СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) *Інформаційних технологій та електроніки*
(повне найменування інституту, факультету)

Кафедра *Інформаційних технологій та програмування*
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА
до кваліфікаційної випускної роботи

освітній ступінь *бакалавр*
(бакалавр, магістр)

спеціальність *126 «Інформаційні системи та технології»*
(шифр і назва спеціальності)

спеціалізація *Інформаційні системи та технології*
(назва спеціалізації)

на тему *Використання мов C# і C++ при реалізації алгоритмів інформаційних систем*

Виконав: студент групи ІСТ-21д _____ К.А. Марценюк
(підпис) (ініціали і прізвище)

Керівник _____ В.О. Лифар
(підпис) (ініціали і прізвище)

Завідувач кафедри _____ О.І. Захожай
(підпис) (ініціали і прізвище)

Рецензент Ратов Д.В.

Київ - 2025

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) _*Інформаційних технологій та електроніки*___
(повне найменування інституту, факультету) .

Кафедра _*Інформаційних технологій та програмування*_____
(повна назва кафедри)

Освітній ступінь _____*бакалавр*_____
(бакалавр, магістр)

спеціальність _*126 «Інформаційні системи та технології»*_____
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____
“____”
_____20___року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

_____*Марценюк Катерина Анатоліївна*_____
(прізвище, ім’я, по батькові)

1. Тема роботи__*Використання мов C# і C++ при реалізації алгоритмів інформаційних систем*_____
_____
_____

Керівник роботи__доцент, д.т.н. Лифар Володимир Олексійович_____,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджений наказом університету від “27” _травня__2025 року №67/15.15-С_ .

2. Строк подання студентом роботи_____18.06.2025_____

3. Вихідні дані до роботи_ *Об'єктом даної роботи є процес порівняння можливостей мов програмування C# і C++ при реалізації алгоритмів в інформаційних системах.*_____
_____
_____
_____

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_*Вступ. Огляд інформаційних систем та їх алгоритмів. Загальний огляд мов програмування. Порівняльний аналіз C# і C++. Сфери застосування мов. Висновок.*_____
_____
_____
_____
_____

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників)
_____
_____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 13.03.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Одержання завдання на виконання роботи	13.03.25	Виконано
2	Укладання і погодження з керівником плану і етапів виконання роботи	17.03.25	Виконано
3	Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі»	12.04.25	Виконано
4	Аналіз шляхів виконання завдання	22.05.25	Виконано
5	Укладання та порівняння алгоритмів	02.06.25	Виконано
6	Укладання, оформлення та погодження пояснювальної записки з керівником	10.06.25	Виконано
7	Здача готової пояснювальної записки на кафедру	18.06.25	Виконано
8	Укладання доповіді і презентації	19.06.25	Виконано

Студент _____ К.А. Марценюк
(підпис) (ініціали і прізвище)

Керівник роботи _____ В.О. Лифар
(підпис) (ініціали і прізвище)

РЕФЕРАТ

Робота містить: 56 сторінок основного тексту, 0 сторінок додатків, 2 рисунка, 12 таблиць, 4 корисних джерел.

Метою випускної кваліфікаційної роботи є аналіз та порівняння можливостей мов програмування C# і C++ при реалізації алгоритмів в інформаційних системах.

При написанні дипломної роботи було проаналізовано особливості мов програмування C# і C++, їх можливості у реалізації алгоритмів в інформаційних системах, виявлено переваги та недоліки кожної мови, а також визначено сфери доцільного застосування.

Також розглянуто особливості синтаксису, управління пам’яттю, типів даних і колекцій, принципи побудови інформаційних систем, а також реалізацію базових алгоритмів, таких як сортування та пошук.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІС – інформаційні системи

ІТ – інформаційні технології

ООП - об’єктно-орієнтоване програмування

БД – база даних

ПЗ - програмне забезпечення

МФУ - багатофункціональний пристрій

DSS (Decision Support System) – підтримка прийняття рішень

ERP (Enterprise Resource Planning) – планування ресурсів

CRM (Customer Relationship Management) – програмне забезпечення

HRM (Human Resource Management) – системне управління

API (Application Programming Interface) – набір різних правил

REST (Representational State Transfer) – архітектурний стиль для розробки веб-сервісів

SOAP (Simple Object Access Protocol) – протокол обміну повідомленнями

XML (eXtensible Markup Language) – розширювана мова розмітки

CSV (Comma-Separated Values) – текстовий формат файлу

ESB (Enterprise Service Bus) - корпоративна сервісна шина

RAM (Random Access Memory) - оперативна пам'ять

CPU (Central Processing Unit) – центральний процесор

HDD (Hard Disk Drive) - жорсткий диск

SSD (Solid State Drive) – накопичувач

GIMP (GNU Image Manipulation Program) - растровий графічний редактор

SQL (Structured Query Language) - мова структурованих запитів

AES (Advanced Encryption Standard) - розширений стандарт шифрування

RSA (Rivest, Shamir, Adleman) - криптографічний алгоритм

.NET - платформа від Microsoft

STL (Standard Template Library) - бібліотека для C++

IoT (Internet of Things) – концепція мережі

LINQ (Language Integrated Query) - мова структурованих запитів

JIT (Just-In-Time) – компіляція

CIL (Common Intermediate Language)/ MSIL (Microsoft Intermediate Language)– проміжна мова для платформи .NET Framework

WPF (Windows Presentation Foundation) - десктопні додатки у Windows

ML.NET – машинне навчання для .NET

MAUI (Multi-platform App UI) - інтерфейс додатків .NET

IDE (Integrated Development Environment) – середовище розробки

RAII (Resource Acquisition Is Initialization) - управління ресурсами

IL (Instruction List) – мова програмування

CLR (Common Language Runtime) – загальномовне виконующе середовище

UWP (Universal Windows Platform) - універсальна платформа Windows

CAD (Computer-Aided Design) - система автоматизованого проектування

ВСТУП

Все в нашому світі пов'язане з інформаційними системами та технологіями. Вони проникають у всі сфери життя, формуючи нову цифрову реальність. Сучасне суспільство вже неможливо уявити без інтернету, мобільних пристроїв, хмарних сервісів та штучного інтелекту. Інформаційні технології управляють процесами на підприємствах, забезпечують зв'язок між людьми по всьому світу, допомагають в освіті, медицині, науці і навіть у повсякденному житті. Вони роблять інформацію доступною, процеси швидкими, а рішення більш точними. У той же час з розвитком технологій виникають нові завдання та ризики: захист персональних даних, кібербезпека, цифрова грамотність. Саме тому розуміння та правильне використання інформаційних систем стає ключовою навичкою у 21 столітті.

Але особливо стрімко розвивається програмування — основа, де будується цифровий світ.

Програмування — це мова технологій, за допомогою якої створюються додатки, веб-сайти, системи управління, штучний інтелект і багато іншого.

Програмування буває різне - залежно від завдань, областей застосування та рівня абстракції. Існують десятки мов програмування, кожна з яких підходить для певних цілей. Крім того, програмування можна класифікувати за стилями: об'єктно-орієнтоване, функціональне, процедурне та навіть візуальне програмування.

Мови програмування - це набір спеціальних (формальних) правил, якими пишуть різні програми. Програми можуть бути різними в залежності від їх призначення, способу роботи та розповсюдження. Популярними мовами програмування, що часто вживаються, є C# і C++.Вони мають загальну основу, оскільки обидві мови походять від C і підтримують об'єктно-орієнтоване програмування (ООП), але різняться за принципами роботи управління пам'яттю, швидкості виконання та застосування.

C++ і C# мають багато подібних характеристик: ООП, базовий синтаксис, роботу з потоками та пам'яттю. Однак C++ більше орієнтований

на продуктивність та низькорівневе програмування, а C# - на зручність, безпеку та бізнес-додатки.

1.1 Актуальність теми

Актуальність теми обумовлена тим, що мови програмування C# і C++ залишаються одними з найбільш затребуваних інструментів у розробці програмного забезпечення завдяки своїй потужності та універсальності. Їх порівняння дозволяє визначити оптимальні сфери застосування кожної мови, що особливо важливо в умовах швидкого розвитку технологій та збільшення вимог до якості та продуктивності програм. Глибоке розуміння їхніх переваг та обмежень допомагає розробникам приймати обґрунтовані рішення при виборі технологій для реалізації проектів.

1 ОГЛЯД ІНФОРМАЦІЙНИХ СИСТЕМ ТА ЇХ АЛГОРІТМІВ

1.1 Поняття ІС

Інформаційна система(ІС) – сукупність взаємопов’язаних компонентів, які включають людей, програмне забезпечення, апаратне забезпечення, дані та процедури, що функціонують разом для збирання, зберігання, обробки, аналізу та розповсюдження інформації з метою підтримки прийняття рішень, координації, контролю, аналізу та візуалізації в межах організації чи в повсякденному житті.

В інформаційній системі здійснюються такі основні процеси:

- введення даних, отриманих із різних джерел інформації;
- опрацювання або перетворення цієї інформації;
- зберігання як початкових, так і оброблених даних;
- виведення інформації, підготовленої для використання користувачем;
- передача та отримання інформації через мережу.

Розробка інформаційної системи включає виконання двох ключових завдань:

- заповнення системи даними, що стосуються конкретної предметної області;
- розробка (переважно з використанням графічного) інтерфейсу користувача для забезпечення доступу до потрібної інформації.

Дані в інформаційній системі можуть зберігатися як у структурованому, так і в неструктурованому вигляді.

Структурування даних — це насамперед процес організації та впорядкування інформації таким чином, щоб вона була зручною для зберігання, пошуку, обробки та аналізу.

Цей процес включає:

- розподіл даних на логічно обґрунтовані категорії;
- визначення формату представлення (таблиці, списки, графіки, бази даних тощо);
- встановлення зв’язків між окремими елементами;
- гарантування точності та цілісності інформації.

Метою цього процесу є забезпечення ефективного зберігання, швидкого доступу, зручної обробки та надійного використання даних у межах інформаційної системи. Завдяки структурованим даним можна скоротити повторення інформації, підвищити її точність, полегшити аналіз і гарантувати узгодженість між різними модулями або підсистемами ІС.

Наприклад, підприємство веде таблицю з працівниками, де вся інформація чітко структурована та впорядкована. У такій таблиці кожен запис (рядок) відповідає конкретному працівнику, а кожне поле (стовпець) містить визначений тип даних — прізвище, ім’я, посаду, ідентифікаційний номер, дату прийняття на роботу, заробітну плату тощо. Завдяки цьому підприємство може швидко знаходити необхідну інформацію, формувати звіти, проводити аналіз кадрового складу або здійснювати розрахунки заробітної плати.

№	Прізвище	Ім’я	Посада	Табельний №	Дата прийняття	Зарплата (₴)
1	Устинова	Марія	Бухгалтер	00121	2018-05-12	30 000
2	Кравець	Наталія	Маркетолог	00122	2020-10-02	15 000
3	Швець	Андрій	Оператор	00123	2019-07-21	20 000
4	Герасименко	Денис	Інженер-технолог	00124	2021-11-13	25 000
5	Удовенко	Софія	Економіст	00125	2022-02-	15 000

Неструктуровані дані — текстові документи, які можуть містити ілюстрації, серед яких статті, реферати, журнали, книги тощо. Системи, призначені для зберігання таких даних, зазвичай не можуть надати чітку і конкретну відповідь на запит користувача. Замість цього вони переважно надають сам текст документа або перелік матеріалів, що потребують подальшого опрацювання для відшукання потрібної інформації.

Неструктуровані дані можуть включати:

- Електронну пошту (текст, вкладення, підписи)
- Зображення, відео та аудіофайли
- Повідомлення з месенджерів і соціальних мереж
- Документи у форматах Word або PDF, створені без конкретного шаблону
- Відгуки користувачів та коментарі
- Вміст веб-сторінок, блогів і форумів

Особливості:

- Значний обсяг даних (часто називають «Big Data»)
- Не підлягають прямому сортуванню чи фільтрації
- Потребують попередньої обробки для отримання ключової інформації (зокрема, із застосуванням штучного інтелекту)

Наприклад, підприємство отримує електронний лист, у якому містяться різноманітні матеріали — текстова інформація, зображення, документи у форматах PDF чи Word. Такий лист не має чітко визначеної структури: дані можуть бути подані у довільному порядку, без стандартизованих полів або форматування.

1.2 Структура та функціональні компоненти ІС

Створення ефективної інформаційної системи ґрунтується на наборі загальних принципів, які гарантують її функціональність, надійність, гнучкість і відповідність вимогам користувачів. Дотримання цих принципів сприяє розробці систем, що легко інтегруються, масштабуються та адаптуються до змін.

Принципи побудови інформаційних систем

№	Принцип	Суть принципу
1	Системний підхід	ІС являє собою інтегровану структуру, що об'єднує взаємопов'язані компоненти, зокрема апаратні, програмні, організаційні та людські, які спільно забезпечують виконання її призначення.
2	Цілеспрямованість	Кожна ІС повинна мати чітко сформульовану мету, наприклад, автоматизацію обліку, допомогу в ухваленні рішень або оптимізацію управлінських процесів.
3	Модульність	Система розробляється у вигляді набору окремих модулів або підсистем, кожна з яких виконує визначену функцію, наприклад, управління персоналом чи складський облік. Такий підхід полегшує обслуговування, оновлення та розширення інформаційної системи.
4	Інтеграція	Усі компоненти ІС мають бути взаємно сумісними та функціонувати як єдиний механізм. Завдяки інтеграції можна уникнути повторення даних і

		гарантувати узгодженість та цілісність інформації.
5	Автоматизація	Ключові процеси збору, обробки та передачі інформації мають бути максимально автоматизовані. Це дозволить мінімізувати вплив людського фактора та покращити загальну ефективність системи.
6	Відкритість і масштабованість	Система має бути здатною адаптуватися до змінних вимог, забезпечувати легке масштабування та інтеграцію з іншими системами чи платформами.
7	Безпека	Захист даних від несанкціонованого доступу, а також забезпечення конфіденційності, цілісності та доступності інформації мають ключове значення для будь-якої інформаційної системи.
8	Зручність для користувача	ІС повинен бути простим у використанні, зручним і гнучким для задоволення потреб користувачів. Це сприяє підвищенню продуктивності та скорочує час, необхідний для навчання персоналу.
9	Стандартизація	Проектування ІС повинно базуватися на застосуванні загальновизнаних стандартів, зокрема апаратних, програмних та протокольних. Це сприяє забезпеченню сумісності й значно полегшує їх подальший розвиток.
10	Економічна доцільність	Система має бути економічно

		доцільною для впровадження: її розробка, впровадження й утримання повинні відповідати фінансовим можливостям організації та забезпечувати відчутну користь.

Однак велика кількість сфер і форм використання сучасних інформаційних технологій призводить до значного різноманіття методів класифікації ІС.

Інформаційні системи (ІС) поділяють за різними ознаками залежно від цілей використання, рівнів управління, способів обробки даних, виконуваних функцій та інших факторів. Подібна класифікація дає змогу глибше розуміти їхню структуру, можливості та області застосування.

№	Критерій класифікації	Тип ІС	Характеристика
1	За рівнем управління	Оперативні	Забезпечують виконання рутинних, щоденних операцій (наприклад, облік продажів).
		Тактичні	Підтримують середньострокове планування та аналіз діяльності підрозділів.
		Стратегічні	Спрямовані на довгострокове планування та прийняття управлінських рішень.
2	За сферою застосування	Банківські, медичні, освітні тощо	Спеціалізовані ІС для конкретної галузі, адаптовані під її

			вимоги.
3	За функціональним призначенням	Управління БД	Забезпечують створення, зберігання, обробку та запити до баз даних.
		Інформаційно-довідкові	Надають швидкий доступ до збереженої інформації для користувачів.
		DSS (Підтримки рішень)	Аналізують варіанти, прогнозують наслідки та допомагають приймати рішення.
		Експертні системи	Використовують бази знань для моделювання рішень експерта в конкретній сфері.
4	За способом обробки даних	Ручні	Обробка здійснюється людиною без застосування програмних засобів.
		Автоматизовані	Частина операцій виконується автоматично, частина — вручну.
		Автоматичні	Усі процеси повністю автоматизовані (напр., ІС на виробництві).
5	За взаємодією з користувачем	Локальні	Працюють на одному комп’ютері або

			обмеженій мережі без зовнішнього доступу.
		Мережеві (клієнт-серверні)	Користувачі отримують доступ через мережу до центрального сервера.
		Веб-орієнтовані	Працюють через браузер, мають доступ із будь-якого місця з інтернетом.
		Хмарні	Розміщені на віддалених серверах, доступні за підпискою або ліцензією.
6	За ступенем охоплення процесів	Часткові	Автоматизують окремі бізнес-процеси (наприклад, зарплатна система).
		Комплексні (ERP, CRM, HRM)	Охоплюють усі або більшість процесів підприємства.
7	За часом обробки	Пакетні	Дані обробляються партіями у визначений час (наприклад, звіти раз на добу).
		Інтерактивні (реального часу)	Дані обробляються та відображаються миттєво після введення.
8	За способом реалізації	Програмні	Побудовані переважно на програмних

			рішеннях без спеціалізованого обладнання.
		Програмно-апаратні	Поєднують програмне забезпечення та спеціальні пристрої (касові апарати, сканери).

ІС зазвичай не працюють у відриві одна від одної. Для забезпечення ефективної діяльності підприємства або організації різноманітні ІС повинні співпрацювати, обмінюючись даними, викликами функцій чи послугами. Ця взаємодія може відбуватися за кількома різними моделями.

Типи взаємодії інформаційних систем

№	Тип взаємодії	Суть / Характеристика
1	Вертикальна взаємодія	Взаємодія між ІС різних рівнів управління — наприклад, від локального відділу до центральної системи.
2	Горизонтальна взаємодія	Взаємодія між системами одного рівня або одного типу — наприклад, обмін даними між двома філіями.
3	Клієнт-серверна взаємодія	Один елемент (клієнт) надсилає запити, інший (сервер) їх обробляє й надає результат.
4	Сервіс-орієнтована взаємодія	Взаємодія через вебсервіси або API, які надають функції для зовнішніх систем (напр., REST, SOAP).
5	Через спільну базу даних	Кілька ІС використовують одну базу даних для зчитування та запису інформації.
6	Файловий обмін	Системи обмінюються даними через

		спільні або синхронізовані файли (наприклад, XML, CSV).
7	Подієва (event-driven)	Взаємодія через повідомлення або сигнали про події — система реагує на зміни в іншій системі.
8	Інтеграція через шину даних (ESB)	Центральна шина даних пов’язує всі системи і забезпечує маршрутизацію повідомлень між ними.

Приклади:

CRM-система, що працює на горизонтальному рівні, використовується в банках для обміну даними з бухгалтерською системою через API, реалізуючи сервіс-орієнтовану взаємодію.

ERP-система, яка застосовується у виробництві, отримує вхідні замовлення від веб-порталу клієнтів через шину даних або API, що забезпечує швидкий та узгоджений обмін інформацією між підсистемами.

У сучасному інформаційному суспільстві існує велика кількість ІС, які відрізняються рівнем автоматизації, технічними особливостями та призначенням. Утім, усі вони мають дві ключові складові:

1. Технічна частина — апаратне забезпечення, яке забезпечує функціонування системи. До нього належать комп’ютери, мережеве обладнання, сервери, пристрої для введення та виведення даних та інші елементи.

2. Програмна частина — комплекс програмного забезпечення, призначеного для обробки, зберігання, передавання та представлення інформації користувачам.

Окрім технічного та програмного забезпечення, більшість сучасних ІС включають ще кілька ключових компонентів:

3. Інформаційна складова — це сукупність даних і знань, які постійно використовуються в системі: початкові дані, бази даних, документи, довідники, звіти тощо. Якість і достовірність цієї інформації безпосередньо впливають на ефективність роботи системи.

4. Організаційно-методична складова — охоплює правила, інструкції, алгоритми та методики використання системи. Вона регулює, як система застосовується в рамках конкретної організації чи підприємства.

5. Людський фактор (користувачі) — навіть за високого рівня автоматизації будь-яка інформаційна система потребує участі людей. Це можуть бути звичайні користувачі, адміністратори, аналітики чи розробники. Вони займаються налаштуванням, підтримкою і прийняттям рішень на основі оброблених системою даних.

Будь-який комп’ютер за своєю конструкцією складається з чотирьох ключових компонентів, кожна з яких виконує конкретну функцію:

1. Пристрої введення — відповідають за введення інформації в систему (наприклад, клавіатура, миша, сканер, чи мікрофон).

2. Пристрої обробки — займаються виконанням обчислень та управлінням процесами (зокрема, центральний процесор (CPU) і оперативна пам’ять (RAM)).

3. Пристрої зберігання — призначені для запису та зберігання даних і програм (такі як жорсткі диски (HDD), твердотілі накопичувачі (SSD), або флеш-накопичувачі).

4. Пристрої виведення — слугують для передачі обробленої інформації користувачу (до них належать монітор, принтер, акустичні системи тощо).

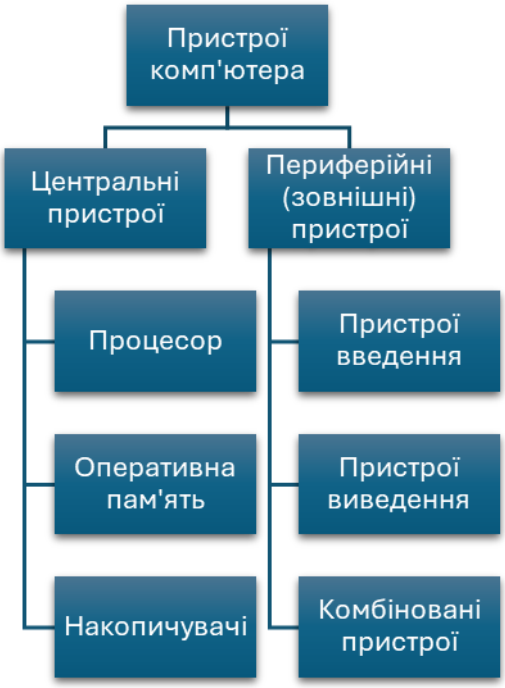
Комп'ютерні пристрої традиційно поділяються на дві основні категорії за їхньою функціональною роллю:

1. Центральні пристрої – це внутрішні компоненти, розташовані в системному блоці. До них належать:

- Процесор (CPU), який виконує всі обчислення та керує операціями;
- Оперативна пам’ять (RAM), що забезпечує тимчасове зберігання даних для швидкого доступу;
- Материнська плата, відеокарта, блок живлення, жорсткі диски та твердотільні накопичувачі (HDD/SSD) та інші елементи.

2. Периферійні (зовнішні) пристрої – це ті елементи, які підключаються до системного блоку і слугують для взаємодії користувача з комп’ютером:

- Пристрої введення, такі як клавіатура, миша, сканер, мікрофон;
- Пристрої виведення, до яких належать монітор, принтер, аудіоколонки;
- Комбіновані пристрої, наприклад, сенсорні екрани, багатофункціональні пристрої (МФУ), модеми та інші.



Існує два основних типи програмного забезпечення:

1. Системне програмне забезпечення

До цього типу відносять програми, що забезпечують базову роботу комп’ютера, керують його ресурсами та створюють умови для функціонування прикладного програмного забезпечення.

Ця категорія включає:

- Операційні системи (Windows, Linux, macOS);
- Драйвери пристроїв;
- Утиліти для обслуговування системи (антивірусне програмне забезпечення, інструменти для резервного копіювання тощо).

2. Прикладне програмне забезпечення

Це програми, які виконують конкретні завдання користувача залежно від його потреб або галузі діяльності.

До такого програмного забезпечення належать:

- Офісні програми (Word, Excel, PowerPoint);
- Веббраузери (Chrome, Firefox);
- Графічні редактори (Photoshop, GIMP);
- Програми для бухгалтерського обліку, освітні платформи, ігри та інші.

Сучасна ІС взаємодіє з іншими платформами, здійснюючи процеси передачі та отримання даних. Вона формує запити до джерел інформації та отримує у відповідь потрібну інформацію. У свою чергу, користувачі також направляють свої запити до системи, яка обробляє їх і надає відповідні відповіді.

Схема роботи сучасної інформаційної системи

Сучасна ІС виконує роль посередника між джерелами даних та користувачами. Її функціонування можна розділити на кілька основних етапів:

1. Споживачі звертаються до інформаційної системи для отримання потрібних даних.
2. У разі відсутності необхідної інформації, система формує запити до зовнішніх джерел, таких як бази даних, сенсори або сторонні сервіси.
3. Джерела надають запитувану інформацію у відповідь.
4. Система проводить обробку отриманих даних: виконує аналіз, фільтрацію та структурування.
5. Оброблені результати надсилаються споживачам у вигляді відповідей на їхні запити.

Основні структурні рівні ІС

№	Рівень	Опис	Основні елементи
1	Технічний (апаратний)	Забезпечує фізичну основу для роботи ІС. Охоплює все обладнання, що використовується для збору, обробки, зберігання та передачі даних.	Сервери, комп’ютери, мережеве обладнання, сховища даних, периферія
2	Програмний	Здійснює управління	Операційні

		апаратними ресурсами та забезпечує виконання інформаційних процесів.	системи, прикладне ПЗ, бази даних, драйвери
3	Інформаційний	Відповідає за збереження, організацію та забезпечення доступу до даних, необхідних для функціонування системи.	Бази даних, масиви даних, знання, документи
4	Організаційний	Містить правила, регламенти та процедури, які визначають порядок використання інформаційної системи в організації.	Регламенти роботи, бізнес-процеси, стандарти, інструкції
5	Людський	Охоплює всіх осіб, які взаємодіють із ІС: як користувачів, так і технічний персонал	Кінцеві користувачі, адміністратори, програмісти, аналітики
6	Комунікаційний (мережевий)	Забезпечує передачу даних між компонентами ІС або з зовнішніми системами.	Мережі, протоколи зв’язку, інтернет, канали передачі даних

ІС розроблена для ефективного управління інформаційними процесами в межах організації чи певної сфери діяльності. Вона виконує важливі функції, охоплюючи всі етапи життєвого циклу даних — від їх збору до подальшого використання.

Інформаційна технологія (ІТ) охоплює методи, інструменти та процеси, що забезпечують збирання, зберігання, обробку, передачу та застосування інформації за допомогою комп’ютерних і телекомунікаційних систем.

Іншими словами, це технології, які дозволяють ефективно, точно та автоматизовано працювати з даними.

Основні умови для успішного функціонування ІС:

№	Умова	Пояснення
1	Технічне забезпечення	Сучасні комп’ютери, сервери, мережі, обладнання для введення та виведення.
2	Програмне забезпечення	Надійне ПЗ для обробки, зберігання та аналізу інформації.
3	Якісні дані	Актуальні, повні та достовірні вхідні дані.
4	Кваліфікований персонал	Адміністратори, аналітики, розробники, користувачі з необхідними навичками.
5	Організація обробки інформації	Чіткі алгоритми обробки, швидкість і точність обчислень.
6	Захист і безпека інформації	Системи резервного копіювання, шифрування, контроль доступу.
7	Зручний користувацький інтерфейс	Інтуїтивне та просте у використанні середовище взаємодії з системою.
8	Підтримка й обслуговування	Регулярне оновлення ПЗ, технічна підтримка, навчання персоналу.
9	ІТ-інфраструктура	Наявність Інтернету, локальних мереж, стабільного живлення, резервних каналів.
10	Підтримка з боку керівництва	Організаційна та фінансова підтримка, стратегічне бачення впровадження ІС.

Функціональні компоненти ІС

Функціональні компоненти являють собою ключові елементи, відповідальні за реалізацію завдань ІС, включаючи збір, обробку, зберігання та передачу даних. Кожен із компонентів відіграє унікальну роль на різних етапах життєвого циклу інформації.

№	Компонент	Функції / Опис
1	Збирання даних	Отримання первинної інформації з джерел (сенсори, форми, документи тощо).
2	Обробка даних	Перетворення сирих даних у зручну для аналізу або зберігання форму.
3	Зберігання даних	Організація баз даних для надійного та довготривалого зберігання інформації.
4	Передавання даних	Комунікація між компонентами ІС, обмін інформацією між користувачами та системами
5	Виведення інформації	Формування результатів у вигляді звітів, графіків, повідомлень тощо.
6	Керування процесами	Координація роботи системи, забезпечення логіки виконання завдань.
7	Захист інформації	Автентифікація, контроль доступу, шифрування, резервне копіювання.
8	Інтерфейс користувача	Засоби взаємодії користувача з системою (вікна, меню, кнопки, веб-інтерфейс).
9	Підтримка прийняття рішень	Аналітичні модулі, прогнози, моделі, що допомагають у виборі варіантів дій.
10	Інтеграція з іншими системами	Механізми обміну даними з зовнішніми або суміжними інформаційними системами.

1.3 Алгоритми в ІС

Алгоритм являє собою чітко визначену, формалізовану послідовність кроків, яка призначена для досягнення конкретного результату або розв'язання певної задачі. У сфері ІС алгоритми складають основу роботи програмного забезпечення, сприяючи автоматизації процесів, пов'язаних із обробкою, зберіганням, аналізом та передаванням даних.

Алгоритми використовуються повсюди, не тільки в ІТ-сфері, але й у повсякденному житті, медицині, освіті, транспорті, праві, бізнесі та інших галузях. Вони лежать в основі будь-якої діяльності, де важливо дотримуватися чіткої послідовності дій для досягнення конкретного результату. Завдяки алгоритмам процеси стають більш організованими, передбачуваними та ефективними.

Ось пару прикладів алгоритмів:

Перше із повсякденного життя – це Приготування страви. В ньому використовується така послідовність дій: Підготувати інгредієнти → Виконати кулінарні дії у певній послідовності → Подати страву.

Друге — з програмування — це написання коду. Тут виконується така послідовність: Визначення завдання → Розробити алгоритм обробки даних → Написання коду → Виконати тестування → Виправлення помилок → Запуск програми.

Виконання алгоритму повинно відбуватися в суворо встановленому порядку. Збій у послідовності або пропуск хоча б одного пункту, швидше за все, призведе до отримання некоректного результату. Алгоритм має ряд специфічних характеристик, які вирізняють його серед інших інструкцій. Щоб набір інструкцій вважався алгоритмом, він повинен мати кілька ключових властивостей:

1. Скінченність (алгоритм має завершуватися після досягнення визначеної кількості кроків)

2. Дискретність (алгоритм складається з послідовності чітко визначених етапів)

3. Визначеність (однозначність) (кожен алгоритм має містити команди, які є зрозумілими та однозначними для виконавця)

4. Масовість (алгоритм повинен бути застосовним до цілого спектра схожих завдань)

5. Результативність (алгоритм має забезпечувати отримання конкретного результату після його виконання)

6. Формалізованість (алгоритм повинен бути описаний такою мовою, яка є зрозумілою та доступною для виконавця, чи то людини, чи то машини)

1.4 Використання алгоритмів в ІС

Роль алгоритмів в ІС:

№	Застосування	Приклад алгоритму
1	Обробка даних	Алгоритми, що охоплюють сортування, фільтрацію та агрегацію, використовуються для обробки та аналізу даних. Наприклад, можна застосувати алгоритм для впорядкування працівників за рівнем заробітної плати. Це дає змогу зручно організувати інформацію, виділити співробітників із найвищими чи найнижчими доходами або навіть проводити подальший аналіз на основі отриманих результатів.
2	Пошук і фільтрація інформації	Алгоритми пошуку в базах даних передбачають фільтрацію записів на основі заданих умов за допомогою SQL-запитів.
3	Аналіз та	Алгоритми прогнозування попиту, методи

	прогнозування	кластеризації та регресійний аналіз.
4	Оптимізація ресурсів	Алгоритми для розподілу завдань, оптимізації витрат і планування маршрутів.
5	Шифрування та безпека	Криптографічні алгоритми, такі як AES та RSA, використовуються для забезпечення захисту даних.
6	Автоматизація процесів	Алгоритми управління документообігом, організації логістичних процесів і надсилання повідомлень.
7	Інтелектуальний аналіз даних (Data Mining)	Алгоритми машинного навчання призначені для виявлення закономірностей та трендів у великих обсягах даних. Вони аналізують інформацію, знаходять приховані залежності та використовують ці знання для прийняття рішень або прогнозування.

Основні структурні класи алгоритмів за типом дій:

- Лінійні — послідовне виконання команд (наприклад, обчислення суми).
- Розгалужені — з умовами (if/else, наприклад, перевірка наявності даних).
- Циклічні — повторення дій (наприклад, аналіз усіх записів у базі).
- Рекурсивні — алгоритми, які викликають самі себе (наприклад, обходи дерев).

Форми подання алгоритму

Алгоритм, як абстрактна послідовність дій, може мати різноманітні форми представлення. Кожна з них має свої сильні та слабкі сторони і підбирається в залежності від завдання: розробки, документування, навчання чи безпосереднього виконання.

№	Форма подання	Характеристика	Приклад

1	Словесна (текстова)	Алгоритм формулюється у вигляді послідовності дій, поданих у формі інструкції.	Рецепт приготування страви
2	Графічна (Блок-схема)	Алгоритм представляється у формі блок-схеми, в якій кожен етап відображено у вигляді окремого блоку.	Сортування чисел
3	Таблична	Дії алгоритму представлені покроково у таблиці, що включає пояснення або відповідні умови.	Наукова стаття
4	Псевдокод	Алгоритм описується у формі, близькій до програмного коду, але без використання специфічного синтаксису конкретної мови програмування.	Розробка алгоритмів
5	Програмний Код	Алгоритм представлений у текстовому вигляді, написаному на якійсь мові програмування. Це найбільш детальна і точна форма, яку можна безпосередньо виконати на комп'ютері.	C#, C++, Java тощо

2 ЗАГАЛЬНИЙ ОГЛЯД МОВ ПРОГРАМУВАННЯ

Мова програмування - це офіційна мова, на якій розробники пишуть інструкції (алгоритми) на комп’ютер. Вони виступають посередниками між людьми та машинами, що дозволяє людям створювати програми, ігри, веб - сайти, системи управління тощо. Проте комп’ютери розуміють лише машинний код (0 і 1). Тому між написаним людиною кодом і виконанням його машиною відбувається переклад (трансляція).

Класифікація мов програмування

Тип	Опис
Машинні мови	Найнижчий рівень — взаємодіє безпосередньо з процесором.
Асемблерні мови	Більш читабельні, але ще низькорівневі
Мови високого рівня	Зрозумілі людині, зручно писати код
Об’єктно-орієнтовані мови	Орієнтовані на об’єкти (структури з даними та методами)
Скриптові мови	Спрямований на автоматизацію, веб-розробку та інтеграцію
Функціональні мови	Побудовані на математичних функціях
Декларативні мови	Описують, що робити, а не як

Основні парадигми програмування

Парадигма програмування — це стиль або модель, що визначає спосіб написання програм. Вона впливає на те, як програміст формулює логіку програми та які структури і підходи використовує.

1. Імперативна парадигма (програміст точно визначає порядок дій, які повинен виконати комп’ютер)

2. Декларативна парадигма (програміст пояснює, що потрібно виконати, а не як саме це реалізувати)

3. Об’єктно-орієнтована парадигма (ООП) (програма побудована на основі об'єктів, що включають дані та методи)

4. Функціональна парадигма (програма містить функції, що не викликають побічних ефектів)

5. Логічна парадигма (програміст формулює факти та правила, а система самостійно знаходить рішення)

Мови програмування застосовуються в різних сферах людської діяльності. Кожна мова має свої сильні сторони і найкраще підходить для певного типу задач.

2.1 Коротка історія C# і C++

C# - це мова програмування, яка зосереджується на об'єктах та даних, також заснована на строгій компонентній архітектурі та реалізує передові механізми забезпечення безпеки коду. Він дотримується принципів об'єктно-орієнтованого програмування (ООП), що робить його ідеальним інструментом для опису та моделювання складних систем. C# підтримує об'єктно-орієнтоване програмування: інкапсуляцію, успадкування та поліморфізм. Також має більш високий рівень абстракції порівняно з C++, що спрощує процес розробки та знижує ризик помилок, пов'язаних з управлінням пам'яттю. Проте C# має обмежену сумісність з іншими екосистемами, та її продуктивність може поступатися C++ в деяких високонавантажених додатках.

C# був створений компанією Microsoft наприкінці 1990-х років. Але він був представлений світу у 2000-х рр. разом із випуском першої версії .NET

Framework. Авторами цієї мови програмування стали Скотт Вілтамут та Андерс Хейльсберг — творець Турбо Паскаля та Дельфі, який перейшов у 1996 році в Microsoft. Він став частиною платформи .NET і призначений для розробки різних програм - від настільних до веб-додатків та ігор. Мова програмування C# була створена спеціально для ASP.NET. На C# повністю була написана сама ASP.NET. Історія створення C# пов'язана з бажанням компанії створити потужну, безпечну та зручну мову для розробки програмного забезпечення на своїй новій платформі, в основному створювалася для Windows та .NET Framework.

C++ - це мова з багатим функціоналом, яка надає розробнику повний контроль над пам'яттю та можливістю глибокої оптимізації. Він підтримує основні концепції ООП, такі як класи, об'єкти, успадкування, інкапсуляція та поліморфізм.

C++ був створений в 1983 році Бйорном Страуструп як розширення мови програмування C. Його початкова мета полягала в тому, щоб зробити C більш гнучкою і потужною мовою, додавши можливості об'єктно-орієнтованого програмування та інших концепцій. Він є багатопарадигмальною мовою програмування, що означає, що дозволяє використовувати кілька різних стилів програмування. C++ широко використовується для розробки програмного забезпечення, а сфера його застосування включає операційні системи, прикладні програми, драйвери пристроїв, програми для вбудованих систем, високопродуктивні сервери, а також розважальні програми.

2.2 Характеристика мов програмування

C# і C++ — це дві популярні та потужні мови програмування, кожна з яких має унікальні особливості та специфічні області застосування. Попри схожість синтаксису, що походить із мови C, ці мови суттєво відрізняються за філософією розробки та принципами виконання програм, що робить кожну з них оптимальною для певних типів проектів.

C++:

C++ — це мультипарадигменна мова програмування, яка розширює можливості мови C та надає розробнику широкий контроль над системними ресурсами. Завдяки цьому вона є незамінною для виконання ресурсомістких і високопродуктивних завдань.

Ключові характеристики C++

Низькорівневий контроль: C++ забезпечує прямий доступ до пам'яті за допомогою покажчиків та їх управління, що дозволяє програмістам точно контролювати розподіл і звільнення ресурсів. Це сприяє досягненню максимальної продуктивності та ефективному використанню апаратного забезпечення.

Висока продуктивність: код C ++ складається безпосередньо в машинний код, який виконує центральний процесор. Можливий - C ++ - це швидка мова, оскільки вона не потребує віртуальної машини або колектора сміття. - Віртуальна машина та колектор сміття додають додаткової роботи мові, але C ++ вони не потребують.

Мультипарадигменність: Підтримує різні стилі програмування, зокрема об'єктно-орієнтоване (ООП) з класами, успадкуванням і поліморфізмом; процедурне; узагальнене (через шаблони); а також елементи функціонального програму

Складність: Завдяки низькорівневому доступу та широким можливостям, C++ характеризується складною кривою навчання і вважається однією з найважчих мов для повного освоєння. Необережне ручне управління пам’яттю може спричинити витоки пам’яті та інші помилки.

Платформна залежність: Код, скомпільований на C++, прив’язаний до певної операційної системи та архітектури процесора. Щоб програма працювала на іншій платформі, її необхідно повторно скомпілювати.

Багата стандартна бібліотека (STL): Містить ефективні контейнери (такі як вектори, списки, мапи), алгоритми (наприклад, сортування та пошук) і корисні утиліти, що суттєво полегшують процес розробки.

Сфери Застосування C++:

Ігрова розробка: Використовується для створення ігрових рушіїв (наприклад, Unreal Engine) та розробки високопродуктивних ігор із складною графікою та логікою.

Системне програмування: Застосовується для розробки операційних систем, драйверів пристроїв, файлових систем та іншого низькорівневого програмного забезпечення.

Високопродуктивні обчислення (HPC): Використовується для наукових симуляцій, фінансових розрахунків і реалізації бібліотек машинного навчання, де критично важлива висока швидкість обробки даних.

Вбудовані системи та IoT: Застосовується у програмуванні мікроконтролерів та пристроїв з обмеженими обчислювальними ресурсами, що характерно для Інтернету речей.

Високонавантажені сервери та бази даних: Використовується для розробки компонентів систем керування базами даних і веб-серверів, де критичною є максимальна продуктивність та швидкодія.

C#:

C# — сучасна об’єктно-орієнтована мова програмування, розроблена компанією Microsoft у межах платформи .NET. Вона створена для забезпечення ефективної, безпечної та масштабованої розробки програм, функціонуючи в керованому середовищі виконання.

Ключові Характеристики C#:

Програми C# виконуються у середовищі Common Language Runtime (CLR), що є складовою .NET. CLR забезпечує автоматичне збирання сміття (Garbage Collection), що полегшує керування пам’яттю та мінімізує ймовірність її витоків.

Об'єктно-орієнтована: Повністю реалізує основні принципи об’єктно-орієнтованого програмування — інкапсуляцію, успадкування, поліморфізм і абстракцію. Розроблена як сучасна мова з повною підтримкою ООП-парадигми.

Продуктивність розробки: C# забезпечує високу швидкість і зручність розробки завдяки розвиненій стандартній бібліотеці .NET, широкому набору вбудованих абстракцій (таких як LINQ, async/await, властивості та події), а також потужним інструментам, зокрема Visual Studio.

Висока продуктивність виконання: Завдяки JIT-компіляції (Just-In-Time) проміжний код C# (CIL/MSIL) перетворюється на машинний безпосередньо під час виконання, що дозволяє застосовувати динамічні оптимізації. Сучасна платформа .NET забезпечує рівень продуктивності, який у багатьох випадках наближається до C++.

Крос-платформність: Завдяки розвитку .NET Core (а нині — .NET 5 і новіших версій), C# став повноцінною мовою для крос-платформної розробки, що дає змогу створювати та запускати застосунки на Windows, Linux і macOS.

Безпека та стабільність: Кероване середовище виконання разом із суворою типовою системою C# забезпечують автоматичну перевірку меж масивів і обмежують прямий доступ до пам’яті (окрім спеціальних блоків unsafe), що підвищує надійність і безпеку програм.

Розвинена екосистема: C# має доступ до широкого спектра бібліотек і фреймворків, таких як ASP.NET Core, MAUI, ML.NET, а також підтримується активною спільнотою розробників, що сприяє швидкому розвитку та підтримці проєктів.

Сфери Застосування C#:

Веб-розробка: Використовується для розробки продуктивних веб-застосунків і API за допомогою фреймворку ASP.NET Core.

Хмарні сервіси: Застосовується для створення мікросервісів, серверної логіки мобільних застосунків, Azure Functions та інших рішень, орієнтованих на хмарну інфраструктуру.

Настільні додатки: Використовується для створення застосунків під Windows за допомогою WPF та WinForms, а також крос-платформних рішень із використанням MAUI або Avalonia.

Ігрова розробка: Є основною мовою програмування для створення ігор на базі ігрового рушія Unity.

Аналіз даних та машинне навчання: Використовується з бібліотекою ML.NET та інтегрується з іншими платформами для розробки рішень у сфері машинного навчання та обробки даних.

3 ПОРІВНЯЛЬНИЙ АНАЛІЗ C# І C++

3.1 Подібності та відмінності

Спочатку розглянемо подібності між C# і C++.

Хоча C# і C++ відрізняються за призначенням і архітектурою, вони мають багато спільного через спільне походження від мови C. Знання цих подібностей допомагає розробникам легше переходити між мовами або ефективніше застосовувати досвід, отриманий в одній, для розуміння іншої.

Обидві підтримують об’єктно-орієнтований підхід, включаючи такі ключові поняття, як класи, наслідування, поліморфізм та інкапсуляція. Крім того, і C#, і C++ дозволяють працювати з типами даних низького рівня, такими як цілі числа, символи та вказівники (хоча в C# робота з вказівниками обмежена та потребує небезпечного коду). Обидві мови забезпечують високу продуктивність і контроль над пам’яттю (у C++ — за допомогою ручного керування, у C# — через збирач сміття, але з можливістю оптимізацій). Також обидві широко використовуються для розробки складних програмних систем, включаючи настільні застосунки, ігри та серверні рішення. Ці подібності полегшують розуміння однієї мови, які вже знайомі з іншою.

1. Синтаксичні подібності

Обидві мови мають подібну синтаксичну структуру: умовні конструкції (if, switch), цикли (for, while, do-while), оператори та базові типи даних оформлені майже однаково. Завдяки цьому код на C# і C++ виглядає схожим і легко сприймається розробниками, знайомими з однією з мов.

Основні синтаксичні подібності:

1	Використання Фігурних Дужок {} для Блоків Коду:	Блоки коду — включаючи тіла функцій і методів, циклів, умовних операторів, а також визначення класів і структур — оформлюються за

		допомогою фігурних дужок, що забезпечує структурованість та зрозумілу організацію коду.
2	Завершення Інструкцій Крапкою з Комою ;:	В обох мовах кожна окрема інструкція (оператор) зазвичай закінчується крапкою з комою, що є обов’язковою частиною синтаксису для правильного розділення команд у коді.
3	Схожі Оператори:	Більшість арифметичних, логічних, операторів порівняння та присвоєння в C# і C++ або повністю збігаються, або мають дуже подібне використання, що спрощує перехід між цими мовами. Арифметичні: +, -, *, /, % Присвоєння: =, +=, -=, *= Порівняння: ==, !=, <, >, <=, >= Логічні: && (І), || (АБО), ! (НЕ) Інкремент/Декремент: ++, -- Порозрядні: &, |, ^, ~, <<, >>
4	Схожі Керуючі Конструкції:	Синтаксис умовних конструкцій і циклів майже ідентичний, що робить їх легкими для розуміння розробниками, знайомими з однією з мов. Умовні оператори: if, else if, else, switch. Цикли: for, while, do-while.

		Оператори переходу: break, continue, return.
5	Коментарі:	Використовують однаковий синтаксис для створення коментарів у коді: однорядкові коментарі позначаються за допомогою //, а багаторядкові — обмежуються між /* та */.
6	Оголошення Змінних та Типів Даних (базові):	Синтаксис оголошення змінних і базових типів даних, таких як цілі числа, числа з плаваючою комою та булеві значення, майже однакові, що спрощує розуміння та написання коду між цими мовами.
7	Визначення Класів та Членів Класу:	Оголошення класів, їхніх полів (змінних-членів) і методів (функцій-членів) має подібну структуру, що забезпечує схожий підхід до об’єктно-орієнтованого програмування в обох мовах. Обидві мови підтримують модифікатори доступу — public, private, protected, — що регулюють рівень доступу до членів класу. Хоча загальні принципи однакові, існують деякі відмінності у використанні, наприклад, у C++ модифікатор private встановлюється за замовчуванням для класів.

Цикли (for, while, do-while)

Цикли — це базові керуючі конструкції в програмуванні, які дають змогу багаторазово виконувати певний блок коду. Вони широко використовуються для автоматизації повторюваних дій, обробки масивів і колекцій, реалізації ітераційних обчислень та вирішення різноманітних типових завдань.

Виділяють три основні типи циклів, кожен з яких найкраще підходить для певних ситуацій використання:

- Цикл for — цикл із лічильником, зручний для ситуацій, коли відома кількість ітерацій заздалегідь.
- Цикл while — цикл з передумовою, що виконується доти, поки умова істинна, ідеально підходить для невизначеної кількості повторень.
- Цикл do-while — цикл з післяумовою, який гарантує хоча б одне виконання тіла циклу, оскільки умова перевіряється після його виконання.

Цикл for найкраще використовувати тоді, коли кількість повторень відома наперед або її можна визначити. Його заголовок складається з трьох основних елементів: ініціалізації лічильника, умови продовження циклу та оновлення лічильника після кожної ітерації.

Цикл while повторює виконання блоку коду доти, доки задана умова залишається істинною. Оскільки перевірка умови відбувається перед кожною ітерацією, цикл може жодного разу не виконати тіло, якщо умова з самого початку є хибною.

Цикл do-while подібний до циклу while, але з тією різницею, що умова перевіряється після виконання тіла циклу. Завдяки цьому тіло циклу гарантовано виконується хоча б один раз, незалежно від того, чи є умова істинною з самого початку.

2. Підтримка об’єктно-орієнтованого програмування (ООП)

C# і C++ — об’єктно-орієнтовані мови програмування, які реалізують базові принципи ООП: інкапсуляцію, успадкування, поліморфізм і абстракцію. При

цьому кожна з них пропонує власні особливості та механізми для втілення цих концепцій.

1	Інкапсуляція (Encapsulation)	Механізм об'єднання даних (полів) та методів (функцій), що працюють з цими даними, в єдину сутність — клас. Інкапсуляція також передбачає приховування внутрішньої реалізації об'єкта від зовнішнього світу, надаючи лише чітко визначений інтерфейс для взаємодії.
2	Успадкування (Inheritance)	Механізм, який дає змогу похідному (нащадку) класу успадковувати властивості та методи базового (батьківського) класу. Це сприяє повторному використанню коду та побудові об’єктно-орієнтованих ієрархій.
3	Поліморфізм (Polymorphism)	Здатність об’єктів різних класів по-різному обробляти один і той самий виклик методу залежно від свого конкретного типу. Це дозволяє використовувати спільний базовий інтерфейс для взаємодії з різними об’єктами, забезпечуючи гнучкість і розширюваність коду
4	Абстракція (Abstraction)	Механізм, що приховує внутрішню реалізацію об’єкта та показує лише його важливі властивості й функції. Це дозволяє розробнику зосередитися на тому, що робить об’єкт, замість того, як він це реалізує.

У C# і C++ класи можуть включати такі елементи, як поля, методи, конструктори та властивості, що дозволяє повноцінно моделювати об’єкти з даними та поведінкою.

3. Підтримка перевантаження

Перевантаження (Overloading) — це ключова концепція об'єктно-орієнтованого програмування, яка дає змогу створювати кілька методів, функцій або конструкторів з однаковим іменем, але різними параметрами в межах одного класу чи області видимості. Такий підхід покращує читабельність коду та полегшує його використання, дозволяючи реалізовувати схожі дії під спільним ім’ям, залежно від типу чи кількості аргументів.

Принцип перевантаження:

Компілятор розрізняє перевантажені методи за їхньою сигнатурою, яка включає:

- назву методу,
- кількість параметрів,
- типи параметрів,
- порядок параметрів.

Назва методу може бути однаковою, але комбінація зазначених характеристик має бути унікальною для кожної версії методу.

C# забезпечує повноцінну та зручну підтримку перевантаження методів, конструкторів і операторів. Це є звичайною практикою при розробці на C# і широко застосовується в базовій бібліотеці .NET (BCL).

C++ також повністю підтримує перевантаження функцій, методів, конструкторів і операторів. Ця можливість є однією з важливих рис, що робить C++ потужним інструментом для створення гнучких і розширюваних API.

Подібності та відмінності в реалізації перевантаження

Спільні риси:

- Обидві мови розрізняють перевантажені функції за сигнатурою методу, яка включає ім'я, кількість, типи та порядок параметрів.
- Тип повертаного значення не враховується у сигнатурі для перевантаження в обох мовах.
- І C#, і C++ підтримують перевантаження методів/функцій, конструкторів та операторів.
- Перевантаження сприяє створенню більш чистого та зрозумілого коду, оскільки відпадає необхідність вигадувати унікальні назви для схожих функцій.

Відмінності (незначні, а не принципові):

- Синтаксис перевантаження операторів відрізняється: у C# використовується ключове слово static operator, тоді як у C++ оператори можуть бути методами класу або окремими функціями.
- У C++ можливе перевантаження функцій-членів залежно від модифікатора const (наприклад, void print() const; і void print();), що не має прямого аналогу у C#.

4. Парадигма статичної типізації

Статична типізація — це підхід до перевірки типів, при якому сумісність типів змінних і виразів визначається під час компіляції, а не під час виконання програми. Це означає, що тип кожної змінної має бути відомий або виведений компілятором до запуску програми.

Основні принципи статичної типізації:

- Визначення типів під час компіляції

У мовах із статичною типізацією тип змінної має бути заданий під час її оголошення. Це дозволяє компілятору заздалегідь знати, які саме дані буде зберігати змінна ще до початку виконання програми. Наприклад: int, double, string, bool.

- Перевірка типів компілятором

Компілятор перевіряє, чи використовуються змінні відповідно до їхніх типів. Якщо спробувати виконати операцію, непридатну для конкретного типу (наприклад, додати число до рядка без явного перетворення), буде згенеровано помилку ще до запуску програми.

- Незмінність типу

Після того як змінну оголошено з певним типом, цей тип не може змінитися протягом усього часу її існування. Наприклад, змінна int age завжди зберігатиме лише цілі числа — її значення можна оновлювати, але тип залишатиметься незмінним. Типові помилки виявляються на етапі компіляції, що сприяє більшій надійності та безпеці коду.

Переваги статичної типізації:

- Раннє виявлення помилок

Одна з головних переваг. Помилки, пов’язані з неправильним використанням типів, виявляються вже на етапі компіляції, ще до запуску програми. Це зменшує ризик помилок у виконанні (runtime errors) і спрощує процес відлагодження. Ви виявляєте проблему до того, як код потрапить до кінцевого користувача.

- Підвищена надійність коду

Завдяки перевірці типів компілятором, імовірність виникнення помилок, пов’язаних із типами, значно знижується. Це підвищує стабільність і безпечність роботи програми.

- Краща продуктивність

Оскільки типи відомі заздалегідь, компілятор може генерувати оптимізований машинний код. Немає необхідності виконувати динамічну перевірку типів під час виконання, що знижує накладні витрати і підвищує швидкість виконання програми.

- Покращена підтримка в IDE

Сучасні середовища розробки (наприклад, Visual Studio, Rider, VS Code) використовують інформацію про типи для:

  - автодоповнення коду (IntelliSense)
  - підсвічування помилок
  - переходу до визначень
  - зручного рефакторингу
- Краща зрозумілість коду

Явне визначення типів робить код самодокументованим. Достатньо поглянути на сигнатуру методу чи оголошення змінної, щоб зрозуміти, які саме дані обробляються. Це полегшує читання, тестування й супровід програмного коду.

Недоліки статичної типізації:

- Більш багатослівний код

Необхідність явно вказувати типи змінних, параметрів і повертаємих значень може зробити код об'ємнішим, особливо в простих сценаріях.

- Менша гнучкість

Робота з динамічно структурованими даними або операціями, де тип визначається під час виконання, вимагає:

  - явного приведення типів

  - використання узагальнень (generics)
  - застосування рефлексії
- Вища складність для новачків

Початківцям часто важко одразу зрозуміти концепції типів, сигнатур, помилок компіляції, приведення типів тощо. Це може спричиняти фрустрацію та сповільнювати навчання на старті.

5. Можливість роботи з покажчиками

Покажчики — це одна з основних концепцій у низькорівневому програмуванні, яка дозволяє працювати безпосередньо з адресами оперативної пам’яті. На відміну від звичайних змінних, що зберігають значення, покажчик зберігає адресу іншої змінної або області пам’яті. Це забезпечує розробнику гнучкий і точний контроль над роботою з пам’яттю, що особливо важливо при оптимізації продуктивності та створенні складних алгоритмів.

Використання покажчиків відкриває широкі можливості, проте потребує обережності. Неправильне поводження з ними може спричинити серйозні помилки, серед яких: витоки пам’яті, аварійне завершення роботи програми (наприклад, через спробу звернення до неіснуючої адреси), або створення вразливостей у системі безпеки.

Призначення та переваги покажчиків

- Безпосередній доступ до пам’яті: Покажчики надають можливість працювати безпосередньо з адресами пам’яті, що є характерною рисою низькорівневого програмування.
- Ефективне управління ресурсами: Завдяки точному контролю над пам’яттю, покажчики широко застосовуються в системному програмуванні, розробці драйверів, компіляторів, а також у високопродуктивних застосунках, таких як відеоігри.

- Передача об'єктів у функції без копіювання: Використання покажчиків дозволяє передавати великі об’єкти у функції за адресою, уникаючи накладних витрат на копіювання.
- Побудова динамічних структур даних: Покажчики є основою для реалізації таких динамічних структур, як зв’язані списки, дерева, графи, що змінюються під час виконання програми.
- Оптимізація продуктивності: Завдяки роботі на рівні адрес, зменшується використання додаткових ресурсів, що дозволяє суттєво підвищити швидкодію програм при роботі з великими обсягами даних.

Ризики використання покажчиків

Хоча покажчики забезпечують потужні можливості низькорівневого керування пам’яттю, їхнє неправильне використання може призвести до серйозних проблем:

- Витоки пам’яті

Виникають, коли пам’ять виділяється за допомогою оператора new або malloc, але не звільняється за допомогою delete або free. Це призводить до поступового зростання використаної пам’яті, що врешті може виснажити ресурси системи.

- Сегментаційні помилки

Виникають, коли програма намагається звернутися до області пам’яті, до якої не має доступу (наприклад, за нульовим або невірним покажчиком). Це призводить до аварійного завершення виконання.

- "Висячі" покажчики

Виникають, коли покажчик зберігає адресу пам’яті, яка вже була звільнена. Подальше використання такого покажчика може призвести до непередбачуваної поведінки або збоїв.

- Безпекові вразливості

Неправильне керування покажчиками відкриває шлях до атак типу buffer overflow, use-after-free, pointer arithmetic exploitation тощо. Такі вразливості активно використовуються для написання шкідливого ПЗ або обходу систем захисту.

У C++ покажчики є базовим інструментом, а в C# — використовуються лише у unsafe-блоках через обмеження, накладені керованим середовищем, яке забезпечує контроль за доступом до пам’яті.

6. Підтримка інтерфейсів та абстрактних класів

Інтерфейси та абстрактні класи є ключовими елементами об'єктно-орієнтованого програмування, які забезпечують абстрагування та повторне використання коду. Вони дозволяють визначати загальні контракти або часткову реалізацію функціональності, яку надалі конкретизують похідні класи.

Попри спільну мету — створення основи для інших класів, — інтерфейси та абстрактні класи відрізняються способом реалізації та сферою застосування. І C#, і C++ підтримують ці механізми, хоча реалізація і використання мають низку синтаксичних і концептуальних відмінностей.

У C# інтерфейси та абстрактні класи реалізовані як окремі конструкції мови з чітко визначеним синтаксисом, що забезпечує просте та безпечне використання. Розробникам легше розрізняти їхню роль та застосування.

У C++ концепції інтерфейсів і абстрактних класів реалізуються через успадкування та оголошення чисто віртуальних функцій. Такий підхід є більш гнучким, проте потребує глибшого розуміння принципів роботи мови, особливо в контексті множинного успадкування, що може ускладнити розробку.

7. Можливість створення узагальненого коду

Узагальнене програмування (Generics у C# та Templates у C++) — це ефективний підхід, що дозволяє створювати універсальний код, здатний працювати з різними типами даних без фіксації конкретного типу заздалегідь. Це реалізується шляхом використання параметризованих типів, які компілятор замінює під час компіляції.

Основна мета узагальненого програмування полягає в забезпеченні універсальності, безпеки та ефективності коду.

- Мінімізація дублювання: Один універсальний алгоритм або структура даних (наприклад, сортування, стек, список) може використовуватися для роботи з різними типами, без необхідності створення окремих версій.
- Перевірка типів під час компіляції: Узагальнення дозволяє виявляти помилки, пов’язані з типами, ще до запуску програми, що підвищує надійність і стабільність коду.
- Повторне використання логіки: Код, написаний один раз, можна легко застосовувати для широкого кола типів, що відповідають встановленим вимогам — без втрати типізації.
- І C#, і C++ підтримують абстракцію через інтерфейси та абстрактні класи.
- В C# існує офіційне поняття інтерфейсу, а в C++ — його імітація через віртуальні методи.
- В обох мовах абстрактні класи дозволяють змішувати реалізацію та абстракцію.

8. Підтримка обробки виключень

Обробка виключень — це спосіб виявляти та керувати помилками, що виникають під час виконання програми, дозволяючи уникнути її несподіваного завершення.

Ключові поняття механізму обробки винятків:

- Виняток (Exception): Це спеціальний об'єкт, який сигналізує про виникнення помилки або нестандартної ситуації під час виконання програми.
- Генерація винятку (Throw/Raise): Коли програма виявляє помилку, вона створює виняток і "підкидає" його для обробки.
- Обробка винятку (Catch/Handle): Це реакція програми на виняток — виконання спеціального коду, що дозволяє впоратися з помилкою або мінімізувати її наслідки.
- Стек викликів (Call Stack): Якщо виняток не обробляється на місці, система проходить по стеку викликів, шукаючи відповідний блок обробки. Якщо його не знайдено — програма завершується аварійно.

Обидві мови пропонують потужні й гнучкі механізми для обробки винятків, які є ключовими для створення надійного та стабільного програмного забезпечення.

- У C# обробка винятків інтегрована з керованим середовищем виконання, а конструкції на кшталт finally та using полегшують керування ресурсами і підвищують безпеку застосунків.
- У C++ також підтримується обробка винятків, однак для гарантування безпеки ресурсів широко використовується ідіома RAII, що вимагає глибшого розуміння життєвого циклу об’єктів і управління пам’яттю.

А тепер розглянемо відмінності між C# і C++.

1. Типова платформа

Типова платформа — це набір технологій або середовище, з яким певна мова програмування традиційно або переважно асоціюється. Хоча мова може застосовуватися і поза межами цієї платформи, саме вона є основною або найпоширенішою сферою використання цієї мови.

Типовою платформою для C# є Microsoft .NET (раніше .NET Framework / .NET Core).

Це мова високого рівня, створена для платформи .NET. Програми компілюються в проміжний код (IL — Intermediate Language), який виконується у керованому середовищі CLR (Common Language Runtime). CLR забезпечує безпеку, автоматичне керування пам’яттю та інші важливі сервіси.

Основною платформою для C# є .NET Framework, який здебільшого орієнтований на Windows, а також сучасні кросплатформові версії — .NET Core та .NET 5+ — що підтримують Windows, Linux і macOS.

Мова використовується переважно для розробки бізнес-додатків, веб-сервісів і десктопних програм.

Типова платформа для C# спочатку була орієнтована виключно на Windows, але з часом розширилася до широкого спектра систем завдяки екосистемі .NET, з особливим фокусом на веб-додатки, хмарні сервіси та розробку ігор.

Типова платформа для C++ — це практично будь-яка система, де потрібен низькорівневий контроль, висока продуктивність і ефективне управління ресурсами. Завдяки компіляції безпосередньо у машинний код під конкретну архітектуру, C++ є однією з найбільш універсальних мов з точки зору сумісності з різними платформами. Вона є компільованою мовою системного рівня.

Програми на цій мові транслюються у машинний код, який безпосередньо працює на конкретній операційній системі та апаратній архітектурі (наприклад, Windows, Linux, macOS). Вони виконуються напряму на процесорі без потреби в додатковому середовищі виконання. Такий підхід робить мову ідеальною для розробки системного програмного забезпечення, драйверів, ігор та вбудованих систем.

2. Управління пам’яттю

Управління пам’яттю є ключовою та базовою різницею між C# і C++. Саме ця відмінність впливає на безпеку, продуктивність, гнучкість і рівень складності розробки в обох мовах.

В C# керування пам’яттю відбувається автоматично завдяки збирачу сміття (Garbage Collector). Розробнику не потрібно самостійно виділяти або звільняти пам’ять — система сама визначає, які об’єкти більше не використовуються, і звільняє відповідні ресурси. Такий метод підвищує безпеку та спрощує розробку, проте іноді може викликати непередбачені паузи під час виконання програми через роботу збирача сміття.

В C++ керування пам’яттю здійснюється вручну і повністю контролюється програмістом. Пам’ять виділяється за допомогою оператора new і звільняється через delete. Такий підхід дає максимальний контроль і дозволяє досягти високої ефективності, але вимагає обережності, щоб уникнути помилок, таких як витоки пам’яті, подвійне звільнення або використання звільненої пам’яті (dangling pointers). Для спрощення управління ресурсами часто використовують ідіому RAII (Resource Acquisition Is Initialization) та смарт-покажчики, які автоматизують процес звільнення пам’яті.

3. Парадигми програмування

Парадигма програмування — це спосіб мислення або концептуальний підхід до розробки програм. Вона визначає принципи організації коду та методи вирішення завдань. Різні мови програмування підтримують одну або кілька парадигм, що впливає на стиль написання програм і структуру коду.

І C#, і C++ належать до мультипарадигмальних мов програмування, тобто підтримують кілька стилів написання коду. Водночас кожна з них акцентує увагу на різних парадигмах і реалізує їх з різним ступенем інтеграції та зручності для розробника.

C# вирізняється глибокою інтеграцією об’єктно-орієнтованого підходу, а також активною підтримкою функціонального програмування (зокрема,

через LINQ) та узагальненого програмування (Generics). Це робить мову надзвичайно ефективною для створення додатків у керованому середовищі .NET.

C++ поєднує об’єктно-орієнтоване програмування з потужними механізмами шаблонів (Templates) та підтримкою низькорівневого імперативного стилю. Така гнучкість забезпечує високий рівень продуктивності та детального контролю над ресурсами, що особливо важливо для системного програмування та розробки продуктивних застосунків.

4. Синтаксис і структура

C++ забезпечує високу гнучкість і дозволяє програмісту безпосередньо керувати апаратними ресурсами, проте вимагає ретельного підходу та точності при роботі з пам’яттю та структурою програми. Натомість C# зосереджений на безпечній та впорядкованій розробці, пропонуючи лаконічний синтаксис і зручні механізми завдяки роботі в керованому середовищі.

5. Шаблони та узагальнення

Шаблони в C++ і узагальнення в C# — це засоби, що дозволяють створювати універсальний код, який працює з різними типами даних без жорсткої прив’язки до конкретного типу під час написання. Це сприяє багаторазовому використанню коду, покращує типову безпеку та зменшує потребу в дублюванні реалізацій.

Попри спільну мету, підходи до реалізації шаблонів у C++ та узагальнень у C# мають значні відмінності в синтаксисі, поведінці під час компіляції та можливостях.

Шаблони в C++ забезпечують високий рівень гнучкості та контроль під час компіляції, що дає змогу створювати складні, ефективні та оптимізовані рішення. Проте це супроводжується збільшенням складності коду, можливими проблемами з розміром згенерованого коду та менш зрозумілими

повідомленнями про помилки. Такий підхід ідеально підходить для розробки системного ПЗ, ігрових рушіїв і продуктивних бібліотек.

Узагальнення в C# орієнтовані на зручність розробки, безпечну роботу з типами та інтеграцію з середовищем CLR. Вони створені для підвищення ефективності програміста та спрощення підтримки коду. Завдяки цьому Generics є ідеальним вибором для бізнес-застосунків, веб-сервісів і крос-платформних проєктів, де важливо досягти оптимального балансу між продуктивністю та простотою.

6. Обробка виключень

Обробка винятків є важливим інструментом у програмуванні, що дозволяє програмі впоратися з помилками або нестандартними ситуаціями під час виконання без аварійного завершення. Завдяки цьому механізму програма може вчасно виявити проблему, обробити її відповідним чином, відновити нормальну роботу або завершити виконання коректно й передбачувано.

C# завдяки керованому середовищу виконання, а також конструкціям finally та using, робить обробку винятків і керування ресурсами зручнішим та безпечнішим для розробників.

Натомість C++ використовує підхід RAII (Resource Acquisition Is Initialization), який вимагає глибшого розуміння життєвого циклу об’єктів, але забезпечує високий рівень контролю та може забезпечити кращу продуктивність.

7. Платформа та сумісність

У програмуванні "платформа" означає середовище, в якому виконується програма. Вона охоплює операційну систему, апаратне забезпечення та програмні компоненти на зразок фреймворків або віртуальних машин. Платформна сумісність відображає, наскільки легко програму, створену під одну платформу, можна запустити на іншій.

Мови C# і C++ демонструють суттєво різні підходи до роботи з платформами та сумісністю, що обумовлено їхньою архітектурою та середовищами виконання.

C# — це відмінне рішення для швидкої та ефективної розробки крос-платформних застосунків у межах екосистеми .NET. Його сильні сторони — висока продуктивність розробника, автоматичне управління пам’яттю та зручність, що робить мову ідеальною для створення вебсервісів, хмарних рішень, десктопних і мобільних додатків.

C++ залишається незамінним там, де критично важливі повний контроль над апаратним забезпеченням, висока продуктивність та доступ до низькорівневих ресурсів. Саме тому він широко використовується в системному програмуванні, розробці ігрових рушіїв та продуктивних додатків, де потрібна точна оптимізація кожного ресурсу, навіть якщо це вимагає додаткових зусиль для забезпечення крос-платформності.

8. Інструменти розробки

Інструментальна екосистема C# є більш цілісною та інтегрованою, що забезпечує високу продуктивність розробника, особливо в середовищі Visual Studio. Це дозволяє зосередитися на кодуванні, а не на конфігурації.

C# тісно інтегрований із платформою .NET і пропонує зручне середовище для розробки "з коробки", особливо у Visual Studio, що значно спрощує старт і щоденну роботу розробника.

Натомість, екосистема C++ є більш модульною та вимагає більшого ручного контролю. Це дає розробникам величезну гнучкість у виборі найкращих інструментів для конкретного завдання та платформи, але також вимагає глибших знань про їхню взаємодію.

C++ вимагає гнучкого налаштування середовища розробки та часто потребує використання різних інструментів і бібліотек залежно від цільової

платформи. Це надає велику свободу, але водночас вимагає більше технічних знань.

9. Вказівники і небезпечний код

Вказівники (Pointers) — це спеціальні змінні, що містять адреси інших змінних у оперативній пам’яті. Вони відкривають можливість прямого керування пам’яттю, що робить їх надзвичайно потужним інструментом. Водночас, неправильне використання вказівників може спричинити серйозні помилки в програмі.

C# робить акцент на безпеку та ефективність розробки, автоматично керуючи пам’яттю через збирач сміття та використовуючи посилання замість сирих вказівників. Можливість написання "небезпечного" коду доступна лише як виняток для тих випадків, коли потрібно працювати з низькорівневими операціями або добиватися максимальної продуктивності, причому такі ділянки коду явно позначаються як потенційно ризикові.

C++ надає розробнику повний і необмежений контроль над пам’яттю через використання вказівників. Це дозволяє досягати максимальної продуктивності та працювати з найнижчими рівнями системи, але водночас вимагає великої обережності та глибоких знань, адже помилки в управлінні пам’яттю можуть призвести до серйозних наслідків.

10. Вбудовані типи та колекції

Вбудовані типи даних (примітивні типи) та колекції є базовими елементами будь-якої програми, оскільки вони визначають спосіб збереження та організації даних. Хоча концепції в C# та C++ подібні, їх реалізація і застосування відрізняються, що відображає різні підходи та філософії цих мов.

Вбудовані типи — це основні типи даних, які надаються безпосередньо мовою програмування (або її стандартною бібліотекою) і слугують для представлення простих значень, таких як числові, символьні та булеві дані.

Колекції — це типи структур даних, що використовуються для зберігання, впорядкування та обробки наборів елементів як єдиного цілого.

Особливості:

C#

- Колекції входять до складу .NET Base Class Library і доступні "з коробки" через простий та уніфікований інтерфейс.
- Використовують узагальнення (generics) для забезпечення типобезпечності та повторного використання коду.
- Тип string є посилальним типом, який представляє не масив символів, а об'єкт, що інкапсулює текстові дані з підтримкою функцій.
- Пам’яттю керує CLR автоматично за допомогою збирача сміття (Garbage Collector), що зменшує ризик витоків і полегшує розробку.

C++

- Колекції реалізовані за допомогою STL (Standard Template Library) — стандартної бібліотеки шаблонів.
- Завдяки шаблонам, колекції забезпечують типобезпечну роботу з будь-якими типами даних.
- Для використання колекцій необхідно явно підключати відповідні заголовкові файли (наприклад, #include <vector>, #include <map>).
- Керування пам’яттю здійснюється вручну: розробник відповідає за алокацію, звільнення та уникнення витоків пам’яті.

3.2 Переваги і недоліки

C# та C++ — це дві потужні мови програмування, кожна з яких має свої унікальні переваги та сфери застосування. Їх вибір зазвичай обумовлюється специфікою проекту, технічними вимогами, середовищем розгортання, продуктивністю команди та стратегічними цілями розробки. Залежно від

контексту, одна з мов може забезпечити вищу ефективність, кращу підтримку або більшу гнучкість.

Переваги та Недоліки C#

Переваги C#:

- Автоматичне управління пам’яттю — використання збирача сміття (Garbage Collector) для автоматичного звільнення невикористовуваних об’єктів.
- Сучасний та зрозумілий синтаксис — сприяє легкому написанню та читанню коду.
- Багата стандартна бібліотека — включає колекції, LINQ для зручної роботи з даними, підтримку багатопотоковості та інші інструменти.
- Тісна інтеграція з платформою .NET — ідеально підходить для створення веб-додатків, десктопних програм, мобільних та хмарних сервісів.
- Потужне середовище розробки — такі інструменти як Visual Studio, Rider та підтримка IntelliSense підвищують продуктивність розробника.

Недоліки C#:

- Обмежений контроль над пам’яттю — неможливість прямого доступу до низькорівневих механізмів без переходу в unsafe-блоки.
- Менша швидкість виконання порівняно з C++ — через проміжний код (IL) і кероване середовище CLR.
- Залежність від платформи .NET — не найкращий вибір для системного програмування чи розробки драйверів.
- Вимагає додаткового середовища виконання — таких як CLR, Mono чи .NET Runtime.
- Безпечний код обмежений — для роботи з небезпечним (unsafe) кодом потрібні спеціальні дозволи і додаткова увага.

Переваги та Недоліки C++

Переваги C++:

- Висока продуктивність — програми компілюються безпосередньо у машинний код, що виконується без проміжних віртуальних машин.
- Повний контроль над пам’яттю — ручне управління за допомогою вказівників і явної алокації/деалокації.
- Ідеальний для системного програмування — використовується для розробки драйверів, операційних систем та вбудованих систем.
- Потужні шаблони (templates) — підтримка складного метапрограмування та створення гнучких, типобезпечних структур.
- Висока кросплатформеність — доступний на різних операційних системах з великою кількістю компіляторів.

Недоліки C++:

- Складне керування пам’яттю — ризик виникнення витоків і помилок при звільненні ресурсів.
- Триваліший процес розробки — потрібно писати більше коду і використовувати менше високорівневих інструментів.
- Складність відлагодження — помилки з вказівниками і сегментаційні збої можуть бути важкими для виявлення.
- Відсутність автоматичного збору сміття — відповідальність за звільнення пам’яті лежить на розробнику.
- Високий поріг входження — новачкам складно опанувати всі аспекти мови і уникати типових помилок.

4 СФЕРИ ЗАСТОСУВАННЯ МОВ

Вибір між C# та C++ для конкретного проєкту зазвичай визначається його унікальними вимогами щодо продуктивності, цільової платформи, наявних ресурсів та складності розробки. Кожна з мов має свою історично сформовану та логічну нішу, де її сильні сторони використовуються найефективніше.

Сфери Застосування C#

C# разом із екосистемою .NET (включно з .NET Framework та .NET Core) створені з основним фокусом на підвищення продуктивності розробника, забезпечення безпеки та підтримку кросплатформності, що робить їх відмінним вибором для розробки різноманітних сучасних застосунків.

- Корпоративні додатки: бізнес-системи, CRM, ERP рішення, які забезпечують управління бізнес-процесами.
- Веб-розробка: створення вебсайтів та веб-додатків за допомогою ASP.NET і сучасного фреймворку Blazor.
- Десктопні додатки: розробка програм для Windows із використанням технологій WinForms, WPF, UWP.
- Мобільні додатки: кросплатформна розробка для iOS та Android через Xamarin та MAUI.
- Хмарні сервіси: побудова серверних застосунків і сервісів у хмарі на платформі Azure.
- Ігрова розробка: використання мови C# як основної у популярному ігровому рушії Unity.
- Автоматизація та скриптування: створення внутрішніх скриптів і тестових інструментів для підвищення ефективності розробки.

Сфери Застосування C++

C++ обирають тоді, коли необхідний абсолютний контроль над апаратним забезпеченням, найвища продуктивність і максимально ефективне використання ресурсів.

Системне програмування: розробка операційних систем, драйверів та прошивок.

Розробка ігор: використання у створенні відомих ігрових рушіїв, таких як Unreal Engine і CryEngine.

Вбудовані системи: програмування мікроконтролерів та пристроїв Інтернету речей (IoT).

Програмне забезпечення з високими вимогами до продуктивності: створення графічних редакторів, CAD-систем та виконання наукових обчислень.

Бібліотеки та фреймворки: розробка кросплатформних бібліотек, що інтегруються з іншими мовами програмування.

Фінансові додатки: системи для трейдингу та обробки даних у режимі реального часу.

ВИСНОВОК

C++ і C# — це високопродуктивні мови програмування, що походять від мови C, але суттєво відрізняються за своїм призначенням, підходами до управління пам’яттю та сферами використання.

C++ — високопродуктивна мова програмування системного рівня, яка забезпечує повний контроль над пам’яттю. Вона чудово підходить для розробки драйверів, операційних систем, ігрових рушіїв та вбудованих систем, де критично важливі швидкодія та оптимізація. Водночас вона вимагає від розробника глибоких знань, оскільки управління пам’яттю здійснюється вручну, а процес налагодження є складнішим.

C# — сучасна мова програмування з чіткою типізацією, орієнтована на створення прикладного програмного забезпечення в екосистемі .NET. Вона пропонує зручні засоби для швидкої розробки, автоматичне управління пам’яттю та багату стандартну бібліотеку для розв’язання широкого спектра задач — від бізнес-логіки до веб- і хмарних сервісів. Проте вона менш підходить для системного програмування або сценаріїв з жорсткими вимогами до продуктивності.

Зрештою, вибір між C++ та C# визначається завданнями проекту:

- C++ доцільний там, де критично важлива продуктивність і повний контроль над ресурсами.
- C# підходить краще для швидкої розробки, безпечного керування пам’яттю та створення масштабованих додатків.

СПИСОК КОРИСНИХ ДЖЕРЕЛ

1. Грицунов О. В. Інформаційні системи та технології: навчальний посібник, Харків (ХНАМГ) 2010 р. – 223 с.

2. Ушенко Ю.О., Ковальчук М.Л.,Гавриляк М.С., Негрич А.Л. Методологія інформаційних систем та баз даних: навчальний посібник, Чернівці 2021 р. – 240 с.

3. Страуструп Бьорн Мова програмування C++: книга

4. Олексій Васильєв Програмування на C# для початківців. Основні відомості: книга