коду. Це робить їх менш зручними для відладки та виправлення помилок.

1. ***Формати JSON та XML:*** JSON (JavaScript Object Notation) та XML (Extensible Markup Language) є форматами обміну даними, які широко використовуються для зберігання та передачі структурованих даних між програмами. Для роботи з цими форматами в C++ можна використовувати сторонні бібліотеки.
   * Плюси: JSON та XML дозволяють представляти структуровані дані у зручному для читання та обробки форматі. Вони є популярними форматами для обміну даними між програмами та веб-серверами через їхню розширюваність та простоту використання.
   * Мінуси: JSON та XML можуть бути менш ефективними у порівнянні з бінарними форматами у випадках, коли потрібно зберігати великі обсяги даних. Також, обробка JSON та XML може вимагати додаткових бібліотек або інструментів для парсингу та створення даних.

Кожен з цих форматів має свої переваги та недоліки і вибір конкретного формату залежить від потреб конкретного проекту та зручності роботи з ним.

***Тема 4. Системи контролю версій (VCS)***

Системи контролю версій (VCS) - це програмні засоби, які використовуються для відстеження змін у файлів і коду програм під час розробки програмного забезпечення.

Основна мета VCS полягає в тому, щоб зберігати історію змін, дозволяти співробітникам працювати одночасно над одним проектом та відновлювати попередні версії коду у випадку помилок або потреби у відновленні попередньої функціональності.

Основні плюси систем контролю версій включають:

1. ***Історія змін****:* Всі зміни до коду або файлів зберігаються і можуть бути переглянуті, зроблені коментарі до змін, включаючи інформацію про того, хто та коли зробив зміну.
2. ***Робота з різними версіями****:* Ви можете відновити попередню версію файлу або навіть порівняти дві версії, щоб побачити, що було змінено.
3. ***Гілки розробки****:* Ви можете створювати окремі гілки розробки для роботи над різними функціональностями або різними аспектами проекту паралельно.
4. ***Співпраця****:* Декілька розробників може одночасно працювати над одним проектом, а VCS автоматично об'єднає їхні зміни.

Проте, існують і деякі недоліки:

1. ***Вивчення****:* Використання систем контролю версій вимагає від розробників додаткового часу та зусиль для вивчення їхнього функціоналу та інтерфейсу.
2. ***Розмір бази даних****:* База даних VCS може займати значний обсяг пам'яті, особливо для великих проектів зі значною кількістю історії змін.
3. ***Конфлікти злиття***: При роботі над спільними файлами можуть виникати конфлікти злиття, коли дві гілки розробки мають зміни в одному й тому ж рядку коду.

Централізовані та розподілені системи контролю версій:

* + ***Централізовані системи****:* У централізованих системах весь репозиторій (сховище з версіями файлів) зберігається на центральному сервері, а розробники отримують доступ до нього через мережу. Прикладом централізованої системи є Subversion (SVN).
  + ***Розподілені системи****:* У розподілених системах кожен розробник має свою копію повного репозиторію, з якою вони можуть працювати незалежно. Git є однією з найпопулярніших розподілених систем контролю версій.

Відгалуження та злиття (Branching and Merging):

* + ***Гілки (Branches)****:* Гілки дозволяють розробникам працювати над різними аспектами проекту одночасно без впливу на головну гілку. Кожна гілка може мати свою власну історію змін.
  + ***Злиття (Merging)****:* Злиття використовується для об'єднання змін з однієї гілки в іншу. При виявленні конфліктів злиття розробник повинен їх вирішити, переглянувши та злиттям відповідні рядки коду.

Командна робота (Collaboration):

* + Системи контролю версій дозволяють декільком розробникам працювати над одним проектом одночасно, вносячи та відстежуючи зміни, які здійснюються кожним членом команди.
  + Кожна зміна зберігається разом з інформацією про автора, дату та коментарі, що сприяє зручній співпраці та відстеженню історії розробки.

Робота з галузями (Forking):

* + Деякі системи контролю версій, такі як GitHub, дозволяють розробникам створювати власні копії проектів під назвою "форків". Це дозволяє вносити зміни та експериментувати з кодом, не впливаючи на оригінальний проект.

Відстеження проблем (Issue Tracking):

* + Деякі платформи для спільної роботи, такі як GitHub та GitLab, дозволяють створювати та відстежувати проблеми (issues) або завдання (tasks) пов'язані з проектом. Це полегшує співпрацю та координацію роботи команди розробників.

***Тема 5. Мова Markdown для опису проектів***

Markdown - це легка у використанні мова розмітки, призначена для форматування тексту у веб-засобах. Вона часто використовується для написання різноманітних документів, включаючи опис проектів, README файли, блоги та форуми. Кілька основних переваг та використання мови Markdown для опису проектів:

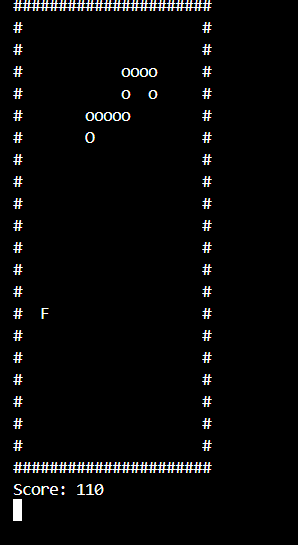
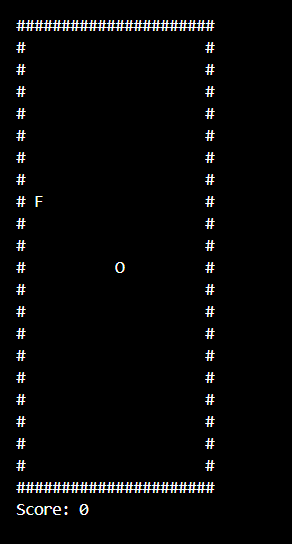
1. ***Простота використання****:* Markdown - це дуже легка мова розмітки, яка проста у вивченні та використанні. Вона використовує прості та зрозумілі синтаксичні правила, такі як використання символів "\*" або "\_" для виділення тексту.
2. ***Читабельність****:* Текст, написаний у форматі Markdown, легко читати у вихідному коді, оскільки він є простим текстом. Він також має чистий вигляд при перегляді на веб-сторінках або інших середовищах.
3. ***Підтримка великої кількості платформ****:* Багато веб-сервісів та платформ, такі як GitHub, GitLab, Bitbucket, Reddit, а також багато блог-систем, підтримують формат Markdown для форматування тексту.
4. ***Можливості розширення****:* Існують різні інструменти та розширення, які дозволяють розширити функціональність Markdown, додаючи підтримку для таблиць, зображень, посилань та інших елементів.
5. ***Інтеграція з іншими мовами програмування****:* Markdown легко інтегрується з іншими мовами програмування, такими як HTML, CSS, а також з системами контролю версій, що дозволяє використовувати його в різних середовищах розробки.
6. ***Підтримка структурованих документів***: Markdown дозволяє легко створювати структуровані документи за допомогою заголовків, списків, блоків коду та інших елементів. Це робить опис проектів більш організованим та зрозумілим для читачів.
7. ***Легкість інтеграції з іншими інструментами****:* Markdown може бути легко інтегрований з різними іншими інструментами та платформами, такими як Jekyll, Hugo, Pelican для створення блогів, або Jupyter Notebook для створення наукових звітів та досліджень.
8. ***Можливості розмітки****:* Markdown підтримує різноманітні можливості розмітки, такі як створення посилань, вставка зображень, таблиць, форматування коду, списків та багато іншого. Це дозволяє створювати документи з різними типами контенту, що підвищує їхню інформативність та привабливість.
9. ***Підтримка спеціалізованих розширень****:* Деякі платформи та середовища розробки підтримують спеціалізовані розширення Markdown, які дозволяють вставляти відео, аудіо, математичні формули та інші складні елементи безпосередньо у документ.
10. ***Крос-платформеність та універсальність****:* Markdown-документи можуть бути відкриті та переглянуті на будь-якому пристрої з будь-яким операційним середовищем, що робить їх універсальними для спільної роботи та обміну інформацією.

Markdown є потужним та зручним інструментом для створення документації проектів. Він надає простий та легкий у використанні спосіб форматування тексту, що дозволяє швидко створювати чіткі та структуровані документи. Плюси використання Markdown включають зручність використання, читабельність, широку підтримку та можливість розширення. Варто також відзначити, що Markdown ідеально підходить для спільної роботи та обміну документацією, оскільки його файлів можна легко читати на будь-якому пристрої. Однак, варто мати на увазі, що для деяких складніших форматувань може знадобитися використання інших мов розмітки. В цілому, Markdown є важливим інструментом для ефективного управління та створення документації в процесі розробки програмного забезпечення.

**Створення ігрової програми**

Мій код є реалізацією простої гри "Змійка" на мові програмування C++. Гравець керує змійкою, яка рухається по полі, намагаючись з'їсти фрукти, щоб збільшити свій рахунок. Основні елементи гри включають змійку, фрукти та логіку руху. Гра завершується, якщо змійка зіткнеться зі стіною або самою собою. У коді реалізовано функції Setup() для початкової настройки гри, Draw() для відображення стану гри на екрані, Input() для обробки введених даних від користувача та Logic() для обчислення логіки гри.

Код демонструє базовий приклад використання мови програмування C++ для створення простої консольної гри. Він показує, як можна використовувати цикли, умовні оператори та змінні для створення ігрового середовища. Цей код може бути використаний як основа для розширення та вдосконалення гри, додавання нових функцій та покращення інтерфейсу користувача.

****

**Висновок**

Під час навчальної практики за розділом "Практична робота з С++" я значно розширила свої знання та здобула цінні практичні навички, які є фундаментальними для подальшого розвитку в галузі програмування. Ось ключові аспекти, які були охоплені під час практики:

1. *Введення та виведення у базових консольних програмах*

Цей розділ дозволив мені опанувати основи взаємодії програм з користувачем через консоль. Я навчилась використовувати стандартні бібліотеки C++ для введення та виведення даних, що є основою для створення більш складних програм. Ці навички є необхідними для розробки ефективних і зрозумілих користувацьких інтерфейсів у консольних додатках.

1. *Робота з файлами: запис та читання*

Практичні завдання з роботи з файлами допомогли мені зрозуміти, як зберігати і обробляти дані у зовнішніх файлах. Це включає навички роботи з текстовими та двійковими файлами, що є критично важливим для створення додатків, які працюють з великими обсягами даних або зберігають дані між сесіями.

1. *Формати файлів та їх обробка*

Ознайомлення з форматами JSON та XML надало мені розуміння структурованого зберігання даних і їх обміну між різними системами. Я навчилася працювати з цими форматами, що дозволяє інтегрувати мої програми з іншими сервісами та додатками, забезпечуючи ефективну обробку даних.

1. *Системи контролю версій (VCS)*

Використання системи контролю версій Git дало змогу зрозуміти принципи управління версіями коду, відстежування змін та співпраці в команді. Ці навички є необхідними для роботи в сучасних командах розробників, де контроль за змінами коду та спільна робота є критично важливими.

1. *Мова Markdown для опису проектів*

Markdown є простою та ефективною мовою розмітки для створення документації. Вивчення та використання Markdown допомогли мені створювати зрозумілі, структуровані та привабливі технічні описи, що полегшує комунікацію в команді та з користувачами.

1. *Створення ігрової програми*

Реалізація гри "Змійка" на мові C++ стала практичним застосуванням всіх вивчених концепцій. Проектування ігрової логіки, обробка введення користувача та використання базових графічних елементів у консолі дозволили мені зрозуміти, як застосовувати теоретичні знання на практиці. Цей проект надав мені впевненості у своїх навичках програмування та вміння вирішувати складні завдання.

Ця практика стала важливим етапом у моєму професійному розвитку. Отримані знання та навички створили міцну базу для подальшого навчання та роботи у сфері програмування. Я впевнена, що зможу успішно застосовувати їх у своїй майбутній кар'єрі, вирішуючи складні завдання та створюючи ефективні та якісні програмні рішення.