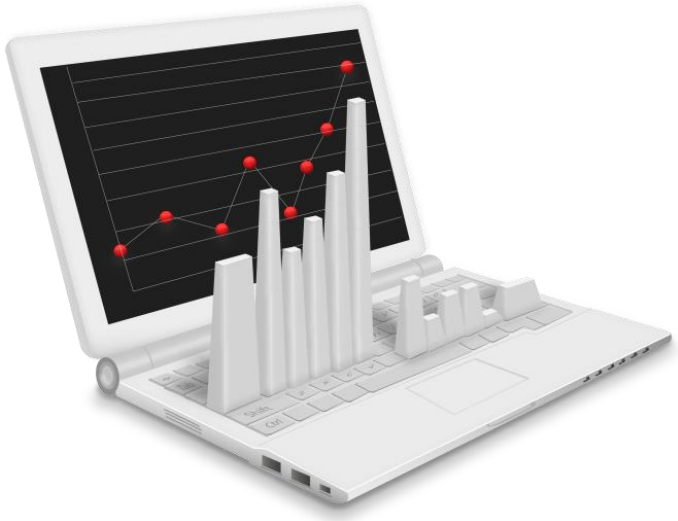
프로그래밍 언어론

Syntax와 Semantics

컴퓨터공학과

조은선



충남대학교
CHUNGNAM NATIONAL UNIVERSITY

학습목표

프로그래밍 언어의 두 가지 측면과 그 중 syntax(구문)를 명세하는 법을 배우고 실제 프로그램이 syntax를 만족하는지를 알아내는 방법에 대해 개념적으로 이해한다.

Syntax와 Semantics

학습내용

- 프로그래밍 언어의 두 가지 측면, syntax에 대한 정의
- derivation, parse tree에 대한 기본 개념

목 차

- 들어 가기
- 학습하기
  - 프로그래밍 언어의 구성: Syntax and Semantics
  - 프로그램의 syntax를 이해하는 방법 Syntax를 표현하기
  - 주어진 문자열이 Syntax에 맞는지 확인하는 방법
- 평가하기
- 정리하기

알고가기

Pretest

다음 C 프로그램 예문 중에서 오류가 없는 것을 고르시오.

1) int i; i = 0;

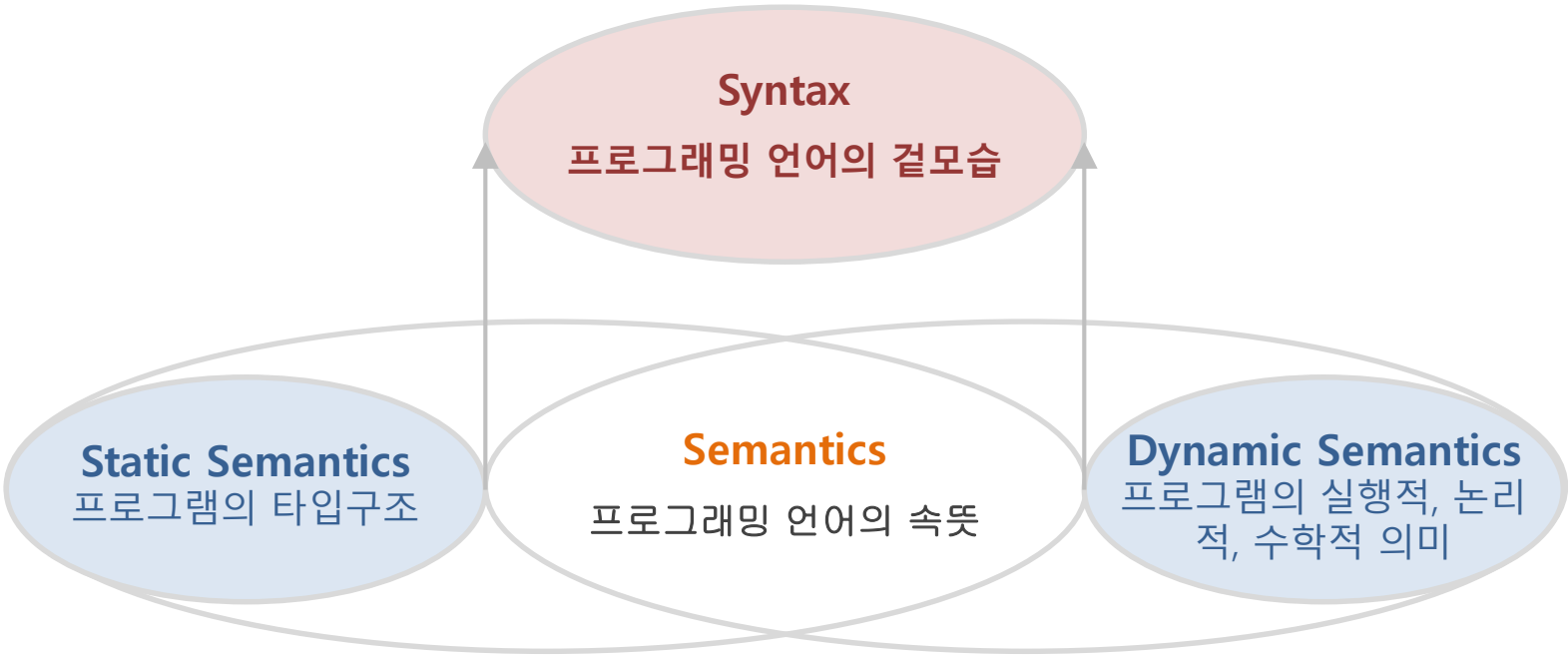
2) if (i > 0) i + +;

3) while (i <> 0) i++;

4) i := i + 3;

Syntax(구문)와 Semantics(의미구조)

[프로그래밍 언어의 두 가지 면]



이 과목 전체에서 다양한 언어의 Dynamic Semantics (실행적 의미구조)를 서술적으로 설명하고 있음

Ⅰ Compiler(컴파일러)와 Interpreter(인터프리터)

컴퓨터의 프로그램 이해과정

- 컴퓨터가 프로그램의 syntax를 이해해서 semantics에 맞게 수행할 수 있도록 하는 방법 두 가지

1. Compiler (컴파일러)
   프로그램 전체를 읽어서 기계어 등으로 번역해 놓고 수행함
2. Interpreter (인터프리터)
   프로그램을 한 문장씩 읽어서, 이해할 때 마다 하나씩 수행하기

Compiler는 프로그램 전체를 이해할 수 있으므로 최적화 등이 가능하다. 그러나 절차가 더 무겁다. (Interpreter는 반대)

Compiler 와 Interpretation의 비교 예

"냄비에 물을 550cc 붓는다. 불을 켠다. 끓을 때까지 기다린다. 사리와 스프를 넣는다. 3분 기다린다. 불을 끈다. 먹는다."

- Compiler는 전부 읽고 머릿속에 넣어서 "냄비에 물을~ "부터 "먹는다"까지 수행한다.
- Interpreter는 한줄씩 읽고 수행한다. 즉, " 냄비에 ~"를 읽고 그대로 하고, "불을 켠다"를 읽고 그대로 한다.

Syntax(구문)

Syntax

- 문자의 "적절한" 조합으로 이루어졌다는 것을 빼면, 프로그래밍 언어마다 다름

  예) 같은 일을 하는 문장들

```
int a;
Dim a As Integer
type a = integer;
a : Integer;
```

- 구성
  - 먼저 '단어'가 있고 : 예) `int, a, =, 0` 등
  - 그 다음 단어를 적당히 구성한 문장으로 이루어져있다.

    예) `int a = 0;`

>> 위로부터 C, Visual Basic, Pascal, Ada 임

Ⅰ Syntax의 표현 방법

BNF(Backus-Naur Form)

- 프로그래밍 언어의 syntax를 표현하기 위한 표기법으로 규칙의 집합으로 표현
  - 규칙 하나의 모양 : "새로운 단어-> 설명"
    예) <A>-><B> c <D>
  - 이 때 새로운 단어 부분에 나타나는 symbol (예에서 <A>)을 "non-terminal"이라고 한다.
  - 설명 부분에 나타나는 것은 symbol의 리스트이다. (예에서 <B>c<D>) non-terminal일 수도 있으나 (예에서 <B>, <D>) terminal symbol 이 올 수 있다. (예에서 c)
    - terminal symbol이란 숫자나 문자, 또는 특정 문자열처럼 뜻이 자명하고 더 이상 쪼개지지 않는 단어들을 의미한다.

- 다양한 변형이 존재함
  (1) -> 대신 ::= 를 쓸 수도 있다. non-terminal symbol은 < >로 묶어 구별하거나, 대문자만으로 표시를 하기도 한다.
  (2) ->의 왼쪽의 non-terminal이 동일한 규칙은 "|" 기호를 써서 묶어 나타낼 수 있다.
  예) <A> -> <B>c<D> 와 <A> -> <B> 가 syntax에 동시에 존재할 때
  <A>-><B>c<D> | <B> 로 나타낼 수 있다.

BNF 구문 정의의 예

Assignment

```
<assign>  ->  <id> = <expr>
<id>      ->  A | B | C
<expr>    ->  <id> + <expr>
              | <id > * <expr>
              | (<expr>)
              | <id>
```

간단한 예제
프로그래밍 언어
Syntax

```
<program> -> <stmts>
<stmts>   -> <stmt> | <stmt> ;<stmts>
<stmt>    -> <var> = <expr>
<var>     -> a | b | c | d
<expr>    -> <term> + <term> | <term>
<term>    -> <var> | const
```

Lexical Analysis(어휘분석)과 Parsing(파싱)

[compiler나 interpreter 단위가 다를 뿐 모두 필요한, syntax와 관련된 작업 두 가지]

Lexical analysis

- 어휘분석: 단어를 식별하는 작업
  - int, return, while, x ....
  - 프로그램 언어에서 이런 단어를 token이라고 한다.

Parsing

- lexical analysis 결과로 주어진 token열이 내포하는 문법적 구조를 알아내는 절차
  - 결과로 parsing tree(파싱 트리)를 생성함

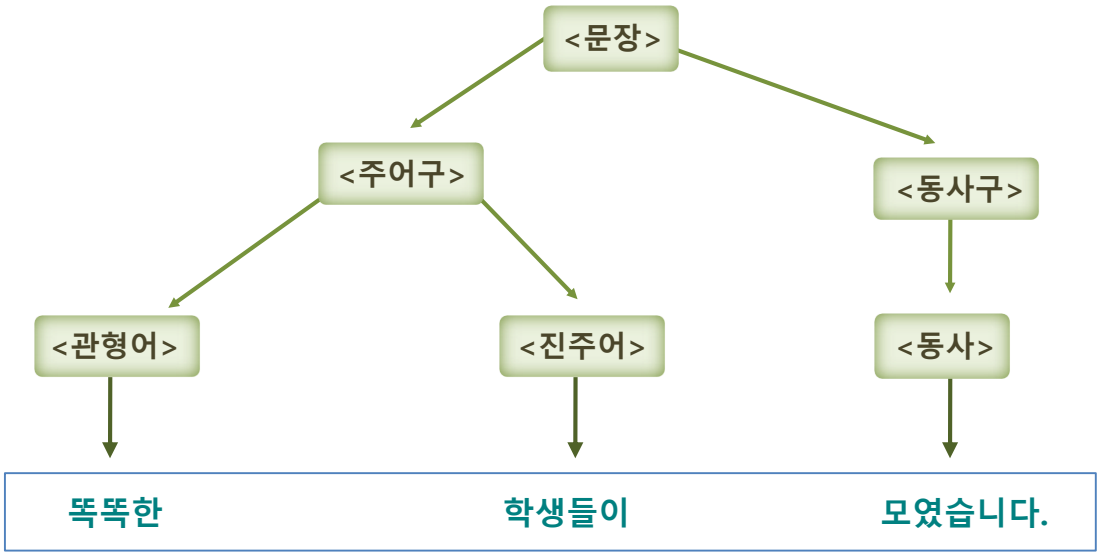
Parsing의 예

우리말 예 }

▶ grammar가 맞는지 확인하고 각 단어의 역할을 확인한다.

<문장> → <주어구> <동사구>
<주어구> → <관형어> <진주어>
<진주어> → "학생들이" |"친구들이" | "3학년이"
<관형어> → "똑똑한" | "게으른" | "착한"
<동사구> → <부사> <동사> | <동사>
<부사> → "재빨리" | "열심히"
<동사> → "모였습니다" | "공부합니다"

"똑똑한 학생들이 모였습니다."



Derivation(유도)

Derivation(유도)란?

- 주어진 token 열이 syntax에 맞는지 확인하고 token의 역할을 파악하는 절차
- grammar 의 최상위 non-terminal symbol 로 부터 grammar 의 규칙들을 반복적으로 적용해서 terminal symbol을 만들어냄

(앞의 문법에 맞는 프로그램의 예)

a = a + b ; b = c

<program> => <stmts>

=> <stmt> ; <stmts>	=> <var> = <expr> ; <stmts>
=> a = <expr> ; <stmts>	=> a = <term> + <term> ; <stmts>
=> a = <var> + <term> ; <stmts>	=> a = a + <term> ; <stmts>
=> a = a + <var> ; <stmts>	=> a = a + b ; <stmts>
=> a = a + b ; <stmt>	=> a = a + b ; <var> = <expr>
=> a = a + b ; b = <expr>	=> a = a + b ; b = <term>
=> a = a + b ; b = <var>	=> a = a + b ; b = c

- left(right)most derivation (좌측유도)

  : derivation 단계마다 맨 왼쪽(오른쪽)에 위치한 non-terminal 을 대치한다.

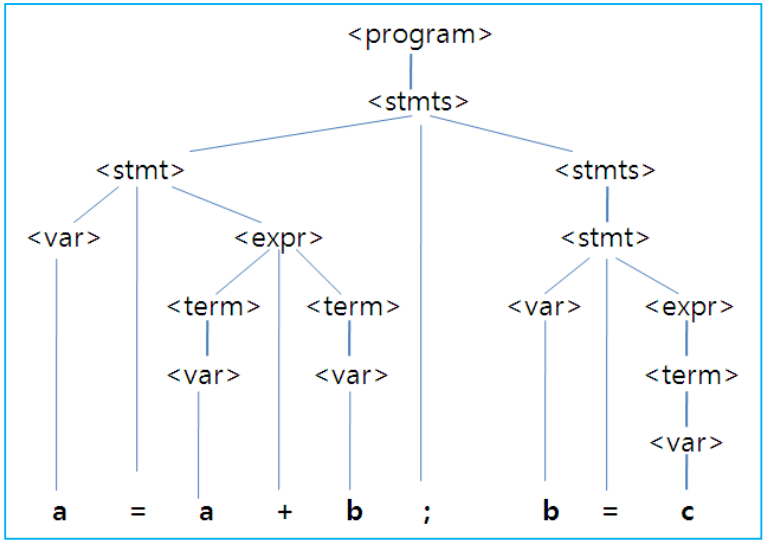
Parsing Tree(파싱 트리)

a = a + b ; b = c

- Parsing Tree (또는 parse tree)
  - derivation 과정을 그림으로 표현한 것
  - 주어진 문장(들)의 syntax 구조를 나타내는 tree
    - 프로그램의 구조를 이해했다 : parsing tree를 만들었다는 뜻

- 앞의 grammar 에 맞는 프로그램의 예

  - left most derivation 이나 right most derivation이나 parsing tree 는 동일하게 생성된다.



평가하기

마지막으로 내가 얼마나 이해했는지를 한번 확인해 볼까요?

총 3문제가 있습니다.

START

평가하기 1

1. 다음 중 어떤 프로그래밍 언어의 syntax적인 설명이 아닌 것은?

① 변수 이름은 반드시 문자로 시작해야 한다.

② 정수 덧셈 연산자는 피연산자의 두 정수를 더한다.

③ If 문에서 else 부분은 나올 수도 있고 없을 수도 있다.

④ 각 문장 뒤에는 반드시 세미콜론이 나와야 한다.

확인

평가하기 2

2. 다음 중 compiler와 interpreter의 설명으로 적당하지 않은 것은?

① 둘 다 프로그램의 syntax를 이해하는 절차를 포함한다.

② interpreter는 한문장씩 수행한다.

③ compiler는 프로그램 전체를 읽어 번역한 후 수행한다.

④ interpreter 는 compiler에 비해 최적화가 용이하다.

확인

평가하기 3

3. grammar 의 최상위 non-terminal symbol 로 부터 grammar 의 규칙들을 반복적으로 적용해서 terminal symbol을 만들어 내는 과정은 무엇인가?

① lexical analysis

② BNF

③ token

④ derivation

확인

정리하기

Syntax and Semantics

프로그래밍 언어는 두 가지 면, 구문구조와 의미구조를 가진다.

Syntax

• 더 이상 쪼개지면 의미를 잃어버리는, 기본이 되는 단어는 token이라 부른다.
특정 프로그래밍 언어가 가지는 문장의 syntax는 BNF 와 같은 방식으로 기술된다.
이것을 syntax를 grammar로 표현한다고 한다.

•어떤 token 열이 해당 syntax를 표현한 grammar에 의해서 parse tree나 derivation을 생성할 수 있으면 그 token 열은 그 syntax를 따른다고 한다.