Компілятор, інтерпретатор, компонувальник, компілятор в байт-код або проміжний код, JIT компілятор, система виконання (Runtime)

Компілятор

Це програма, яка перетворює вихідний код програми з одного мовного рівня (зазвичай високорівневої мови програмування) на машинний код або код цільової машини (наприклад, набір інструкцій процесора).

Після компіляції програму можна виконувати без необхідності перетворення вихідного коду щоразу при кожному запуску.

Компілятори часто використовуються в мовах програмування, таких як C, C++, Java (частково), і т.д.

Інтерпретатор

Інтерпретатор - це програма, яка читає і виконує вихідний код програми рядок за рядком без попередньої компіляції.

Кожна інструкція виконується під час виконання програми.

Інтерпретатори дозволяють відразу виконувати програми на будь-якій підтримуваній платформі, але, зазвичай, повільніше за компільовані мови.

Компонувальник (Linker)

Це програма, яка об'єднує окремі об'єктні файли, що були створені компілятором, в один виконуваний файл або бібліотеку.

Компонувальники розв'язують зовнішні посилання та розміщують символи відповідно до інструкцій програміста або стандартних правил.

Компілятор в байт-код або проміжний код

Компілятор, який перетворює вихідний код програми в проміжний код або байт-код, а не безпосередньо в машинний код.

Цей байт-код може бути виконаний інтерпретатором або компілятором в машинний код за часом виконання (JIT-компіляція).

JIT компілятор (Just-In-Time Compiler)

JIT-компілятор - це компонент системи виконання, який перетворює проміжний код (наприклад, байт-код Java) в машинний код під час виконання програми.

В результаті програми виконуються швидше, оскільки компіляція в машинний код відбувається на льоту, коли програма вже запущена.

Система виконання (Runtime)

Це середовище, яке надає необхідні ресурси та послуги для виконання програм.

Система виконання може включати в себе інтерпретатор, JIT-компілятор, бібліотеки інтерфейсів, алокатори пам'яті та інші компоненти, необхідні для виконання програм.

Компіляція

Процес, у якому компілятор читає записану початковою мовою програму та записує цільовою мовою, називають компіляцією (трансляцією, перекладом). Залежно від типу компілятора та налаштувань, цей процес може бути як простим однопрохідним зчитуванням і записом результату, так і розгалуженою багатокроковою ресурсомісткою обробкою й аналізом вхідного коду для численних оптимізацій та налаштувань.

Загалом компіляцію поділяють на такі послідовні та залежні кроки:

1.Аналіз (front-end) — зчитування та розбиття початкової програми на складові частини для створення проміжного представлення.

Лексичний аналіз — на цьому етапі послідовність символів сирцевого файлу перетворюється в послідовність лексем.

Синтаксичний аналіз, під час якого послідовність лексем перетворюється в дерево розбору.

Семантичний аналіз — перевірка відповідності правилам вхідної мови та побудова таблиці символів.

Генератор проміжного коду будує проміжне представлення для подальших оптимізацій.

2.Синтез (back-end) — побудова цільової програми на базі проміжного представлення.

Попередній аналіз проміжного представлення на залежності, потік даних та інші контекстні властивості, важливі для оптимізації.

Оптимізація — покращення для швидкодії, розміру, паралелізму тощо.

Генератор цільового коду створює кінцевий результат роботи компілятора — цільову програму.

У конкретних реалізаціях компіляторів ці етапи можуть бути розділені або, навпаки, поєднані в тому чи іншому вигляді.

Початкова мова визначається її синтаксисом — описом того, з яких конструкцій складається мова, та семантикою — набором правил, що визначають суть цих конструкцій.

За принципом роботи можна виділити такі види компіляторів:

- Однопрохідні — компіляція здійснюється в один прохід пропускаючи багато проміжних кроків оптимізацій і перевірок.

- Компілятори у зшитий код (англ. threaded code) — код, який повністю складається з підпрограм. По суті компілятор просто замінює кожну інструкцію вхідного коду на підпрограму вихідного — зшиває з заготівок.

- Інкрементальні компілятори — деякі функції можуть бути скомпільовані під час виконання інкрементально, наприклад у поєднанні з інтерпретованим виконанням.

- Передфінальний компілятор – компілятор, який компілює у вихідну мову теоретичної машини. Такий компілятор реалізований для мови програмування Prolog; він генерує вихідний код для абстрактної машини Уоррена (англ. Warren Abstract Machine).

- Динамічний компілятор (англ. JIT, just-in-time) — компіляція на льоту.

- Компілятор зі змінними цілями (англ. retargetable), який відносно просто можна змінити для генерації цільового коду під іншу архітектуру процесору. Загалом така властивість досягається шляхом зниження якості вихідного коду (у порівнянні з цільовими компіляторами під конкретний процесор). Представником такого типу компіляторів є набір GCC.

- Компілятор розпаралелення, який генерує вихідних код для запуску на паралельній архітектурі.

- Компілятор компіляторів — транслятор, що сприймає формальний опис мови програмування й генерує компілятор для цієї мови.

- Векторизувальний компілятор – компілятор, який транслює вихідний код в машинний код комп'ютерів оснащених векторним процесором.

- Багатопрохідний компілятор.

Інтерпретатор

Інтерпретатор мови програмування— програма чи технічні засоби, необхідні для виконання інших програм, вид транслятора, який здійснює пооператорну (покомандну, построкову) обробку, перетворення у машинний код та виконання програми або запиту (на відміну від компілятора, який транслює у машинні коди всю програму без її виконання).

Інтерпретатори можуть працювати як з початковим кодом програми (англ. source code), написаним мовою програмування, так і з байт-кодом (інтерпретатори байт-коду).

Коли потрібно написати програму, слід використовувати **компілятор або інтерпретатор**. Обидва ці інструменти потрібні, щоб перевести мову програмування в ту, яку розуміє комп'ютер. Хоча обидва інструменти виконують одне й те саме завдання, вони роблять це по-різному.

Основна відмінність між ними в тому, як вони обробляють вихідний код програми. Компілятор перетворює весь код на машинну мову, а інтерпретатор виконує код порядково.

Інтерпретатор виконує одну інструкцію за раз, транслюючи і виконуючи її, а потім переходячи до наступної. Компілятор же транслює всю програму відразу, а потім виконує її.

Компілятор генерує звіт про помилки після трансляції всієї програми, тоді як інтерпретатор припиняє трансляцію після першої знайденої помилки.

Компілятор потребує більше часу на аналіз і обробку мови високого рівня порівняно з інтерпретатором.

Час виконання коду компілятора швидший, ніж в інтерпретатора, не тільки через час аналізу й обробки, а й тому, що програма вже скомпільована в машинну мову.

Компілятор

Компілятор являє собою програму, яка переводить код однією мовою програмування на іншу. Він працює з програмою в цілому, перетворюючи її на виконуваний комп'ютерний код, оскільки комп'ютер може розпізнавати тільки двійковий код. Головне його завдання в тому, щоб перетворити вихідний код мовою програмування високого рівня на мову нижчого рівня. Прикладами мов, які використовують компіляцію, є C і C++.

Компілятор може виконувати безліч операцій, включно з попереднім опрацюванням даних, парсингом, семантичним аналізом, перетворенням програми на проміжне представлення, оптимізацією та генерацією коду. Він найефективніший під час роботи з програмами, які не потребують построкового виконання.

Сама ж компіляція – це процес, який дає змогу програмі працювати швидше, але потребує більше ресурсів і може бути складним для розуміння для тих, хто не знайомий з комп'ютерною технологією.

Переваги компілятора:

Виконання програмного коду відбувається швидше, ніж під час роботи з вихідним кодом.

Виконувані файли (у форматі .exe) можна запускати в будь-який час, без необхідності перекладу в машинний код.

Перевірка на наявність синтаксичних помилок відбувається під час компіляції.

Вихідний код програми зберігається в зашифрованому вигляді, що робить його менш доступним для користувача.

Недоліки компілятора:

Процес вимагає великого обсягу пам'яті комп'ютера.

Зміна програми можлива тільки шляхом повернення до вихідного коду.

Створення виконуваного файлу може зайняти багато часу.

Вихідний код має бути без помилок для успішної компіляції.

Інтерпретатор

Це програмний засіб, який виконує набір інструкцій, представлених у вигляді програмного коду високого рівня, без їх попередньої компіляції в машинний код. Цей набір інструкцій може бути представлений вихідним кодом, попередньо скомпільованим, або сценаріями. Приклади мов програмування, які використовують інтерпретатори, включають Perl, Python і Matlab.

Процес інтерпретація – це аналіз і виконання вихідної програми або запиту в режимі рядкової обробки, без попередньої трансляції в машинний код. Це дає можливість швидко перевіряти і тестувати код без необхідності його компіляції. Однак, процес інтерпретації вимагає більших обчислювальних потужностей і може призводити до більш повільної роботи програми.

Переваги інтерпретатора:

Значно полегшує роботу з вихідним кодом програми.

Використовує мінімальний обсяг пам'яті комп'ютера під час перекладу за однією інструкцією.

Може пов'язати повідомлення про помилки з виконанням інструкцій.

Недоліки інтерпретатора:

Витрачається час на інтерпретацію програми кожного разу, коли вона виконується.

Можливість виконання тільки на комп'ютерах, де встановлено відповідний транслятор.

	Компілятор	**Інтерпретатор**
Що це	Програма, яка перетворює вихідний код мовою високого рівня на машинну мову	Програма, яка виконує вихідний код мовою високого рівня
Робота	Читає весь код програми і створює виконуваний файл	Читає і виконує кожен рядок коду по черзі
Помилка	Повідомлення про помилки виводяться після проходження компіляції	Повідомлення про помилки виводяться в міру виконання коду
Виконання	Виконувані файли працюють швидше	Інтерпретований код працює повільніше, ніж скомпільований
Використання	Рекомендується для великих проектів і для мов, де потрібна висока продуктивність	Рекомендується для швидкого розроблення та налагодження, а також для мов, які використовуються в інтерактивному середовищі
Приклади мов	C++, Java, Swift, Rust, Go	Python, Ruby, JavaScript, PHP, Perl