9.4. Трансляція та виконання: компілятор, інтерпретатор, компонувальник

Трансляція – перетворення програми, яка подана однією мовою програмування, в еквівалентну програму іншою мовою.

Трансляція програми - перетворення програми, представленої однією з мов програмування, в програму іншою мовою і, у певному сенсі, рівносильну першій. Під час трансляції виконується переклад програми, зрозумілої людині, на мову, зрозумілу комп'ютеру. Виконується спеціальними програмними засобами (транслятором).

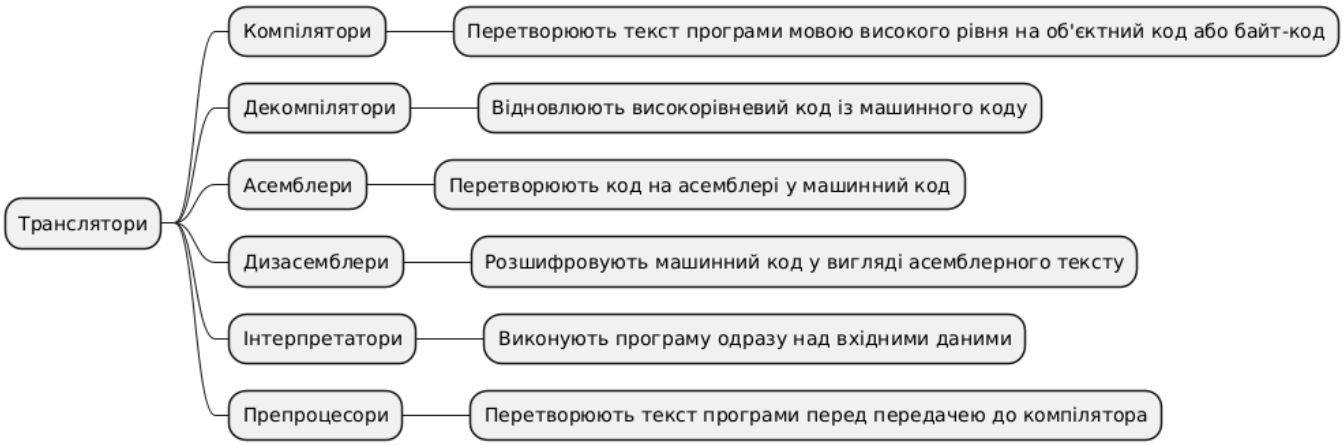
Транслятори реалізуються як компіляторів чи інтерпретаторів. З погляду виконання роботи компілятор та інтерпретатор суттєво різняться. Якщо мета трансляції – перетворення всього вихідного тексту на внутрішню мову комп'ютера (тобто отримання деякого нового коду) і лише, то така трансляція називається також компіляцією. Вихідний текст називається також вихідною програмою чи вихідним модулем, а результат компіляції – об'єктним кодом чи об'єктним модулем. Якщо трансляції піддаються окремі оператори вихідних текстів і при цьому отримані коди відразу виконуються, така трансляція називається інтерпретацією. Оскільки трансляція виконується спеціальними програмними засобами (трансляторами), останні звуться компілятора чи інтерпретатора, відповідно.

Мета трансляції - перетворити текст з однієї мови на іншу, яка зрозуміла адресату тексту. У випадку програм-трансляторів адресатом є технічний пристрій (процесор) або програма-інтерпретатор.

Види трансляторів

Вид	Характеристика
Адресний	Функціональний пристрій, що перетворює віртуальну адресу на реальну адресу
Діалоговий	Забезпечує використання мови програмування в режимі розподілу часу
Багатопрохідний	Формує об'єктний модуль за кілька переглядів вихідної програми
Зворотній	Те саме, що детранслятор (декомпілятор, дизассемблер)
Однопрохідний	Формує об'єктний модуль за послідовний перегляд вихідної програми
Оптимізуючий	Виконує оптимізацію коду в об'єктному модулі, що створюється
Синтаксично-орієнтований (синтаксично-керований)	Отримує на вхід опис синтаксису та семантики мови та текст описаною мовою, який і транслюється відповідно до заданого опису
Тестовий	Набір макрокоманд мови асемблера, що дозволяють задавати різні процедури налагодження в програмах, складених мовою асемблера

Транслятор (англ. translator) – програма або технічний засіб, який виконує перетворення чи іншу обробку текстів програм. Їх класифікація наведена на рисунку.



Оскільки компілятори та інтерпретатори реалізують мови програмування, вони мають спільні риси: їх структура досить схожа, в основу їх реалізації покладено спільні теоретичні результати та практичні методи реалізації.

Компілятор (англ. Compiler від англ. to compile – збирати в ціле) – комп'ютерна програма (або їх набір), що перетворює (компілює) вихідний код, написаний певною мовою програмування (вихідна мова, англ. source language), на семантично еквівалентний код в іншій мові програмування (цільова мова, англ. target language), який зазвичай необхідний для виконання програми машиною, наприклад, комп'ютером. В СРСР, зокрема в УРСР, з 1952 року до середини 1960-х років використовувався термін програмувальна програма (ПП).

Коротко компілятор можна визначити як програму або технічний засіб, що виконує компіляцію.

Історично компілятором називалась програма, що зв'язувала підпрограми, чим і зумовлено походження слова. Сьогодні це завдання виконує компонувальник.

Для виконання програма не завжди повинна бути перекладена компілятором, існує також інший принцип: покрокове виконання програмних інструкцій інтерпретатором.

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**:

1. Однопрохідні – компіляція здійснюється в один прохід пропускаючи багато проміжних кроків оптимізацій і перевірок. Перші компілятори мови Pascal.[7]
2. 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.[8] Такий компілятор має мова програмування Forth.[9]
3. Інкрементальні компілятори – деякі функції можуть бути скомпільовані під час виконання інкрементально, наприклад у поєднанні з інтерпретованим виконанням. Такий тип компіляторів поширений у сімействі LISP.
4. Передфінальний компілятор (англ. stage compiler), 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).[10]
5. Динамічний компілятор (англ. JIT, just-in-time) – компіляція на льоту. Див. нижче.
6. 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.
7. Компілятор розпаралелення, який генерує вихідних код для запуску на паралельній архітектурі.
8. Компілятор компіляторів[en] – транслятор, що сприймає формальний опис мови програмування й генерує компілятор для цієї мови.[джерело?]
9. Векторизувальний. Транслює вихідний код в машинний код комп'ютерів оснащених векторним процесором.
10. Багатопрохідний компілятор[en].

Стратегії компіляції. Компілятори слугують своїм цілям у різний спосіб для різних мов програмування та парадигм. Так, якщо історично первинні компілятори транслювали код мови у апаратно залежний асемблер, то з часом виникла потреба у додаткових методах та шарах абстракції – компіляції на льоту, транслювання більш високорівневого коду в менший за рівнем, оптимізації по використанню вже скомпільованого коду тощо.

Поширені стратегії компіляції

1. Компіляція перед виконанням (англ. AOT, ahead-of-time). Класичний принцип, коли компіляція проводиться окремо від виконання, зазвичай у відмінному від місця виконання середовищі. Компілятори мов програмування Pascal, C, C++, BASIC, Fortran, COBOL використовують стратегію такої попередньої компіляції.

Переваги:

- швидке завантаження в браузері. Витрачається менше часу за рахунок того, що застосунок компілюється до завантаження в браузер та кінцеві файли мають менший розмір;
- виявлення помилок під час збірки. Мається можливість виправити всі помилки до запуску застосунку в режимі експлуатації;
- підвищена безпека.

2. Компіляція під час виконання (англ. JIT, just-in-time), відома також як динамічна компіляція. Суть полягає у компіляції часу виконання (англ. run time), коли текст вхідної мови перетворюється у машинний код на льоту й тут же виконується. Ця техніка поєднує у собі методи як попередньої компіляції у проміжний код так і інтерпретації цього проміжного коду під час виконання. Будучи більш продвинутою технікою, компіляція на льоту має досить суворі вимоги до простоти компіляції, і не є доцільною для певних мов програмування.[11] Принциповим представником JIT компіляції є середовище виконання мови Java, піонером же вважається Smalltalk.
3. Транскомпіляція, або компіляція з однієї високорівневої мови в іншу (англ. source-to-source). Принциповою відмінністю даної стратегії є те, що як вхідний так і вихідний код є мовами програмування високого рівня. Типовими прикладами використання є підготовка коду для паралельної оптимізації, перекомпіляція старішого коду для нової версії стандарту мови програмування, компіляція мов-надбудов у базову мову. Представниками таких мов програмування є CoffeeScript[12], Dart, Haxe, Coccinelle[13].
4. Перекомпіляція – динамічна компіляція частин програми під час виконання (англ. dynamic recompilation). Особливість деяких серед виконання – емуляторів та віртуальних машин, перекомпільовувати деяку частину програми під час її роботи. Ця техніка використовується для переносу коду на іншу архітектуру під час його виконання, зокрема для запуску застарілих програм на сучасних операційних системах. Широко використовується Java run time[14] та .Net Common Language Runtime[15], а також багатьма віртуальними машинами.

Основними завданнями, які повинен виконувати динамічний рекомпілятор, є:

- читання машинного коду з вихідної платформи;
- видає машинний код для цільової платформи.

Відомі компілятори

Компілятор	Опис
Amsterdam Compiler Kit	набір засобів для написання компіляторів для системи Minix авторства Ендрю Таненбаума та Серіла Якобса.
GCC	відомий набір компіляторів C та пізніше C++, створений Річардом Столменом; широко поширений у світі Linux.
Clang	компілятор сімейства C/C++, який використовує технологію LLVM – віртуальної машини на базі проміжного представлення коду
Turbo Pascal	видатний компілятор створений Андерсом Гейлсбергом у 1983 році.
Turbo C	компілятор мови C розроблений компанією Borland у 1987 році
Mono	багатоплатформовий набір засобів та компілятор мови C#, у вільному доступі.
PyPy	JIT-компілятор мови Python написаний на мові Python.

Генератори аналізаторів. Побудовані алгоритми, що перетворюють опис вхідної мови у програму, що виконує аналіз і є велика кількість реалізацій цих алгоритмів. Є також утиліти, що автоматизують решту фаз компіляції та системи створення компіляторів у цілому.

В Unix поширені генератор лексичних аналізаторів (F)Lex, та генератори синтаксичних аналізаторів Bison та Yacc.

Інтерпретатор мови програмування (interpreter) – 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).

Типи інтерпретаторів

Простий інтерпретатор аналізує і відразу виконує (власне інтерпретація) програму покомандно (або порядково), по мірі надходження тексту програми на вхід інтерпретатора. Перевагою такого підходу є миттєва реакція. Недолік – такий інтерпретатор виявляє помилки в тексті програми тільки при спробі виконання команди (або рядка) з помилкою.

Інтерпретатор компілюючого типу – це система з компілятора, який перекладає текст програми в проміжне представлення, наприклад, в байт-код або p-код, і власне інтерпретатора, який виконує отриманий проміжний код (так звана віртуальна машина). Перевагою таких систем є більша швидкодія виконання програм (за рахунок винесення аналізу початкового коду в окремий, разовий прохід, і мінімізації цього аналізу в інтерпретаторі). Недоліки – більші вимоги до ресурсів і вимога на коректність тексту програми. Застосовується в таких мовах, як Java, Tcl, Perl (використовується байт-код), REXX (зберігається результат синтаксичного аналізу), а також у різних СУБД (використовується p-код).

Інтерпретатор компілюючого типу складається з компілятора мови і простого інтерпретатора з мінімізованим аналізом початкового коду; цей код у такому випадку не обов'язково повинен мати текстовий формат – це може бути машинний код якоїсь наявної апаратної платформи. Наприклад, віртуальні машини типу QEMU, Bochs, VMware містять у собі інтерпретатори машинного коду процесорів сімейства x86.

Деякі інтерпретатори (наприклад, для мов Lisp, Scheme, Python, Basic та інших) можуть працювати в режимі діалогу або так званого циклу читання-обчислення-друку (англ. read-eval-print loop, REPL). У такому режимі інтерпретатор зчитує закінчену конструкцію мови (наприклад, s-expression у мові Lisp), виконує її, друкує результати, після чого переходить до очікування введення користувачем наступної конструкції.

Унікальною є мова Forth, яка здатна працювати як в режимі інтерпретації, так і компіляції вхідних даних, дозволяючи переключатись між цими режимами в довільний момент, як під час трансляції початкового коду, так і під час роботи програм.

Слід також зазначити, що режими інтерпретації можна знайти не тільки в програмному, а й апаратному забезпеченні. Так, багато мікропроцесорів інтерпретують машинний код за допомогою вбудованих мікропрограм, а процесори сімейства x86, починаючи з Pentium (наприклад, на архітектурі Intel P6), під час виконання машинного коду попередньо транслюють його у внутрішній формат (в послідовність мікрооперацій).

Порівняння інтерпретатора та компілятора. Програми, як правило, пишуть мовою високого рівня, яка повинна бути перетворена в машинний код для виконання центральним процесором. Це перетворення виконує компілятор або інтерпретатор.

Під час **розробки програмного забезпечення**, програмісти роблять часті зміни у початковому коді. При використанні компіляторів, кожен раз після внесення змін у початковий код компілятор транслює змінені початкові файли і компонує всі файли бінарного коду разом, перш ніж програма може бути виконана. Чим більша програма, тим більшим є час очікування компілювання. З іншого боку, при використанні інтерпретатора, очікування набагато менше, бо інтерпретатору не треба транслювати всю програму, а просто потрібно транслювати код, що зараз виконується, на проміжне представлення (або не транслювати його взагалі), що вимагає набагато менше часу, щоб програма могла бути виконана.

Розповсюдження. Компілятор перетворює початковий код у бінарні інструкції для процесора певної архітектури, що робить його менш портативним. Такий переклад здійснюється тільки один раз в середовищі розробника, після чого той же бінарний файл можна розповсюдити на машини користувача, де він може бути виконаний без додаткового перекладу. Крос-компілятор може генерувати бінарний код для машини користувача, навіть якщо вона має інший процесор, ніж машина розробника, на якій відбувалась компіляція коду.

Програма, що інтерпретується, може поширюватися у вигляді початкового коду. Вона має бути трансльована на кожній машині, що займає більше часу, але робить розповсюдження програми незалежним від архітектури машини. Однак переносимість початкового коду, що інтерпретується, залежить від того, чи має цільова машина відповідний інтерпретатор. Якщо інтерпретатор треба розповсюджувати разом з початковим кодом, загальний процес встановлення програми ускладнюється, порівняно з постачанням одного виконуваного файлу. Те, що інтерпретований код легко читається і копіюється людьми, може представляти проблему з точки зору авторського права. Тим не менш, існують різноманітні системи шифрування і заплутування. Доставка проміжного коду, наприклад, байт-коду, має такий же ефект заплутування, але байт-код можна декодувати з допомогою декомпілятора або дизасемблера.

Ефективність. Основним недоліком інтерпретованих програм є те, що процес інтерпретації зазвичай набагато повільніший, ніж запуск скомпільованої програми. Різниця в швидкості може різнитися від незначної до достатньо відчутної: часто на порядок, а іноді й більше. Проте час інтерпретації програми може бути швидшим, ніж загальний час, необхідний для компіляції і запуску. Це особливо важливо, під час прототипування і тестування коду: цикл редагування-інтерпретація-налагодження часто може бути набагато коротший, ніж редагування-компіляція-запуск-налагодження.

Інтерпретація коду відбувається повільніше, ніж запуск скомпільованого коду, тому що інтерпретатор повинен аналізувати кожну інструкцію у програмі кожного разу, коли вона виконується, а потім виконувати потрібну дію, в той час як скомпільований код просто виконує фіксовані дії, визначені під час компіляції. Цей аналіз під час виконання відомий як «додаткові витрати інтерпретації». Доступ до змінних в інтерпретованих програм також повільніший, тому що операція зв'язування ідентифікаторів з місцями зберігання повторюється під час виконання, тоді як компілятором виконується один раз під час компіляції.

Існують різні компроміси між швидкістю розробки програмного забезпечення при використанні інтерпретатора і швидкістю виконання програми при використанні компілятора. Деякі системи (наприклад, Lisp) дозволяють інтерпретованому і скомпільованому коду викликати один одного і обмінюватися змінними. Це означає, що поточний код, який був протестований і налагоджений інтерпретатором, може бути скомпільований, і таким чином отримати більшу швидкість виконання, в той час як інший код розробляється. Багато інтерпретаторів не виконують оператори програми безпосередньо, а приводять його до більш компактної внутрішньої форми. Багато інтерпретаторів мови BASIC замінюють зарезервовані слова одним байтом токена, який може бути використаний для пошуку команди в таблиці переходів.

Зазвичай початковий код мови програмування високого рівня компілюється у проміжну мову, яка буде скомпільована або інтерпретована у машинний код.

Компонувальник (також редактор зв'язків, лінкер – від англ. Link editor, linker) – програма, яка виконує компонування (англ. linking) – приймає на вхід один або кілька об'єктних модулів (та/або бібліотек) і збирає їх в один виконуваний модуль.

Для зв'язування модулів компонувальник використовує таблиці імен ідентифікаторів, створені компілятором в кожному з об'єктних модулів. Такі імена можуть бути двох типів:

- Певні або експортовані назви функцій та змінних, визначені в даному модулі й надані для використання іншим модулям
- Невизначені або імпортовані імена – функції та змінні, на які посилається модуль, але не визначає їх всередині себе.

Робота компонувальника полягає в тому, щоб в кожному модулі конкретизувати посилання на невизначені імена. Для кожного імпортованого імені, визначення якого перебуває в інших модулях, згадування імені замінюється на його адресу.