ДОДАТОК А

Слайди презентації

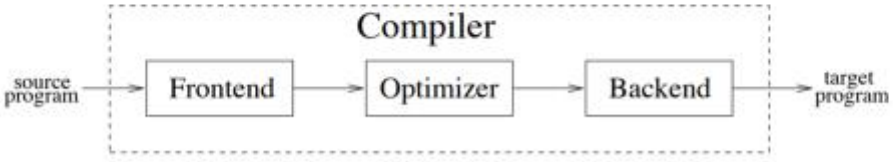
Компілятор

- Компілятор перетворює програму з початкової мови на цільову мову, зберігаючи семантику вихідної програми.

Приклад:

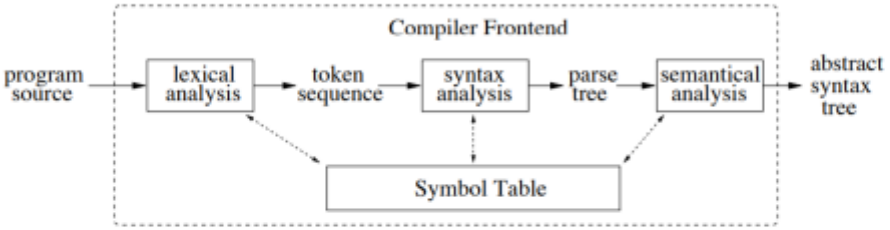
- High level language ⇒ Machine Code (e.g. C++ ⇒ Sparc Code)
- High level language ⇒ Intermediate Code (e.g. Java ⇒ Byte Code)
- High level language ⇒ High level language (e.g. C ⇒ Java)
- Format ⇒ Format (e.g. PDF ⇒ Post Script)
- Всі перераховані вище трансляції базуються на загальних принципах - принципах компілятора.

Огляд компілятора



- Frontend: Перевіряє, чи правильна початкова програма ⇒ згідно з певною специфікацією мови.
- Оптимізатор: програма перезаписує для підвищення продуктивності цільового коду.
- Backend: Переводить програму до цільового коду.

Огляд інтерфейсу



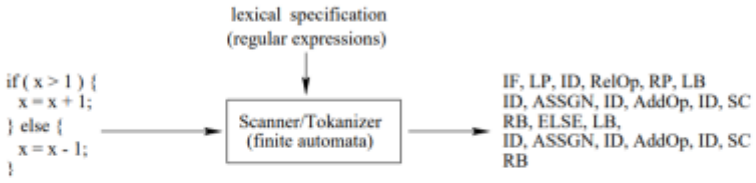
- Лексичний аналіз: визначити конструкції атомної мови. Кожен тип конструкції представлений маркером. (e.g. 3.14 → FLOAT, if → IF, a → ID).
- Аналіз синтаксису: перевіряє правильність послідовності токенів з урахуванням до специфікації мови.
- Семантичний аналіз: перевіряє правила типу відносини + узгодженість. (e.g. if type(lhs) = type(rhs) in an assignment lhs = rhs).
- Кожен крок передбачає перетворення з одного програмного представлення в інше.

Формальне визначення мови програмування

Program	→	*ClassDecl* +
ClassDecl	→	*PUBLIC CLASS Id LB* (*FieldDecl* | *MethodDecl*) * *RB*
MethodDecl	→	*PUBLIC Type Id ParamList LB Statement* * *RB*
Statement	→	*ifStmt* | *whileStmt* | *forStmt* | *assignStmt*
whileStmt	→	*WHILE LP boolExp RP LB Statement* * *RB*
ifStmt	→	...
...	→	...

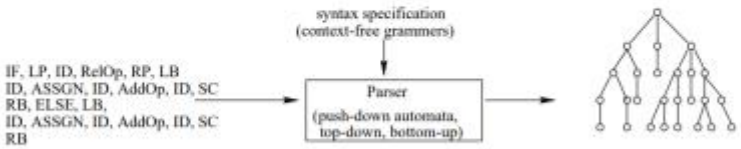
- Ліву сторону можна розглядати як назву даної мовної конструкції. Права сторона як її визначення.
- PUBLIC, CLASS, LB, RB, LP, RP, WHILE є токенами.
- Цей тип визначень називають контекстно-вільними граматиками.

Лексичний аналіз



- Вхідне представлення програми: Послідовність символів
- Представлення вихідної п
- Специфікація аналізу: регулярний вираз для кожного маркерарограми: послідовність маркерів
- Реалізація: Детерміністичні кінцеві автомати

Синтаксичний аналіз



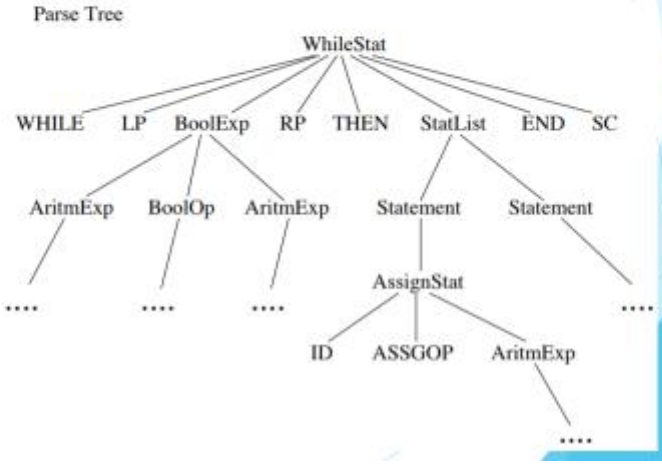
- Представлення вхідної програми: послідовність токенів
- Представлення вихідної програми: Розбір (або синтаксис) дерева
- Специфікація аналізу: граматика без контексту
- Реалізація: парсери зверху вниз або знизу вгору

Розбір: послідовність токенів => Дерево розбору

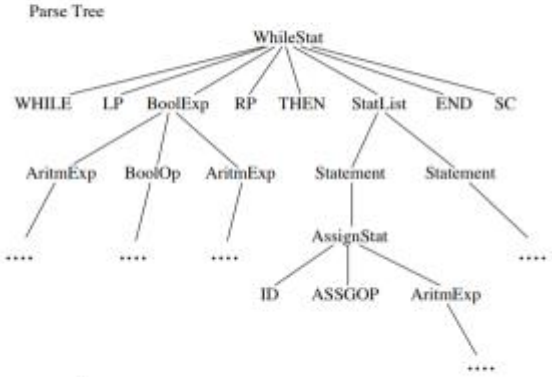
WhileStat --> <WHILE> <LP> BoolExp <RP> <THEN> StatList <END> <SC>

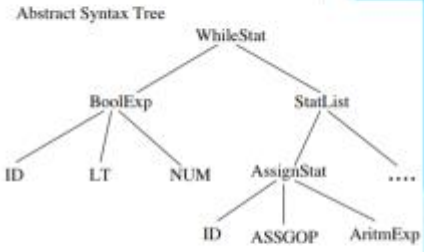
```
Token Sequence
==============

WHILE LP ID(#1)
LT NUM(#3) RP THEN
ID(#1) ASSGNOP ID(#1)
PLUS NUM(#4) MULT
NUM(#5) SC END SC
```



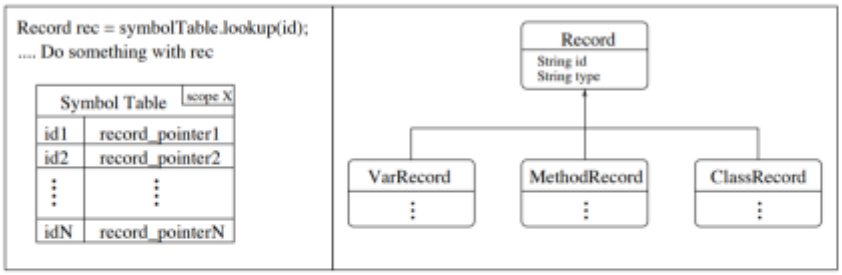
Дерево абстрактного синтаксису





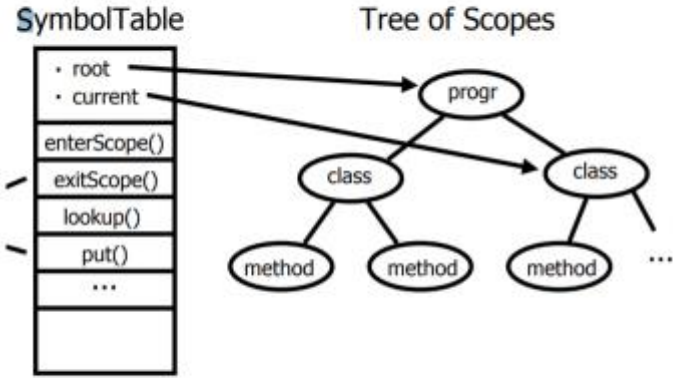
Дерево абстрактного синтаксису є конденсованим синтаксичним деревом, де є лише найважливішу інформацію. Воно, разом з таблицею символів, є кінцевим продуктом фази аналізу.

Таблиця символів



- Таблиця символів - це структура даних, що зберігає інформацію про ідентифікатори
- Вона побудована в один обхід синтаксичного дерева після завершення розбору.
- Кожен ідентифікатор представлений записом, де зберігається інформація.
- Таблиця символів використовується, коли ми хочемо знайти ідентифікатор.

Конструювання таблиці символів



- Область - це внутрішній клас, що формує дерево областей. Вона використовується внутрішнім класом SymbolTable для організації вмісту в областях.

Лексичний аналіз ⇒ Послідовність токенів

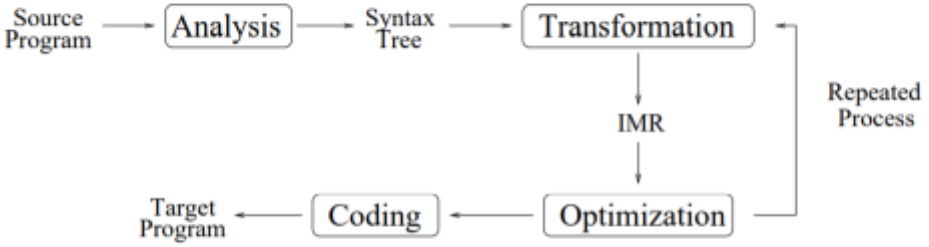
```
Source
======

int I;
I := 0;
while (I < 100) then
  I := I + 3 * 4.0;
end;
```

Symbol and Constant Table

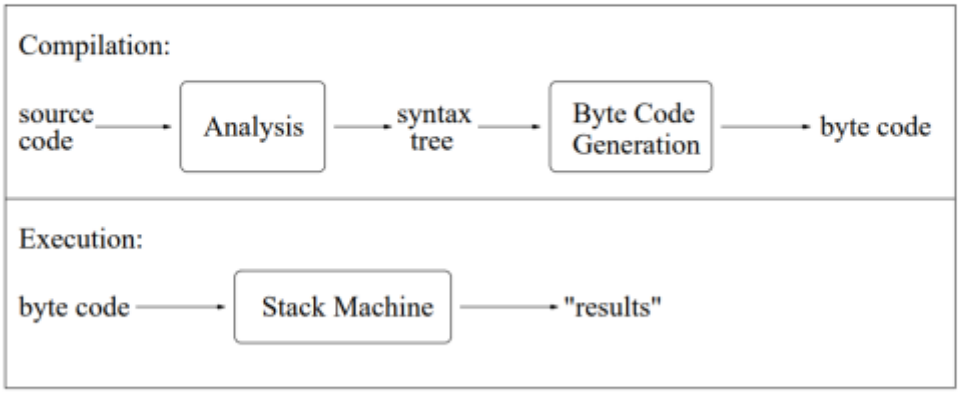
No	Token	Lexeme	Type
1	ID	‘‘I’’	INT
2	NUM	‘‘0’’	INT
3	NUM	‘‘100’’	INT
4	NUM	‘‘3’’	INT
5	NUM	‘‘4.0’’	FLOAT

Виконання стек машиною



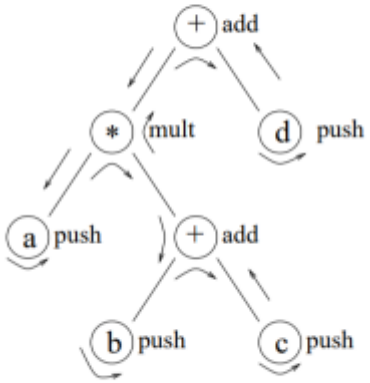
- Проміжне представництво - являє собою незалежне представлення машин і мови джерела, яке може розглядатися як з'єднання між переднім і зворотним кінцями компілятора.

Виконання кода MiniJava



- Використовуємо програму (машину стека) для виконання сформованого коду.

Генерація постфіксного представления



Generated Instructions

1	push a
2	push b
3	push c
4	add
5	mul
6	push d
7	add

- Постфіксне представлення арифметичних виразів може бути згенеровано за допомогою обходу синтаксичного дерева з глибиною першого порядку. Приклад: a * (b + c) + d

Робота з змінними

Instructions

1	lvalue #1
2	rvalue #2
3	rvalue #1
4	sub
5	assign

Variables

#	id	value
1	a	25
2	b	10

Description

lvalue #n = push index n on stack

rvalue #n = push value of variable n on stack

sub = pop v1, pop v2, push (v2 - v1)

assign = pop value v, pop index n, assign variable n value v

- При інтерпретації виразів за допомогою змінних ми повинні пам'ятати поточне значення, що зберігається в цій змінній.
- Нам потрібна нова структура даних для збереження поточного значення всіх змінних.

Виклик методів

```
public int mA() {
    ...
  a = mB(7);
    ...
}

public int mB(int b) {
    ...
  return 49;
}
```

Method mA()

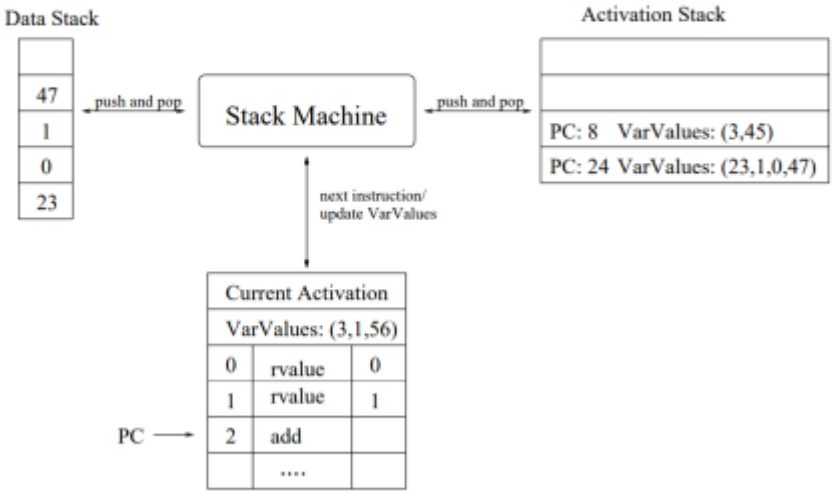
⋮	
9	push 7
10	call mB
11	assign #a
⋮	

Method mB()

1	assign #b
⋮	
24	push 49

- Кожен метод має окремий набір інструкцій. Виклик методу передбачає перехід від одного набору інструкцій до іншого.

Сумарна машина стека



Кожен метод має набір інструкцій та список змінних

Стек даних для перенесення даних між виконанням різних інструкцій.

Стек активації для зберігання станів активних викликів методів.

Висновки

- Метою атестаційної роботи була аналіз і вивчення методів конструювання компіляторів і розробка компілятора для мови програмування
- Досліджено сучасні тенденції інструментів програмування, орієнтованих на реалізацію початкових фаз трансляції, таких як синтаксичний і семантичний аналіз, що враховують особливості синтаксису і семантики вхідної мови.
- Спроектована загальна структура проміжного представлення програм на miniJava, що включає дерево програми, семантичні таблиці та подання системи типів.
- Реалізована система семантичних таблиць, використовувана для аналізу, зберігання і подальшої обробки семантичної інформації про програмні сутності miniJava.
- Розроблено загальну структуру компілятора для Java.
- Розроблено та реалізовано компоненти компілятора, що виконують лексичний, синтаксичний і семантичний аналіз.

Дякую за увагу.