Лекція 2

ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ МОВОЮ C++

Лекція 2. Основи об'єктно-орієнтованого програмування мовою C++

План

1. Введення у мову програмування C++.
2. Основи об’єктно-орієнтованого програмування. Класи та об'єкти.
3. Інкапсуляція. Успадкування. Поліморфізм.
4. Перевантаження операцій.
5. Системи програмування мовою C++.

1. Введення в мову програмування C++

C++ – високорівнева компільована мова програмування загального призначення зі статичною типізацією. На сьогоднішній день C++ є однією з найпопулярніших і найпоширеніших мов, тому що вона підходить для створення найрізноманітніших програм (операційних систем, драйверів пристроїв, серверів, компіляторів, комп'ютерних ігор і т. д.).

Б'ярн Страуструп

Підтримувані парадигми програмування:

- процедурна;
- структурна;
- об'єктно-орієнтована;
- узагальнена.

Мова C++ була розроблена на початку 1980-х років співробітником фірми Bell Labs Б’ярном Страуструпом як об'єктно-орієнтоване розширення мови C. Перша її назва «C з класами».

1. Введення в мову програмування C++

Стандарти C++:

- **C++98** – перший промисловий стандарт, розроблений комітетом зі стандартизації C++ (ISO/IEC JTC1/SC22/WG21 working group) і затверджений у 1998 р.;
- **C++03** – другий стандарт, затверджений 2003 р. (є уточненням стандарту C++98);
- **C++11** – затверджений у 2011 р.; містить значні зміни, порівняно з попередніми стандартами;
- **C++14** – є уточненням стандарту C++11 (затверджено 2014 р.);
- **C++17** – містить низку поліпшень щодо підтримки паралелізму (2017 р.);
- **C++20** – останній чинний стандарт (2020 р.), містить низку поліпшень щодо C++17;

1. Введення в мову програмування C++

C++ має потужну стандартну бібліотеку (**C++ Standard Library**), що представляє собою колекцію класів і функцій, написаних на C++.

Вона містить в собі:

- стандартну бібліотеку мови C;
- роботу з рядками;
- ввід-вивід;
- багатопотоковість;
- стандартну бібліотеку шаблонів (**STL – Standard Template Library**);
- набір алгоритмічних операцій над контейнерами та іншими послідовностями;
- ...

1. Введення в мову програмування C++

Не входить у C++ Standard Library, але значно розширює функціонал мови C++ бібліотека **Boost**. Вона містить:

- різноманітні алгоритми;
- багатопотокове програмування;
- математичні та чисельні алгоритми;
- взаємодію з іншими мовами програмування;
- синтаксичний та лексичний розбір;
- узагальнене програмування;
- структури даних та багато іншого.

2. Основи об’єктно-орієнтованого програмування. Класи та об'єкти

Об’єктно-орієнтований підхід (ООП) до програмування дозволяє розробнику створювати власні типи даних (**класи**), що більш адекватно описують предметну область, ніж наявні у мові програмування вбудовані типи даних.

ООП базується на таких фундаментальних поняттях:

- **абстракція** (використання лише тих характеристик об'єкта, які з достатньою точністю представляють його в даній системі) – модель об'єкта предметної області, що формується у вигляді класу;
- **інкапсуляція** – розміщення в одному компоненті даних та методів, які з ними працюють та/або приховування внутрішньої реалізації від інших компонентів;
- **успадкування** – механізм, що дозволяє типу даних успадковувати дані та функціональність деякого наявного типу, сприяючи повторному використанню компонентів програмного забезпечення;
- **поліморфізм** – здатність функції обробляти дані різних типів.

Фактично **клас** – це абстракція, тобто створений програмістом новий тип даних, що описує сутності предметної області більш точно (наочно), ніж наявні в мові програмування вбудовані типи даних.

Об'єкт – це змінна класу, що фізично існує у пам'яті комп'ютера.

2. Основи об’єктно-орієнтованого програмування. Класи та об'єкти

У C++ клас описується так:

```
class <ім’я_класу>
{
[private:]
	// Закриті властивості та методи (доступні лише методам класу)
	// ...
[protected:]
	//Захищені властивості та методи
	// (доступні тільки методам класу та його нащадкам)
	// ...
[public:]
	// Інтерфейсна частина класу
	// (властивості та методи, оголошені тут, доступні у будь-якому місці програми)
	// ...
};
```

2. Основи об’єктно-орієнтованого програмування. Класи та об'єкти

Наприклад, клас, що описує автомобіль, може мати такий вигляд:

```
#include <string>

class Car
{
private:
	string model; // Модель автомобіля
	string color;   // Колір
	int year;        // Рік випуску

public:
	Car(string m, string c, int y)       // Конструктор класу
	{
		model = m;
		color = c;
		year = y;
	}
	~Car(void) {} // Пустий декструктор класу (оголошений, але нічого не робить)
};
```

2. Основи об’єктно-орієнтованого програмування. Класи та об'єкти

З кожним класом асоціюються два стандартні методи – **конструктор** та **деструктор**.

Конструктор автоматично викликається під час створення об'єкта для його початкової ініціалізації. Його ім'я збігається з ім'ям класу. Конструкторів може бути кілька. Вони повинні в цьому випадку відрізнятися за **сигнатурою** (кількістю та/або типом аргументів).

Наприклад, у класі Car можна оголосити ще один «порожній» конструктор такого вигляду:

```
class Car
{
//…
public:
	void Car(void)
	{
		year = 0;
	}
//...
};
```

2. Основи об’єктно-орієнтованого програмування. Класи та об'єкти

У наведених прикладах у класу Car оголошено два конструктори. Вони відрізняються один від одного кількістю аргументів. Наприклад.

```
// …

Car unknown,
      my_car("ZAZ", "green", 1972);


// …
```

У наведеному вище прикладі при створенні об'єкта (змінної) unknown буде автоматично викликаний порожній конструктор (без аргументів), а при створенні змінної my_car буде викликаний конструктор з трьома аргументами.

2. Основи об’єктно-орієнтованого програмування. Класи та об'єкти

У класу C++ може бути ще один спеціальний тип конструктора - **копіювальний**. Він викликається при початковій ініціалізації об'єкта на момент його оголошення. Наприклад.

```
// …
Car my_car("ZAZ", "green", 1972),
      my_car1 = my_car; // Тут викликається конструктор копіювання
// …
```

Конструктор копіювання оголошується, наприклад, так:

```
// Конструктор копіювання
Car(const Car &right)
{
      model = right.model;
      color=right.color;
      year = right.year;
}
```

Таким чином, аргументом конструктора копіювання є константне (незмінне) посилання на раніше створений об'єкт даного класу, значення якого копіюється в поточний новостворений об'єкт.

2. Основи об’єктно-орієнтованого програмування. Класи та об'єкти

Деструктор автоматично викликається тоді, коли об'єкт видаляється з пам'яті (знищується).

```
// ...

int main(void)
{
    Car my_car("Volvo", "white", 2020); ← Тут викликається конструктор

    // ...


    return 0; ← Тут при виході з функції автоматично викликається деструктор
}
```

Деструктор у класі C++ завжди один. Його ім'я має починатися з тильди «~», а далі воно збігається з ім'ям класу. Задачею деструктора є очищення об'єкту, наприклад, звільнення динамічно виділеної пам'яті, закриття раніше відкритих файлів тощо.

3. Інкапсуляція. Успадкування. Поліморфізм

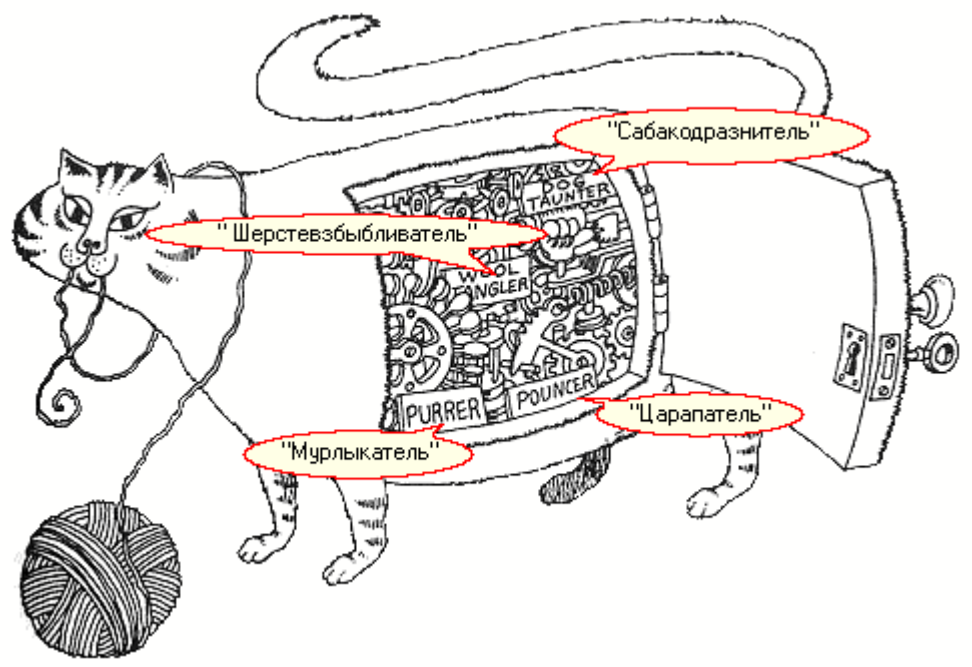
Механізм **інкапсуляції** як приховування даних реалізується шляхом використання модифікаторів **private** – **protected** – **public** при оголошенні класу.

Всі члени класу, описані або відразу після фігурної дужки, що відкривається, або після ключового слова **private**, є його закритою внутрішньою частиною. Доступ до них можливий лише з методів цього класу. У наведеному вище класі Car властивості (змінні) model, color та year змінюються лише у конструкторі. Спроба безпосередньо звернутися до них викликає помилку компіляції.

```
// …

Car my_car;


my_car.model = "ZAZ";  ←  Тут компілятор видасть повідомлення про помилку

// ...
```

3. Інкапсуляція. Успадкування. Поліморфізм

Зазвичай в секції private оголошуються властивості та методи класу, необхідні для його функціонування, але які приховуються від користувача як непотрібні йому деталі реалізації класу.



Для гри з кішкою зовсім не потрібно знати, як вона влаштована (і тим більше бачити як виглядають її нутрощі :)

3. Інкапсуляція. Успадкування. Поліморфізм

У секції **protected** оголошуються властивості та методи класу, доступні для його методів та методів класів-спадкоємців.

У секції **public** оголошуються члени класу, доступні з будь-якої частини програми. Це так звана інтерфейсна частина класу. Вона призначена для взаємодії із ним.

Наприклад, інтерфейсною частиною автомобіля є його кермо, педалі, важіль перемикання швидкостей та панель приладів. Деталі ж реалізації автомобіля, такі як його двигун, шасі, трансмісія, паливна система тощо зазвичай ховаються від водія (хоча і не так строго як в ООП).

Деталі реалізації, які зазвичай приховуються від користувача

3. Інкапсуляція. Успадкування. Поліморфізм

Механізм **успадкування** дозволяє створювати нові класи (**дочірні** чи **похідні**) на основі вже наявних (**батьківських** чи **базових**). При цьому дочірній клас успадковує всі властивості та методи батьківського класу, розширюючи його функціональність за рахунок нових властивостей та методів, реалізованих у ньому.

Спадкування дозволяє без переписування великих обсягів існуючого програмного коду створювати нові класи з покращеною функціональністю.

У C++ клас-спадкоємець створюється так:

```
// Батьківський клас
class A
{
	// …
};

// Дочірній клас
class B : public A
{
	// …
};
```

Наследование позволяет приобретать свойства родителей

3. Інкапсуляція. Успадкування. Поліморфізм

Розглянемо наступний приклад. Нехай потрібно розробити драйвери (програми керування) для різнотипних принтерів одного виробника. Передбачається, що вони мають бути максимально уніфіковані та відрізнятися лише деталями реалізації для кожного конкретного типу принтера (матричного, струменевого чи лазерного). Зрозуміло, що фізично спосіб друку у кожного типу принтера відрізняється. Але всі інші характеристики (підтримуваний формат паперу, швидкість і якість друку, кольору тощо) у них можуть бути уніфіковані.

Матричний принтер

Струменевий принтер

Лазерний принтер

3. Інкапсуляція. Успадкування. Поліморфізм

Деякий абстрактний, що описує принтер, можна в найзагальнішому вигляді описати, наприклад, так.

```
//---------------------------------------- ----------------------------------
// Абстрактний клас, що описує друкувальний пристрій
//---------------------------------------- ----------------------------------
class Printer
{
private:
	string format;	// Формат паперу, що підтримується, (наприклад, A4)
	string model;	// Модель принтера
	// ...
public:
	Printer(string f, string m)	// Конструктор та деструктор
	{
		format = f;
		model = m;
	}
	virtual ~Printer(void) {}
	virtual void print(void) = 0;	// Чиста віртуальна функція друку
};
```

3. Інкапсуляція. Успадкування. Поліморфізм

Якщо в класі на початку оголошення методу додано ключове слово **virtual**, це означає, що цей метод є **віртуальним** і може бути перевизначений в класах-спадкоємцях. Якщо оголошення віртуального методу закінчується виразом «= 0;», він називається **чистої віртуальною функцією**. Наявність чистих віртуальних функцій робить клас **абстрактним**. Абстрактний клас визначає інтерфейс для перевизначення похідними класами і створити об'єкт для нього не можна. Якщо, наприклад, написати у програмі наступний код, компілятор видасть помилку про використання абстрактного класу.

```
#include "Printer.h"
// ...

int main(void)
{
	Printer prn;  // Помилка безпосереднього використання абстрактного класу

	// …
	return 0;
}
```

3. Інкапсуляція. Успадкування. Поліморфізм

У цьому прикладі чиста віртуальна функція **print()** повинна реалізувати низькорівневу процедуру друку на принтері конкретної моделі. Зазвичай в результаті її роботи створюється двійковий файл, що містить коди управління конкретного принтера, який потім копіюється в його пам'ять і після цього фізично починається процес друку. Програмна реалізація даного методу здійснюється у класі-спадкоємці, що розробляється для конкретного типу принтера (або навіть моделі). Тут слід зазначити, що якщо від класу створюватимуться спадкоємці, його деструктор також слід оголошувати віртуальним методом. В цьому випадку при видаленні об'єкта похідного класу по ланцюжку будуть викликані й деструктори всіх його базових класів, що гарантує коректне звільнення пам'яті (і, відповідно, відсутність такого явища, як **витік пам'яті**!).

Припустимо, виробник на базі вже наявного принтера розробив нову вдосконалену модель, яка має низку додаткових можливостей. Замість розробки для нього драйвера «з нуля» достатньо створити дочірній клас з переписаною функцією print(), що програмно реалізує нові можливості друку.

3. Інкапсуляція. Успадкування. Поліморфізм

Наприклад.

```
// Клас, що реалізує драйвер для моделі "Type3"
class PrinterType3 : public Printer
{
public:
    PrinterType3(string f, string m) : Printer(f, m) {} // Виклик конструктора базового класу
    virtual ~PrinterType3(void)
    {
       // ...
    }
    // ...
    void print(void) // Реалізація друку для конкретної моделі
    {
        // do something
        // ...
    }
    // ...
};
```

3. Інкапсуляція. Успадкування. Поліморфізм

Припустимо, до комп'ютера підключено декілька принтерів. Реалізація друку на них в операційній системі може бути такою.

```
#include "Printer.h"

// Поліморфна функція друку
void sysPrint(Printer *prn)
{
    prn->print();      // Виведення на друк у конкретний пристрій
}

int main()
{
    Printer  *prn1 = new PrinterType3("A1", "EpsonDFX8000"),  // Створення об’єктів
             *prn2 = new PrinterType8("A4", "EpsonLX18");
    // ...
    sysPrint(prn1);         // Друк
    sysPrint(prn2);
    // ...
    delete prn1;            // Звільнення пам'яті (тут будуть викликані деструктори)
    delete prn2;
    return 0;
}
```

3. Інкапсуляція. Успадкування. Поліморфізм

У наведеному вище прикладі реалізована функція **sysPrint()**, що демонструє роботу **поліморфізму** – можливості функцій обробляти дані різних типів. Поліморфізм також дозволяє скоротити обсяги написання програмного коду, тому що для уніфікованої обробки даних різних типів можна використовувати той самий код.

У цьому прикладі функція sysPrint() в якості аргументу отримує покажчик на базовий абстрактний клас Printer, яким також є всі його похідні класи (PrinterType3 і PrinterType8). У тілі цієї функції викликається метод базового класу print(), який у батьківському класі є чистою віртуальною функцією, але має бути реалізований у кожному дочірньому класі. При виконанні коду prn->print(); буде автоматично визначено клас, на об'єкт якого посилається покажчик prn, та викликаний відповідний метод друку. Це називається **пізнім зв'язуванням**, що означає, що об'єкт пов'язується з викликом функції лише під час виконання програми, а не раніше.

4. Перевантаження операцій

Правильно спроектований клас має бути так само зручним у застосуванні, як і вбудований тип даних. І тому у класах C++ підтримується **перевантаження операцій** (один із способів реалізації поліморфізму). Під перевантаженням операцій розуміється можливість наявності однакових операцій (операторів), які різняться лише типами оброблюваних властивостей (операндів).

Наприклад, результат обчислення виразу «**a * b**» залежить від типів операндів a та b. Якщо це числа, то очевидно, що результатом є їхній добуток. А якщо це рядки? Яким результатом буде обчислення виразу «"Hello world!" * 3»? І чи коректний такий вираз? У мові програмування **Python**, наприклад, результатом його обчислення є новий рядок, утворений повторенням задану кількість разів вихідного.

```
Type "help", "copyright", "credits" or "license" for more information.
>>> "Hello world!" * 3
'Hello world!Hello world!Hello world!'
>>>
```

Таким чином, наприклад, операція множення може бути поліморфною, тобто результат її виконання залежить від типів параметрів.

Для реалізації такої можливості в ООП і підтримується перевантаження операцій.

4. Перевантаження операцій

У C++ для перевантаження операцій використовується спеціальна функція **operator()**. Розглянемо наступний приклад. Нехай необхідно реалізувати векторні обчислення на площині. Для цього створимо наступний клас Vector2D.

```
// Вектор на площині
class Vector2D
{
private:
    double x, y; // Координати вектора
public:
    Vector2D(void) : x(0), y(0) {}
    Vector2D(double px, double py) : x(px), y(py) {}
    Vector2D(const Vector2D &r) : x(r.x), y(r.y) {}      // Конструктор копіювання
    ~Vector2D(void) {}
    Vector2D operator = (const Vector2D &r)    // Перевантажений оператор присвоювання
    {
        x = r.x;      y = r.y;
        return *this;      // Повернення розіменованого покажчика на поточний об'єкт
    }
    friend ostream& operator << (ostream& out, const Vector2D &r) // Перевантажений оператор
    {                                                              // виведення (в стандартний потік)
        out << '(' << r.x << ',' << r.y << ')';
        return out;
    }
};
```

4. Перегрузка операций

У наведеному прикладі у класі Vector2D перевантажені дві операції: присвоєння («=») і побитового зсуву («<<»), яке тепер стало виведенням у стандартний потік.

Зазначимо, що спосіб перевантаження операцій залежить від того, чи вони є **унарними** чи **бінарними**. Унарна операція має один операнд, наприклад, "-a" (зміна знака). Бінарна – два операнди, наприклад, «a - b» (віднімання b від a). При перевантаженні бінарних операцій слід враховувати наступний нюанс. У кожен метод класу неявно передається ще один параметр, що містить адресу поточного об'єкта. До нього можна звернутися через використання спеціальної зарезервованої змінної **this**, що містить адресу поточного об'єкта.

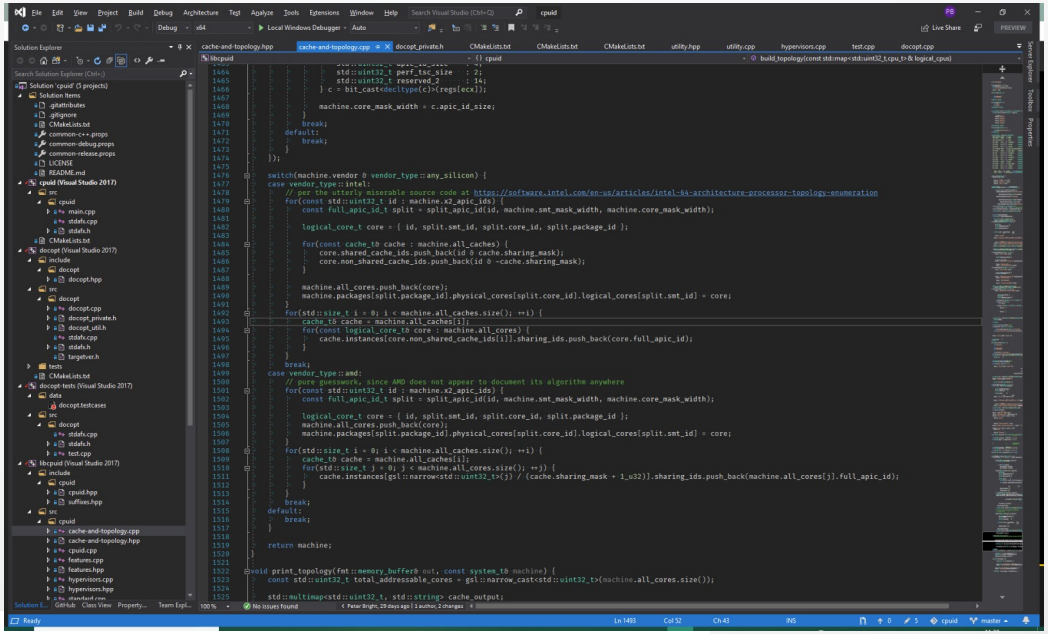
Таким чином, якщо в класі оголосити метод, що перевантажує бінарну операцію, він фактично повинен отримувати не два аргументи (операнда операції), а три (ще і this). Ця суперечність усувається наступним способом: функція, що перевантажує бінарну операцію, оголошується не як член класу (його метод), а як так звана **дружня функція** (**friend**), яка формально не є методом класу, але має доступ до його закритих (private) та захищених (protected) частин.

У наведеному вище прикладі перевантажена операція присвоєння є унарною, а операція виведення у потік – бінарною.

5. Системи програмування мовою C++

Для розробки мовою C++ використовується велика кількість різних інтегрованих середовищ розробки (IDE – Integrated Development Environment). Серед найпопулярніших можна назвати такі.

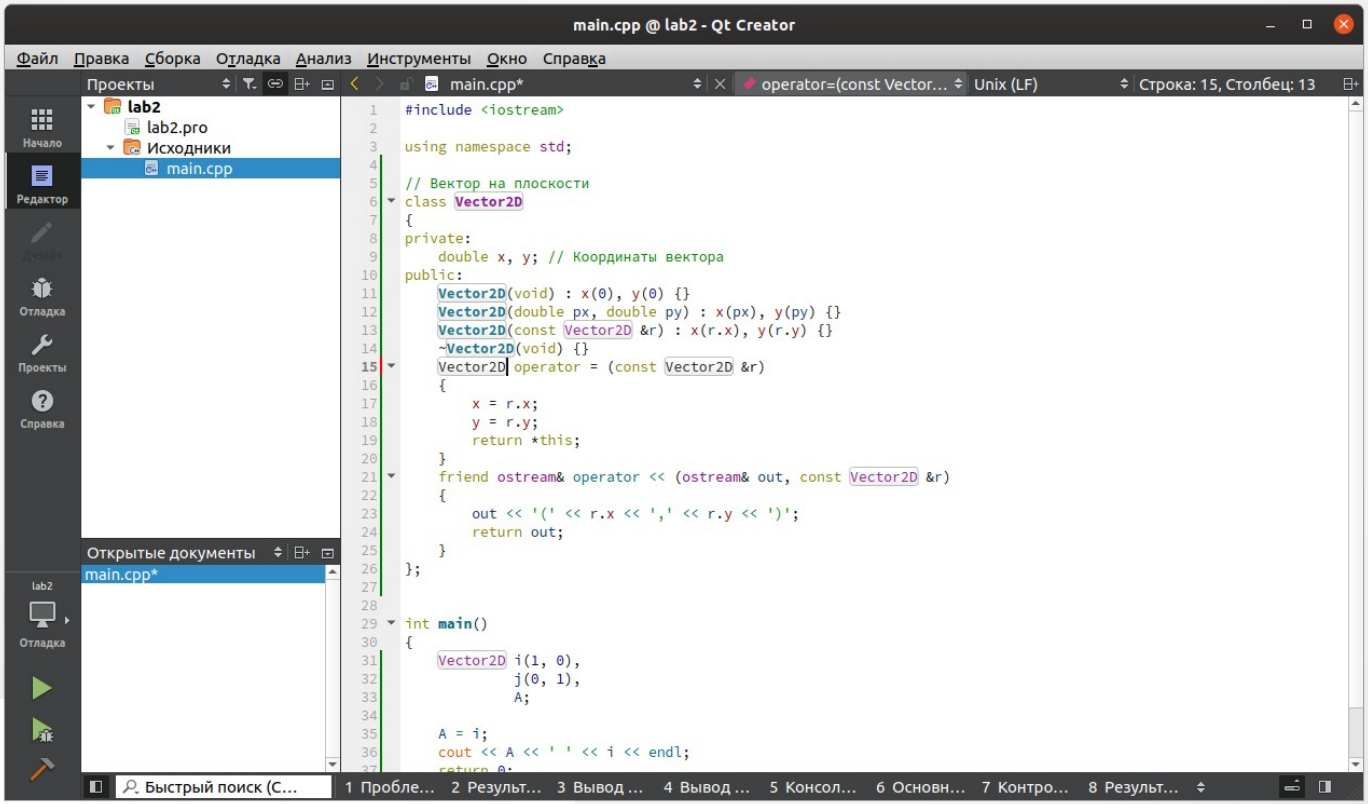
1. **Microsoft Visual Studio** – найпопулярніша IDE розробки програм для сімейства операційних систем Windows. Є пропріетарною, але для неї існують і безкоштовні версії з обмеженою функціональністю.



Visual Studio

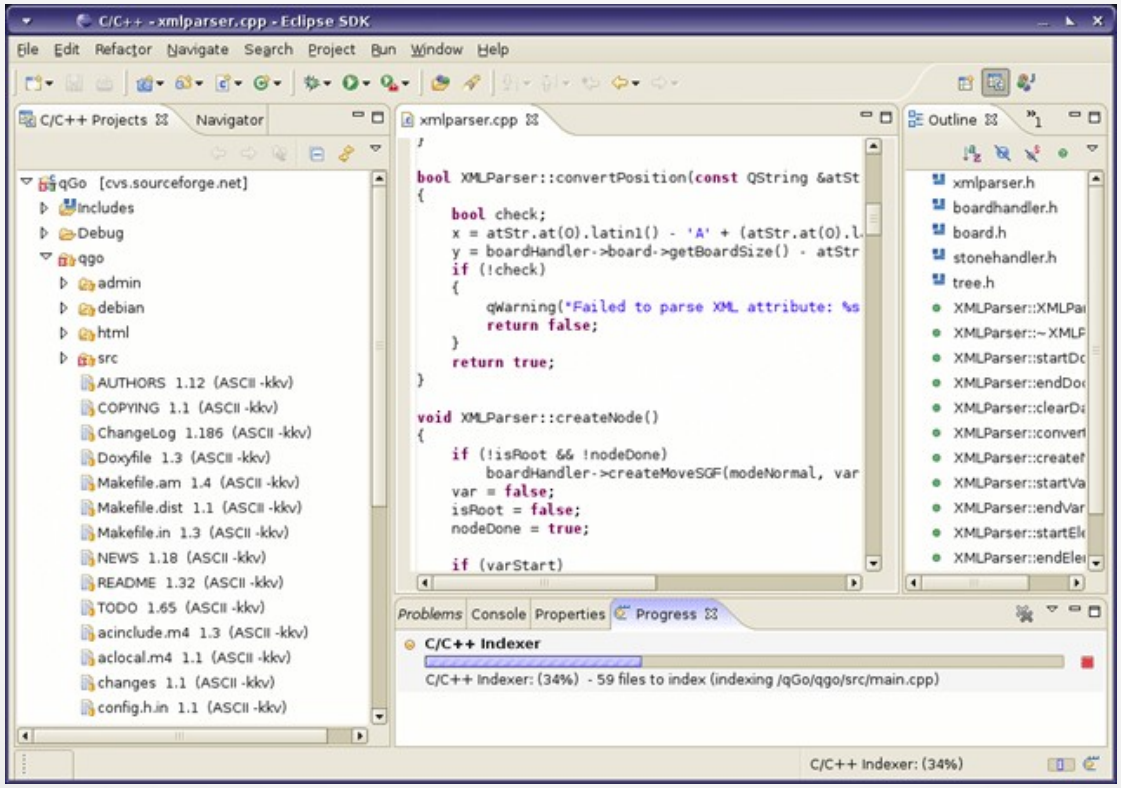
5. Системи програмування мовою C++

2. **Qt Creator** – популярна багатоплатформна IDE для розробки програм під ОС Windows, Linux і MacOS. Повнофункціональна версія цього середовища розробки також є проприєтарною, але є і безкоштовні версії.



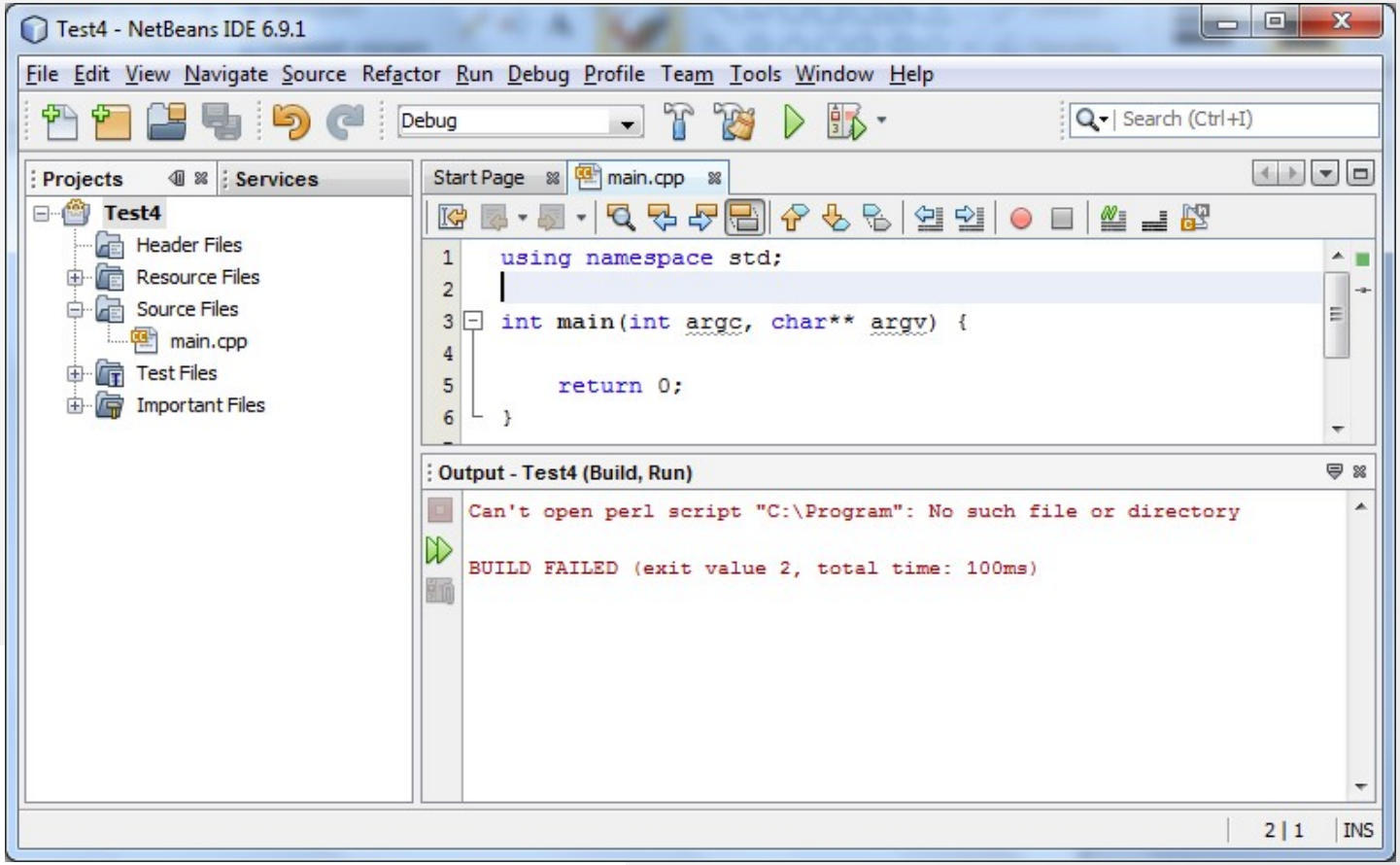
5. Системи програмування мовою C++

3. **Eclipse** – є одним із провідних IDE для розробників на C і C++. Це повністю безкоштовний кросплатформний програмний продукт із відкритим вихідним кодом, що працює з усіма основними ОС.



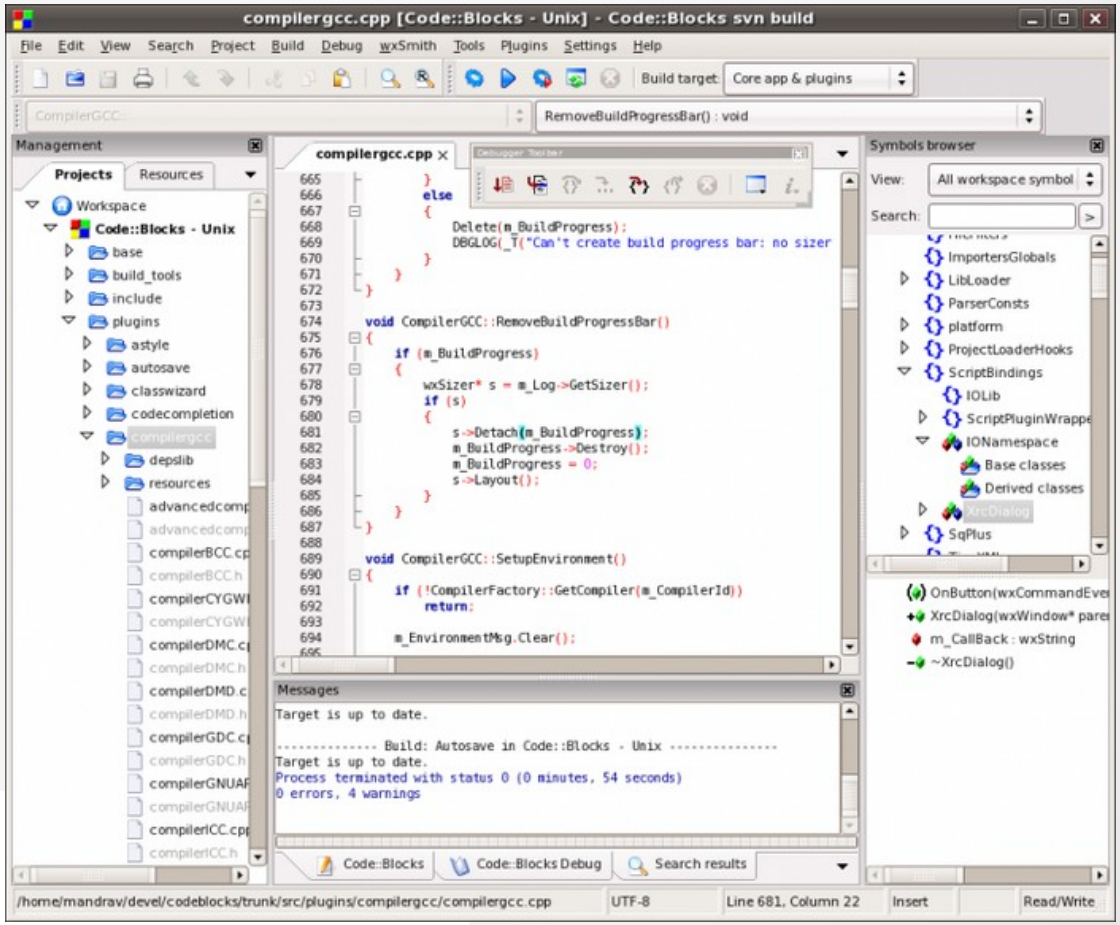
5. Системи програмування мовою C++

4. **NetBeans** – це також одна з популярних серед розробників кросплатформових IDE для програмування на C++ з відкритим вихідним кодом.



5. Системи програмування мовою C++

4. **Code::Blocks** – популярна легковажна кросплатформна IDE з відкритим вихідним кодом, що розповсюджується безкоштовно.



Code::Blocks

The open source, cross-platform IDE