

제4장 변수 및 유효범위

숙명여대 창병모

4.1 변수 선언

변수 선언과 유효범위

- 변수 선언
 - Declaration before Use !
 - 대부분의 언어에서 변수는 사용 전에 먼저 선언해야 한다.
- 변수의 유효범위(scope) 판정영역
 - 선언된 변수가 유효한(사용될 수 있는) 프로그램 내의 범위/영역
 - 변수 이름뿐 아니라 함수 등 다른 이름도 생각해야 한다.
- 정적 유효범위(Static scope) 규칙
 - 선언된 이름은 선언된 블록 내에서만 유효함
 - 대부분 언어에서 표준 규칙으로 사용됨

변수 선언

구문법

<stmt> → ...

| id = <expr>;

| let <decls> in

<stmts>

end;

optional

<decls> → {<type> id [= <expr>];}

<stmts> → {<stmt>}

초기화

<type> → int | bool | string

의미

- 변수 id는 <type>타입 변수이며 초기화가 가능하다.
- ✓ 초기화하지 않는 변수는 자동으로 기본값(0, "", false)으로 초기화한다.
- 변수 id는 지역 변수로 유효범위는 선언된 블록 내이다.

변수 선언 및 유효범위: 예

- Example in S

```
let int x; in  
  x = 1;  
  x = x + 1;  
end;
```

- Example in C

```
{ int x;  
  x = 1;  
  x = x + 1;  
}
```

- 정적 유효범위(Static scope) 규칙
 - 선언된 이름은 선언된 블록 내에서만 유효함

블록의 중첩 ; nesting 허용

- 블록의 중첩

<stmt> → ...
| let <decls> in
 <stmts> → 이 자리에 다시 let statement가 오는 경우
end;

선언이 여러개 가능하도록

- let 블록 내의 문장 S에 다시 let 블록이 나타날 수 있다.

let D1; in
 let D2; in
 ...
end;
end;



let D1;D2; in
 ...
end;

↳ 더 편리함

중첩 블록

- [예제 2] in S

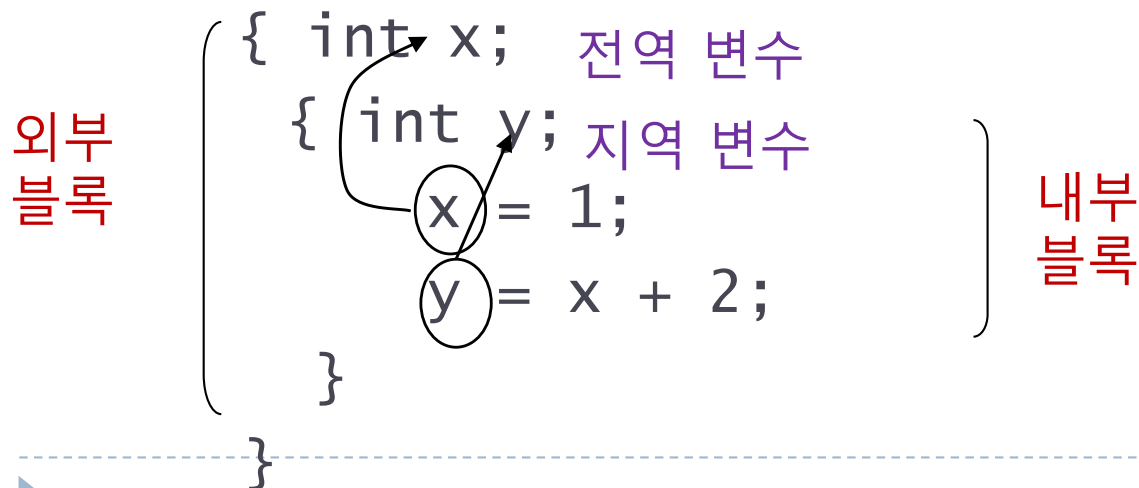
```
let int x; in
  let int y; in
    x = 1;
    y = x + 2;
  end;
end;
```



[예제 3]

```
let int x; int y; in
  x = 1;
  y = x + 1;
end;
```

- [예제 2] in C



같은 이름의 변수 선언과 유효범위

- 같은 이름의 변수가 여러 개 선언되었을 때 유효범위

- [예제 4] in S

```
let int x = 1; in → global
|
|   let int y = 0; in
|       y = x + 2;
|   end;
|   let int x = 5; in → local
|       x = x + 1;
|   end;
|   x = x * 2;
end;
```

block 내에서는 local을 우선함

→ global

- [예제 4] in C

```
{ int x = 1;
  { int y = 0;
    y = x + 2;
  }
  { int x = 5;
    x = x + 1;
  }
  x = x * 2;
}
```


언어 S의 전역 변수

- 지역 변수

- let 문 내에서 선언된 변수는 지역변수

- 전역 변수

- let 문 밖에서 선언된 변수는 전역 변수(global variable)

<command> → <decl> | <stmt> | <function>

<decl> → <type> id [= <expr>];

- 예) command의 연동

>> int x=1; → global

>> let int y=2; in
 x = x+y;
end; → local

>> print x;

python은 비슷하지만 타입선언 x
→ 임의로 사용

언어 S의 전역 변수

- 전역 변수와 같은 이름의 지역 변수 선언

```
>> int x=1;  
>> let int x=2; in  
    ( print x;  
      end;
```

2 → local

```
>> print x;
```

1 → global

- 함수를 정의한 후 호출해서 사용

```
>> fun int f(int x) return x*x; → f라는 이름이 필요해짐 (범위; global)
```

```
>> print f(10);
```

100

타입 없는 변수 선언

- 동적 타입 언어(dynamically typed language)
 - 변수의 타입을 선언하지 않고 바로 사용
 - 변수에 어떤 타입의 값이든지 저장 가능
 - Lisp/Scheme, JavaScript, Python 등
- Python 예
 - 대입문을 사용하여 변수에 대입하면 변수는 자동으로 생성된다.
 - `id = <expr>`
 - ```
>>> score = 80
```

 Assign → 자동생성
  - ```
>>> print(score)
```
 - 80
 - ```
>>> score = "80 %"
```
  - ```
>>> print(score)
```
 - 80 %

Python 전역 변수

[예제 5]

```
>>> percent = 20 → global
```

```
>>> def salePrice(price): (할인가 계산해주는 함수)
```

```
    result = price * (1 - percent/100) → local  
    return result
```

```
>>> salePrice(48000)
```

```
38400.0
```

● 주의

- 함수 내에서 전역 변수 사용은 가능하나

전역 변수 수정은 불가능 하다. (assign 불가)

Why? 전역 변수에 대입하면 자동으로 지역 변수가 생성된다.

: 같은 이름의 전역/지역 변수를 쓸 수 있게 하기 위함

Python 전역 변수

[예제 6]

```
>>> percent = 20
>>> def salePrice(price):
    percent = percent + 10 ❌ error!
    result = price * (1 - percent/100)
    return result
```

local 변수가
assign 전에 사용됨

```
>>> salePrice(48000)
```

File "<pyshell#6>", line 2, in salePrice

```
    percent = percent + 10
```

UnboundLocalError: local variable
'percent' referenced before assignment

[예제 7]

```
>>> percent = 20
>>> def salePrice(price):
    global percent
    percent = percent + 10
    result = price * (1 - percent/100)
    return result
```

사용 가능!

```
>>> salePrice(48000)
```

할인 가격: 33600.0

```
>>> print(percent)
```

30

4.2 블록 구조 언어

중첩어용

블록

- 블록

- 서로 연관된 선언문과 실행문들을 묶어놓은 프로그래밍 단위
- 블록은 변수나 함수를 선언하는 선언문들과 실행문들로 구성됨.

- 블록을 나타내는 기호

- 중괄호({}) : C
- begin-end : Ada
- 프로시저 또는 함수: C, Pascal 등

블록

- Pascal
 - 프로시저 혹은 함수가 하나의 블록
 - 프로시저(함수) 내에 변수 선언 및 실행 문장
- C, C++, Java
 - { ... }
 - 괄호 내에 변수 선언 및 실행 문장
- Ada
 - declare
... 선언
begin 실행 end
- ML, S
 - let 선언 in 실행 end
왜 (라아나는 용어)?
수학적 표현은 그대로 옮겨놓음

블록 구조 언어

- 블록의 중첩을 허용하는 언어

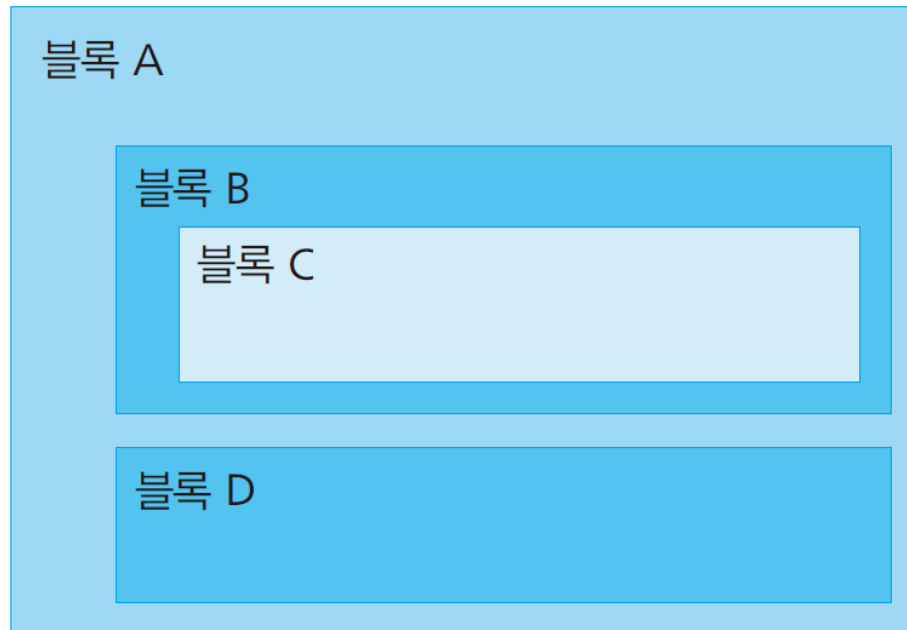


그림 4.1 블록 구조

- Algol, Pascal, Modula, Ada, C, S, ...

블록 구조 언어의 특징 및 장점

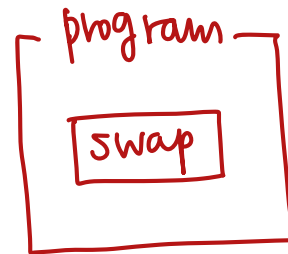
- (1) 대형 프로그램을 여러 블록으로 나누어 작성하면
복잡한 수행 내용을 단순화하며 프로그램의 해독성을 높여준다.
- (2) 프로그램 오류가 발생하여도 그 범위가 블록단위로 한정되므로
수정이 쉬워지며 블록의 첨가, 삭제, 수정 등이 용이하다.
- (3) 블록 내에 선언된 변수들은 그 안에서만 유효하며
실행이 종료되면 기존에 선언되었던 변수들은 모두 무효화 된다.
- (4) 사용자로 하여금 변수의 사용과 기억장소의 할당에 관한 경계를
명확하게 할 수 있다.

Pascal [예제 8]

가장 바깥 block

```
program ex(output);  
  var x, y : integer;           // 전역 변수  
  procedure swap(var A, B : integer);  
    var temp : integer ;       // 지역 변수  
  begin  
    temp := A;  
    A:=B;  
    B:=temp;  
  end;  
begin  
  x := 5; y := -3;  
  swap (x,y);  
end.
```

→ period로 끝내기



C 언어

C언어가장 바깥블록은? ⇒ file

- C 프로그램 구조

변수 선언;

리턴타입 함수명(매개변수) {
 변수 선언;
 실행문;
}

[예제 9]

```
int x=1;
```

```
int f(int x) { return x*x; }
```

```
int main() {  
    int y = 2;  
    x = f(y);  
    printf("%d %d\n", x, y);  
}
```

main도 하나의
block일뿐

→ 제일먼저 실행하는 함수(block)

C 언어의 유효범위 규칙

- 핵심 아이디어
 - 사용 전 선언(Declaration before use)
 - 선언의 유효범위는 선언된 지점부터 선언된 블록 끝까지
- 지역 변수의 유효 범위
 - 선언된 지점부터 함수 끝까지
- 전역 변수의 유효범위
 - 선언된 지점부터 파일 끝까지


C 언어 [예제 10]

```
int x = 1; // 전역 변수 선언
void p( ) // 함수 p 정의
{ char x; // 지역 변수 x 선언
  x = 'A'; // 지역 변수 x 사용
}
void q( ) // 함수 q 정의
{ double y = 2; // 지역 변수 y 선언
  { int z = 3; // 중첩 블록에서 지역 변수 z 선언
    x = y + z; // 전역 변수 x 사용
  }
  ...
}
main() // main 함수 정의
{ int w[10]; // 지역 변수 선언
  x = x + 1; // 전역 변수 사용
  ...
}
```

* C는 함수내에서 다른 함수 정의 불가!

Ada 블록

- Ada 블록 구조

 declare
 변수 선언
begin
 실행문
end

[예제 11]

```
declare
    x: integer;
    y: boolean;
begin
    x := 2;
    y := true;
    x := x+1;
    ...
end;
```

Ada 언어의 유효범위 규칙

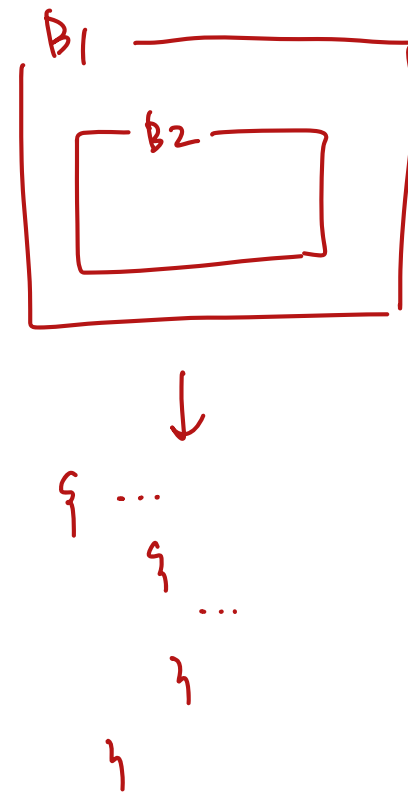
- 핵심 아이디어
 - 선언의 유효범위는 선언된 블록 내

- [예제 12]

```
B1: declare
    x: integer;
    y: boolean;
begin
    x := 2;
    y := false;

    B2: declare
        a, b: integer;
        begin
            if y then a := x;
            else b := x;
            end if
        end B2;
    ...
end B1;
```


중첩



ML 블록

- ML, S 블록 구조

let 변수 선언
함수 선언
in
실행문
end;



[예제 13]

```
let
  val x = 1
  fun f(x:int) = x + x 행복정의
in
  f(x+1)
end;
```

⇒ 변수의 유효범위를 block 단위로 살펴봄

Semantics

4.3 변수의 상태와 의미

변수의 의미

- 변수

- 메모리 위치(주소)를 나타내는 이름

- 대입문의 예

- $id = E;$
- 예를 들어 $\overset{\text{위치}}{\textcircled{x}} = \overset{\text{값}}{\textcircled{x}} + 1;$
- 오른쪽 x 와 왼쪽 x 의 의미가 다르다 !

파싱기의 assign :=

- 변수 x 의 의미

- 오른쪽 변수 x 의 의미 : 메모리 위치에 저장된 값(r-value)
- 왼쪽 변수 x 의 의미 : 메모리 위치(l-value)

변수 및 상태

- 수식 $23 + 5$ vs. 수식 $x + y$
- 수식 $x + y$ 의미는 ? $V(x + y)$
- 수식 $x + y$ 의 값은 변수 x, y 의 현재 값에 따라 다르다.
- 변수들의 현재 값을 무엇이라고 할까요?
- 상태(state)

기초 지식

- 함수집합

$$A \rightarrow B = \{f \mid f : A \rightarrow B\}$$

- 함수 $f : A \rightarrow B$

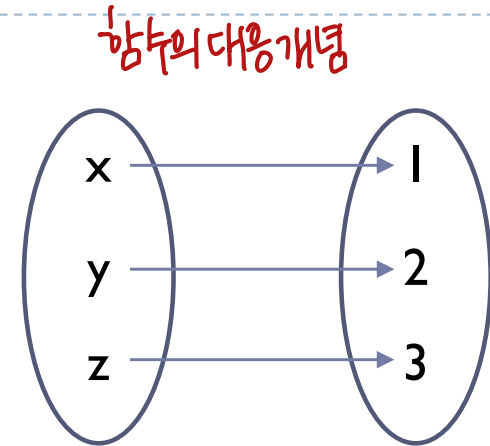
$$\{a_1 \mapsto b_1, a_2 \mapsto b_2, \dots, a_n \mapsto b_n\}$$

- 함수 수정 $f[a \mapsto b]$

$$f[a \mapsto b](x) = \begin{cases} b & \text{if } x = a \\ f(x) & \text{otherwise} \end{cases}$$

상태(State)

- 상태(A state)
 - 변수들의 현재 값
 - 하나의 함수로 생각할 수 있다.
 - $s : \text{Identifier} \rightarrow \text{Value}$
 - 예 $s = \{x \mapsto 1, y \mapsto 2, z \mapsto 3\}$



- 모든 가능한 상태들의 집합
 - $\text{State} = \text{Identifier} \rightarrow \text{Value}$
- 상태 s 에서 변수 값
 - $s(x)$ 변수가 가지고 있는 값

- 상태 갱신: $s' = s[y \mapsto v]$
$$s'(x) = \begin{cases} s(x) & \text{if } x \neq y \\ v & \text{if } x = y \end{cases}$$

ex) $s' = s[y \mapsto 5]$
y에 5를 assign ($y=5$)

수식의 의미

- 수식 E의 의미

상태에서 수식의 값

$V : (\text{State}, \text{Expr}) \rightarrow \text{Value}$: 수식의 값을 계산하는 함수

- 예

- $s = \{x \mapsto 1, y \mapsto 2\}$
- $V(s, x+y) = 3$

- 상태 s에서 간단한 수식의 의미

- $E \rightarrow \text{true} \mid \text{false} \mid n \mid \text{str} \mid \text{id}$

- $V(s, \text{true}) = T$

- $V(s, \text{false}) = F$

- $V(s, n) = n$

- $V(s, \text{str}) = \text{str}$

- $V(s, \text{id}) = s(\text{id})$ \rightarrow 식별자에 대응되는 값

↓
string
리터럴

수식의 의미

- 산술 수식

- $E \rightarrow E + E \mid E - E \mid E * E \mid E / E$
- $V(s, E1 \text{ '+' } E2) = \underline{V(s, E1)} + \underline{V(s, E2)}$
- ...

- 비교 수식

- $E \rightarrow E > E \mid E < E \mid E == E \mid E != E$
- $V(s, E1 == E2) = \begin{matrix} T & \text{if } V(s, E1) == V(s, E2) \\ F & \text{otherwise} \end{matrix}$
- ...

변수의 상태 파악 \rightarrow 의미 \rightarrow 변수 계산

문장의 의미

→ 변화와 관련된 문장만 다룬다 (개체)
; 대입문/복합문

statement

- 문장 S의 의미

- 문장 S가 전상태 s를 후상태 s'으로 변경시킨다.
- 상태 변환

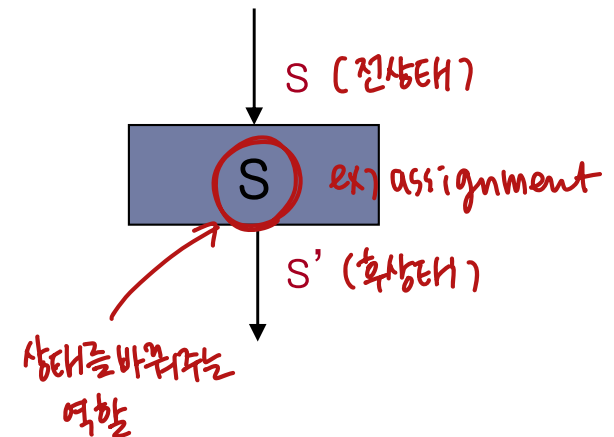
- 상태 변환 함수(state transform function)

- $\text{Eval} : (\text{State}, \text{Statement}) \rightarrow \text{State}$
- $\text{Eval}(s, S) = s'$ for each statement S

- 의미론

- 각 문장 S마다 상태 변환 함수 정의
- 프로그램의 실행과정을 상태 변환 과정으로 설명한다.

즉, 문장이 하는 일을 함수 형태로 정의한다

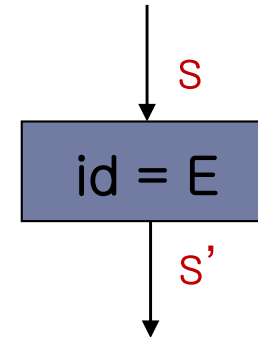


대입문, 복합문의 의미

- 대입문 $id = E$

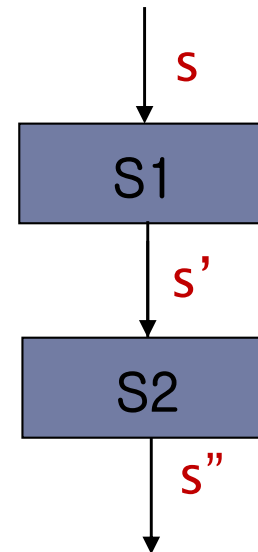
$$\text{Eval}(s, id = E) = s[id \mapsto V(s, E)]$$

id의 값이 E로 바뀜 (나머지 값은 그대로)



- 복합문 $S_1; S_2$

$$\text{Eval}(s, S_1; S_2) = \text{Eval}(\text{Eval}(s, S_1), S_2)$$



예

- (1) 실행

- $s = \{x \mapsto 0\}$
- $\text{Eval}(s, x = 1) = s[x \mapsto 1] = \{x \mapsto 1\}$

- (2) 실행

- $s = \{x \mapsto 1, y \mapsto 0\}$
- $\text{Eval}(s, y = 2) = s[y \mapsto 2] = \{x \mapsto 1, y \mapsto 2\}$

- (3) 실행

- $s = \{x \mapsto 1, y \mapsto 2\}$
- $\text{Eval}(s, x = x + y) = s[x \mapsto V(s, x + y)] = s[x \mapsto 3] = \{x \mapsto 3, y \mapsto 2\}$

```
let int x; in
  x = 1;      (1)
let int y; in
  y = 2;      (2)
  x = x + y;  (3)
end
end
```

4.4 변수의 유효범위 관리

블록 구조를 위한 상태 관리

- 블록 시작을 만났을 때 *exit let*
 - 블록 내에 선언된 변수는 유효해 진다.
 - 선언된 변수에 대한 상태 정보를 새로 생성한다.
- 블록 내 문장을 만났을 때 *exit assignment*
 - 유효한 변수들의 상태 정보를 이용해서 문장들을 해석(실행)한다.
- 블록 끝을 만났을 때 *exit end*
 - 블록 내의 선언된 변수들은 더 이상 유효하지 않음.
 - 블록 내의 선언된 변수들의 상태 정보를 제거한다.
- 상태(state)를 스택(stack) 형태로 유지 관리

[예제 16]

- 중첩된 블록에서 선언된 변수의 유효범위

- First-In Last-Out
- Last-In First-Out → stack
↳ LIFO

1 let int x; in

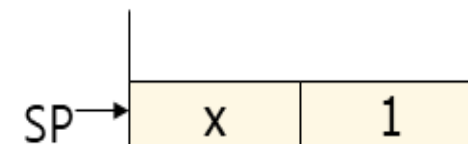
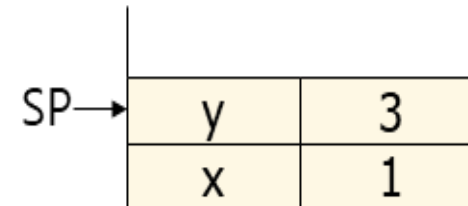
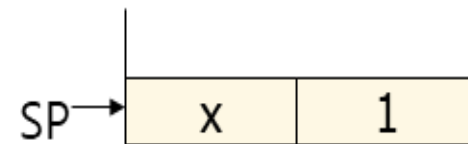
2 x = 1;

3 let int y; in

4 y = x+2;

5 end → 'y' pop

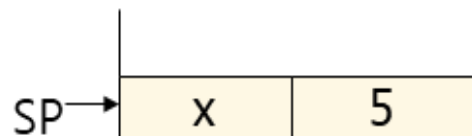
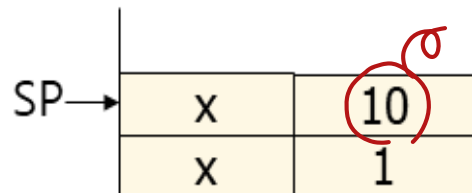
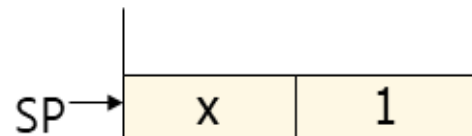
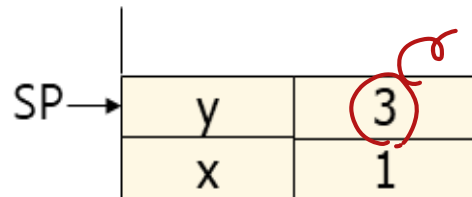
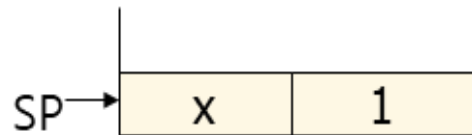
6 end → 'x' pop



유효한 변수의 상태정보만
Stack에서 유지 관리됨

[예제 17]

상태 관리



```
1 let int x = 1; in
2   let int y; in
3       y = x + 2;
4   end
5   let int x; in
6       x = 10;
7   end
8   x = x * 5;
9 end
```

4.5 구현

let 블록과 변수 선언

[변수 관련 언어 S의 문법]

$\langle \text{program} \rangle \rightarrow \{ \langle \text{command} \rangle \}$
 $\langle \text{command} \rangle \rightarrow \langle \text{decl} \rangle \mid \langle \text{stmt} \rangle \mid \langle \text{function} \rangle$
 $\langle \text{decl} \rangle \rightarrow \langle \text{type} \rangle \text{ id } [= \langle \text{expr} \rangle];$
 $\langle \text{stmt} \rangle \rightarrow \dots$
 $\mid \text{id} = \langle \text{expr} \rangle;$
 $\mid \text{let } \langle \text{decls} \rangle \text{ in } \langle \text{stmts} \rangle \text{ end};$
 $\langle \text{decls} \rangle \rightarrow \{ \langle \text{decl} \rangle \}$
 $\langle \text{stmts} \rangle \rightarrow \{ \langle \text{stmt} \rangle \}$
 $\langle \text{type} \rangle \rightarrow \text{int} \mid \text{bool} \mid \text{string}$

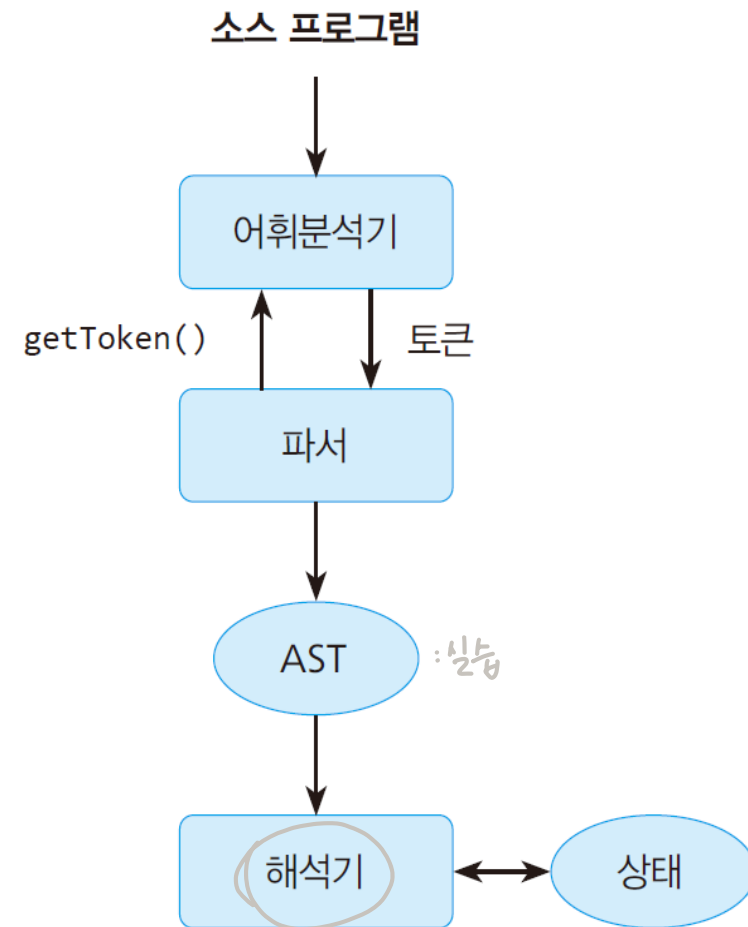


그림 3.1 어휘분석기, 파서, 해석기 구현

이것의 변수 관련 내용을 다루는 것!

상태 구현

- 상태는 변수와 그 값을 대응시키는 하나의 함수이다.

$s: \text{Identifier} \rightarrow \text{Value}$

- 상태 구현

- 변수의 값을 나타내는 **<변수 이름, 값>** 쌍들의 집합으로 표현

- 변수의 값을 **<id, val>** 쌍 형태로 정의.

```
class Pair {
```

```
    Identifier id;
```

```
    Value val;
```

```
    Pair (Identifier id, Value v) {
```

```
        this.id = id;
```

```
        this.val = v;
```

```
    }
```

```
}
```

AST 노드
(대문자로 시작)

integer / boolean / string 3종류

```
class Value {
```

```
    Object value;
```

```
    Type type; // 상속 받음
```

```
    ...
```

```
}
```

상태 구현

- **스택** 형태로 유지 관리

- 변수의 유효범위를 동적으로 관리하기 위함.
- <변수 이름, 값> 쌍들의 스택

```
class State extends Stack<Pair> {
```

```
    // id는 식별자로 변수 이름을 나타낸다.
```

```
    public State( )
```

```
    public State(Identifier id, Value val)
```

```
    public State push(Identifier id, Value val)
```

```
    public Pair pop ( )
```

```
    public int lookup(Identifier id) → 이름에 해당하는 쌍이 있는지 확인 : 위치한 인덱스를 리턴
```

```
    public State set(Identifier id, Value val) // 상태에서 변수 값 설정 : 변경된 스택을 리턴
```

```
    public Value get (Identifier id) // 상태에서 변수 값 조회
```

```
}
```

자바의 generic (A)

: 클래스를 템플릿으로 정의하는 것

→ pair 스택 형태로 나타냄

수식의 값 계산 = 수식의 의미

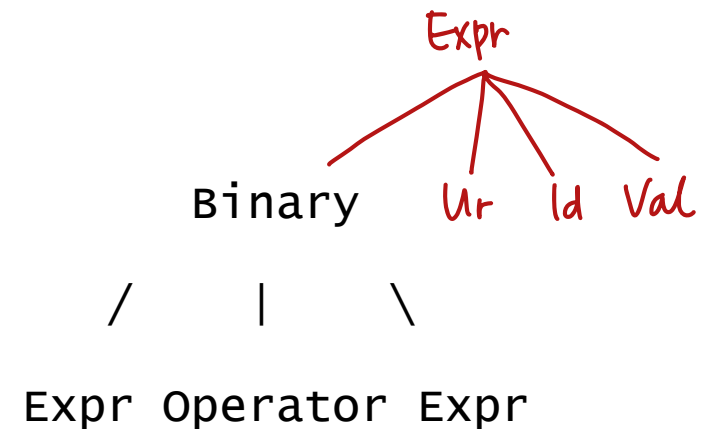
값을 리턴 \rightarrow 수식을 나타내는 AST (binary, unary, identifier, value)

```
Value V(Expr e, State state) {  
    if (e instanceof Value) return (Value) e;  
    if (e instanceof Identifier) {  
        Identifier v = (Identifier) e;  
        return (Value)(state.get(v));  
    }  
    if (e instanceof Binary) {  
        Binary b = (Binary) e;  
        Value v1 = V(b.expr1, state);  
        Value v2 = V(b.expr2, state);  
        return binaryOperation(b.op, v1, v2);  
    }  
    if (e instanceof Unary) {  
        Unary u = (Unary) e;  
        Value v = V(u.expr, state);  
        return unaryOperation(u.op, v);  
    }  
}
```

자바의 abstract class

: 다 구현되지 않아서, 상속받아
구현해야함

\rightarrow Expr은 abstract class 이므로
val, id, bin, ur 각각에 맞춰 다르게 구현



수식의 값 계산

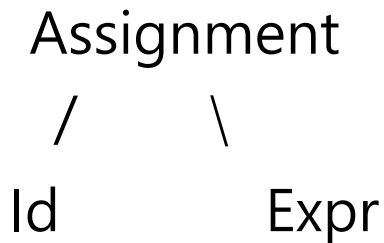
```
Value binaryOperation (Operator op, Value v1, Value v2) {  
    switch(op.val) {  
        case "+":  
            return new Value(v1.intValue( ) + v2.intValue( ));  
            ...  
        case "<":  
            return new Value(v1.intValue( ) < v2.intValue( ));  
            ...  
        case "==":  
            return new Value(v1.intValue( ) == v2.intValue( ));  
            ...  
        case "&":  
            return new Value(v1.boolValue( ) && v2.boolValue( ));  
    }  
}
```

대입문 실행 : 대입문, 라문

- 구문법

id = <expr>;

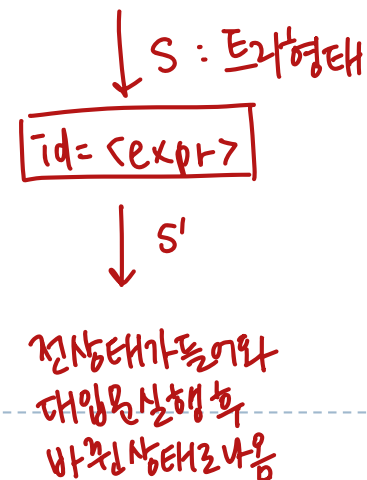
- AST



- 상태 변환 함수

```
State Eval(Assignment a, State state) {
    Value v = V(a.expr, state);
    return state.set(a.id, v);
}
```

값을 할당
stack 구조에 있던 메모리



let 문 실행

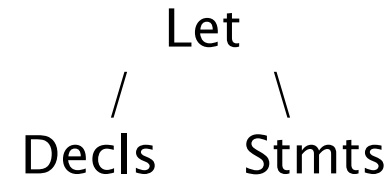
- 구문법

let <decls> in
 <stmts>

end

<decls> → {<type> id [=<expr>];}

<stmts> → {<stmt>}



- 상태 변환 함수

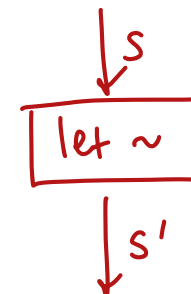
```
State Eval(Let l, State state) {  
    State s = allocate(l.decls, state);
```

③ = Eval(l.stmts, s); 실행해서 s의 상태가 바뀜

return free(l.decls, s);

}

stack에 let~end의
엔트리쌍을 만들어주는 것



: 전 상태(AST)가 들어옴

< ① 변수선언
 ② 실행

: 다음 상태(AST)를 반환함

팝의 기능

let 블록 실행

- allocate 함수

```
State allocate (Decls ds, State state) {  
    // 선언된 변수들(ds)을 위한 엔트리들을 상태 state에 추가  
}
```

- 엔트리 추가 함수

```
State push(Identifier id, Value val)
```

- free 함수

```
State free (Decls ds, State state) {  
    // 선언된 변수들(ds)의 엔트리를 상태 state에서 제거  
}
```


전역 변수 선언

- 언어 S의 명령어

$\langle \text{command} \rangle \rightarrow \langle \text{stmt} \rangle \mid \langle \text{decl} \rangle$
 $\langle \text{decl} \rangle \rightarrow \langle \text{type} \rangle \text{ id } [= \langle \text{expr} \rangle];$

- 상태 변환 함수

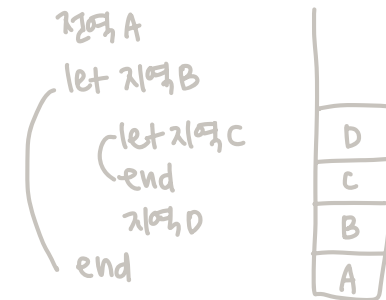
```
State Eval(Command p, State state) {  
  if (p instanceof Decl) {  
    Decls decls = new Decls();  
    decls.add((Decl) p);  
    return allocate(decls, state);  
  }  
  if (p instanceof Stmt) {  
    return Eval((Stmt) p, state);  
  }  
}
```

여러개 가능: recursive로 구현

let문에서의 allocate vs. 전역변수의 allocate

ex) let에서 선언후 전역변수를 선언하면?

→ 지역변수는 let문이 끝나면 pop돼서 사라짐
않아지 않는다!



선언이 하나인 경우는 이렇게 말고
stack에 직접 넣어도 됨