1. Мовні процесори. Стадії збірки виконуваного файлу

Мовні процесори - це програмні засоби, що перетворюють програми, написані на одній мові програмування, в програми на іншій мові або в машинний код.

Типи мовних процесорів:

- **Компілятори** - перетворюють програму з мови високого рівня в машинний код
- **Інтерпретатори** - виконують програму безпосередньо, не створюючи машинний код
- **Препроцесори** - виконують попередню обробку тексту програми
- **Асемблери** - перетворюють програми з мови асемблера в машинний код
- **Лінкери** - об'єднують окремі модулі в єдиний виконуваний файл

Стадії збірки виконуваного файлу:

1. **Препроцесування** - обробка директив препроцесора (#include, #define)
2. **Компіляція** - перетворення вихідного коду в асемблерний код
3. **Асемблювання** - перетворення асемблерного коду в об'єктний файл
4. **Лінкування** - об'єднання об'єктних файлів та бібліотек у виконуваний файл

2. Структура та фази компілятора

Основні фази компілятора:

Фронт-енд (Frontend):

1. **Лексичний аналіз** - розбиття вхідного тексту на лексеми (токени)
2. **Синтаксичний аналіз** - побудова дерева розбору або AST
3. **Семантичний аналіз** - перевірка семантичних правил

Бек-енд (Backend):

4. **Генерація проміжного коду** - створення машинно-незалежного коду
5. **Оптимізація коду** - покращення ефективності коду
6. **Генерація машинного коду** - створення цільового коду

Допоміжні компоненти:

- **Таблиця символів** - зберігає інформацію про ідентифікатори
- **Обробка помилок** - виявлення та повідомлення про помилки

3. Формальні мови та граматики

Формальна мова - це множина рядків (слів) над деяким алфавітом, які утворені за певними правилами.

Граматика - це формальний опис правил утворення речень мови.

Компоненти граматики:

- **Термінальні символи** (T) - символи алфавіту мови
- **Нетермінальні символи** (N) - допоміжні символи
- **Стартовий символ** (S) - головний нетермінальний символ
- **Правила виводу** (P) - правила перетворення нетерміналів

4. Класифікація мов програмування

За парадигмою:

- **Імперативні** - описують послідовність дій (C, Pascal)
- **Декларативні** - описують що потрібно зробити (SQL, Prolog)
- **Об'єктно-орієнтовані** - базуються на концепції об'єктів (Java, C++)
- **Функціональні** - базуються на функціях (Haskell, Lisp)

За рівнем абстракції:

- **Низького рівня** - асемблер, машинний код
- **Високого рівня** - C++, Java, Python

За способом виконання:

- **Компільовані** - C, C++, Pascal
- **Інтерпретовані** - Python, JavaScript

5. Ієрархія формальних граматик Н. Чомського

Типи граматик (від найзагальніших до найспеціальніших):

1. **Тип 0 (Граматики загального вигляду)**
   - Без обмежень на правила
   - Розпізнаються машинами Тьюринга
2. **Тип 1 (Контекстно-залежні граматики)**
   - Правила вигляду: $\alpha A \beta \rightarrow \alpha \gamma \beta$, де $|\gamma| \geq 1$
   - Розпізнаються лінійно-обмеженими автоматами
3. **Тип 2 (Контекстно-вільні граматики)**
   - Правила вигляду: $A \rightarrow \alpha$
   - Розпізнаються автоматами з магазинною пам'яттю
4. **Тип 3 (Регулярні граматики)**
   - Правила вигляду: $A \rightarrow aB$ або $A \rightarrow a$
   - Розпізнаються скінченними автоматами

6. Лексичний аналізатор

Лексичний аналізатор (сканер) - перша фаза компілятора, яка читає вхідний текст і розбиває його на лексеми (токени).

Функції лексичного аналізатора:

- Розпізнавання лексем (ключові слова, ідентифікатори, константи)
- Видалення пробілів та коментарів
- Обробка директив препроцесора
- Виявлення лексичних помилок

Типові проблеми:

- **Максимальна згодність** - вибір найдовшої лексеми
- **Ключові слова vs ідентифікатори** - розрізнення зарезервованих слів
- **Числові константи** - різні формати чисел

Лексичний аналізатор на основі діаграм переходів:

- Використовує граф станів для розпізнавання лексем
- Кожен стан відповідає частковому розпізнаванню лексеми
- Переходи між станами відбуваються за символами вхідного потоку

7. Операції з формальними мовами

Основні операції:

- **Об'єднання** ($L_1 \cup L_2$) - множина рядків, що належать L_1 або L_2
- **Конкатенація** (L_1L_2) - множина рядків xy, де $x \in L_1$, $y \in L_2$
- **Замикання Кліні** (L^*) - множина всіх скінченних конкатенацій рядків з L
- **Позитивне замикання** (L^+) - L^* без порожнього рядка
- **Доповнення** ($\bar{L}$) - всі рядки, що не належать L
- **Перетин** ($L_1 \cap L_2$) - рядки, що належать обом мовам

8. Регулярні вирази та мови

Регулярний вираз - формальний спосіб опису регулярних мов.

Базові конструкції:

- **a** - символ 'a'
- **ε** - порожній рядок
- **∅** - порожня множина
- $\mathbf{r_1|r_2}$ - альтернатива (об'єднання)
- $\mathbf{r_1r_2}$ - конкатенація
- **r*** - замикання Кліні

Приклади:

- (a|b)* - всі рядки з символів a і b
- a*b+ - нуль або більше 'a', потім одна або більше 'b'

9. Скінченні автомати

Скінченний автомат - математична модель обчислення з скінченною кількістю станів.

Типи:

1. **Детерміновані (DFA)** - з кожного стану є точно один перехід за кожним символом
2. **Недетерміновані (NFA)** - можуть мати кілька переходів за одним символом

Компоненти автомата:

- **Q** - множина станів
- **Σ** - вхідний алфавіт
- **δ** - функція переходів
- $\mathbf{q_0}$ - початковий стан
- **F** - множина фінальних станів

Розпізнавання мови:

Автомат приймає рядок, якщо після читання всіх символів він знаходиться у фінальному стані.

10. Детермінізація скінченних автоматів

Детермінізація - процес перетворення NFA в еквівалентний DFA.

Алгоритм конструкції підмножин:

1. Створити стан DFA для кожної підмножини станів NFA
2. Обчислити ε-замикання для кожного стану
3. Побудувати функцію переходів для нових станів
4. Відмітити фінальні стани DFA

Оптимізація лексичного аналізатора:

- **Мінімізація автомата** - зменшення кількості станів
- **Таблиці переходів** - ефективне представлення функції переходів
- **Пряме кодування** - генерація коду без побудови таблиць

11. Породження лексичних аналізаторів (lex)

Lex - генератор лексичних аналізаторів, який створює код сканера за специфікацією.

Структура lex-файлу:

```
%{
/* C-код */
%}
```

/* визначення */
%%
/* правила */
%%
/* додатковий C-код */

Переваги:

- Автоматична генерація коду
- Оптимізовані автомати
- Підтримка складних шаблонів

12. Синтаксичний аналізатор

Синтаксичний аналізатор (парсер) - друга фаза компілятора, яка будує дерево розбору або AST.

Функції:

- Перевірка синтаксичної правильності
- Побудова структурного представлення програми
- Виявлення синтаксичних помилок

Види синтаксичного розбору:

1. **Зверху-вниз** - починає зі стартового символу
2. **Знизу-вгору** - починає з терміналів

13. Обробка помилок при компіляції

Типи помилок:

- **Лексичні** - недопустимі символи
- **Синтаксичні** - порушення граматики
- **Семантичні** - порушення семантичних правил

Стратегії обробки помилок:

1. **Режим паніки** - пропуск символів до синхронізуючого
2. **Фразовий рівень** - локальні виправлення
3. **Продукції для помилок** - спеціальні правила граматики
4. **Глобальна корекція** - мінімальні зміни в програмі

14. Контекстно-вільні мови

Контекстно-вільна мова - мова, породжена контекстно-вільною граматикою.

Властивості:

- Розпізнаються автоматами з магазинною пам'яттю

- Замкнені відносно об'єднання, конкатенації, замикання
- Не замкнені відносно перетину та доповнення

Види виводу:

- **Лівий вивід** - заміна найлівішого нетерміналу
- **Правий вивід** - заміна найправішого нетерміналу

15. Неоднозначність граматики

Неоднозначна граматика - граматика, для якої існує рядок з більш ніж одним деревом розбору.

Усунення неоднозначності:

- **Пріоритети операторів** - встановлення порядку виконання
- **Асоціативність** - напрямок групування операторів
- **Переписування граматики** - створення однозначних правил

16. Розбір зверху-вниз

Метод рекурсивного спуску:

- Кожен нетермінал відповідає процедурі
- Процедури викликають одна одну відповідно до правил граматики
- Простий у реалізації, але може зациклитися на ліворекурсивних правилах

Усунення ліворекурсивності:

Пряма ліворекурсивність: $A \to A\alpha \mid \beta$ **Усунення**: $A \to \beta A'$, $A' \to \alpha A' \mid \varepsilon$

Розбір з передбаченням:

- Використовує k символів для прийняття рішення
- LL(k) граматики - розбір зліва направо з лівим виводом

17. LL-граматики

LL(k) - граматики, які можна розбирати зліва направо з лівим виводом, дивлячись на k символів вперед.

Умови LL(1):

- Відсутність ліворекурсивності
- Для кожного нетерміналу A: $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$
- Якщо $\varepsilon \in FIRST(\alpha)$, то $FIRST(\beta) \cap FOLLOW(A) = \emptyset$

Множини FIRST і FOLLOW:

- **FIRST(α)** - множина терміналів, з яких може починатися α
- **FOLLOW(A)** - множина терміналів, що можуть слідувати за A

18. Висхідний синтаксичний розбір

Метод згортка-переніс:

- **Перенос** - читання наступного символу
- **Згортка** - заміна правої частини правила лівою

Стани парсера:

- Кожен стан представляє множину елементів (items)
- Елемент показує, яка частина правила вже розпізнана

19. LR-парсери

LR(k) - розбір зліва направо з правим виводом, дивлячись на k символів вперед.

Типи LR-парсерів:

1. **LR(0)** - без передбачення
2. **SLR(1)** - простий LR з одним символом передбачення
3. **LALR(1)** - скорочений LR(1)
4. **LR(1)** - канонічний LR з одним символом

Канонічний LR(0)-автомат:

- Стани - множини LR(0)-елементів
- Переходи за терміналами та нетерміналами
- Конфлікти згортка-згортка та переніс-згортка

20. Автомати з магазинною пам'яттю

Автомат з магазинною пам'яттю (МП-автомат) - скінченний автомат із стеком.

Компоненти:

- **Q** - множина станів
- **Σ** - вхідний алфавіт
- **Γ** - алфавіт стека
- **δ** - функція переходів
- $\mathbf{q_0}$ - початковий стан
- $\mathbf{Z_0}$ - початковий символ стека
- **F** - фінальні стани

Способи прийняття:

1. **За фінальним станом**
2. **За порожнім стеком**

21. Синтаксично-керовані визначення

Синтаксично-кероване визначення - граматика з приєднаними семантичними правилами.

Типи атрибутів:

- **Синтезовані** - обчислюються знизу-вгору в дереві розбору
- **Спадкові** - передаються зверху-вниз

Граф залежності атрибутів:

- Вершини - екземпляри атрибутів
- Ребра - залежності між атрибутами
- Повинен бути ациклічним

L-атрибутні граматики:

- Спадкові атрибути можуть залежати тільки від атрибутів лівих братів
- Можуть оцінюватися за один прохід зліва направо

22. Семантичний аналіз

Семантичний аналіз - перевірка семантичної правильності програми.

Основні завдання:

- Контроль типів
- Перевірка оголошень
- Контроль області видимості
- Перевірка сумісності операцій

Контроль області видимості:

- **Блочна структура** - вкладені області видимості
- **Таблиця символів** - зберігання інформації про ідентифікатори
- **Стек областей** - управління вкладеними областями

Системи типів:

- **Статична типізація** - перевірка типів під час компіляції
- **Динамічна типізація** - перевірка типів під час виконання
- **Сильна типізація** - суворі правила сумісності типів
- **Слабка типізація** - автоматичні перетворення типів

Правила виведення типів:

- Формальний опис типізації за допомогою логічних правил
- Дозволяє автоматично виводити типи виразів
- Базується на теорії типів