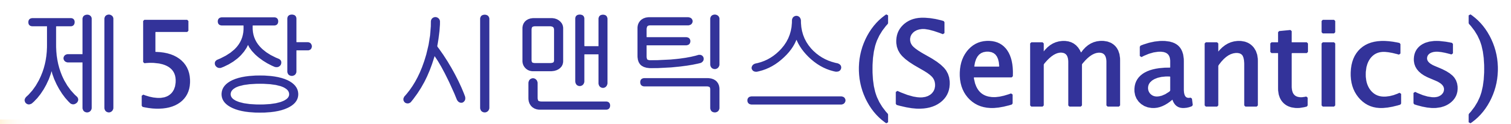
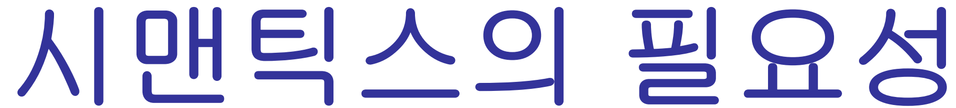


제5장 시맨틱스(Semantics)

5.1 간단한 시맨틱스



시맨틱스의 필요성

- 프로그램 의미의 정확한 이해
- 소프트웨어의 정확한 명세
- 소프트웨어 시스템에 대한 검증 혹은 추론
- 컴파일러 혹은 해석기 작성의 기초

- Operational Semantics
  - 프로그램의 동작 과정을 정의
- Denotational Semantics
  - 프로그램의 의미를 함수 형태로 정의
- Axiomatic Semantics
  - 프로그램의 시작 상태와 종료 상태를 논리적 선언 (assertion) 형태로 정의

동작 시맨틱스(Operational Semantics)

- 기본 아이디어
  - 프로그램의 의미(실행)를 상태 전이 과정으로 설명한다.
  - 각 문장 S마다 상태 전이 규칙 정의

- 예제

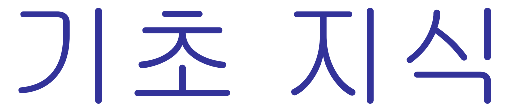
```
y := 1;
while (x != 1) do (y := x*y; x := x-1)
```

1. y에 1 배정
2. x 값이 1이 아닌지 검사
3. 참이면 종료
4. 거짓이면 y 값을 x 값과 현재 y 값의 곱으로 변경
5. x 값 1 감소
6. 2 번부터 반복

우리 언어

- 정수
  - n ∈ **Int**
- 변수
  - x ∈ **Var**
- 식
  - E ∈ **Exp**

```
Stmt S → x = E;
       | S; S
       | if E then S
       | if E then S else S
       | while E do S
       | read id
       | print E
Expr E → n | x | true | false
       | E + E | E - E | E * E | E / E
       | E == E | E < E | E > E | !E
Prgm P → S
```



기초 지식

- 합집합

  $A + B = \{ a \mid a \in A \vee a \in B\}$

- 곱집합

  $A \times B = \{(a,b) \mid a \in A \wedge b \in B\}$

- 함수집합

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

- 함수 $f \in A \rightarrow B$

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

- 함수 수정 $f[a \mapsto b]$

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

- [23 + 5] vs [x + y]
- [x + y]의 의미는 ?
- 변수 x, y의 현재 값에 따라 다르다.
- 변수의 현재 값을 무엇이라고 할까요?
- 상태(State)

상태(State)

- 상태(A state)
  - 변수들의 현재 값
  - 하나의 함수로 생각할 수 있다.
  - $s \in Var \rightarrow Int$
- 모든 상태들의 집합
  - $State = Var \rightarrow Int$
- 상태 s에서 변수 값
  - $s(x)$
- 상태 갱신: $s' = s[y \mapsto v]$

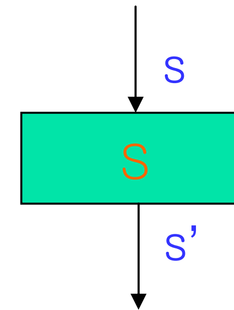
$$s'(x) = \begin{cases} s(x) & \text{if } x \mathrel{!=} y \\ v & \text{if } x = y \end{cases}$$

수식의 의미

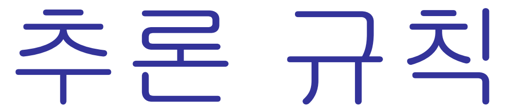
- 수식 E의 의미
  - 상태 s에서 수식 E의 값
  - [E]s
  - s = {x ↦ 1, y ↦ 3}
  - [x+y]s = 4
- 상태 s에서 수식 E의 의미
  - [true]s = T [false]s = F
  - [n]s = n
  - [x]s = s(x)
  - [E1 + E2]s = [E1]s + [E2]s
  - [E1 == E2]s = T if [E1]s == [E2]s
F otherwise
  - …

문장의 의미

- 문장 S의 의미
  - 문장 S가 상태 s를 s'으로 변경시킨다.
  - 상태 전이(state transition)라고 한다.
  - (s, S) → s'
- 상태 전이(State transition)
  - (s,S) → s' where
    - s is the initial state
    - S is a statement
    - s' is the final state



- 동작 시맨틱스
  - 각 문장 S마다 상태 전이 규칙 정의
  - 프로그램의 의미를 상태 전이 과정으로 설명한다.



- 조상 추론 규칙

$$\frac{\text{X parent Y}}{\text{X ancestor Y}} \qquad \frac{\text{X ancestor Y} \quad \text{Y parent Z}}{\text{X ancestor Z}}$$

$$\frac{\qquad\qquad}{\text{a parent b}} \qquad \frac{\qquad\qquad}{\text{b parent c}}$$

전이 규칙(Transition Relation)

- Assignment $\quad (s,\ x := E) \rightarrow s[x \mapsto [E]s]$

- skip

$$\frac{}{(s, skip) \rightarrow s}$$

- sequence

$$\frac{(s,\ S1) \rightarrow s',\ (s',\ S2) \rightarrow s''}{(s,\ S1;\ S2) \rightarrow s''}$$

- if-then-else

$$\frac{(s,\ S1) \rightarrow s'}{(s,\ \text{if } E \text{ then } S1 \text{ else } S2) \rightarrow s'} \quad \text{if } [E]s = T$$

$$\frac{(s,\ S2) \rightarrow s'}{(s,\ \text{if } E \text{ then } S1 \text{ else } S2) \rightarrow s'} \quad \text{if } [E]s = F$$

예1

```
x = 1;
y = 2;
if x > y then max =x;
          else  max =y;
```

$s0 = \{\ \}$

$(s0, x = 1) \rightarrow s1$ $s1 = \{x \mapsto 1\}$

$(s1, y = 2) \rightarrow s2$ $s2 = \{x \mapsto 1, y \mapsto 2\}$

$(s2, \text{if } x > y \ldots) \rightarrow s3$ $[x>y]\, s2 = F$

$(s2, max=y) \rightarrow s3$ $s3 = \{x \mapsto 1, y \mapsto 2, max \mapsto 2\}$

- while

$$\frac{(s, S) \rightarrow s', \quad (s', \text{while } E \text{ do } S) \rightarrow s''}{(s, \text{while } E \text{ do } S) \rightarrow s''} \quad \text{if } [E]s = T$$

$$(s, \text{while } E \text{ do } S) \rightarrow s \qquad \text{if } [E]s = F$$

- read

$$\frac{\text{READ } n}{(s, \text{read } x) \rightarrow (s[x \mapsto n])}$$

- Print

$$\frac{\text{PRINT } [E]s}{(s, \text{print } E) \rightarrow s}$$

예2

```
read x;
y = 1;
while (x != 1) do (y = x*y; x = x-1)
```

$s0 = \{ \}$

$(s0, \text{read } x) \rightarrow s1$ $\quad s1 = \{x \mapsto 3\}$

$(s1, y = 1) \rightarrow s2$ $\quad s2 = \{x \mapsto 3, y \mapsto 1\}$

$(s2, \text{while } \ldots) \rightarrow s''$ $\quad [x \mathrel{!=} 1]s2 = T$

$\quad (s2, y = x*y; x = x-1) \rightarrow s3$ $\quad s3 = \{x \mapsto 2, y \mapsto 3\}$

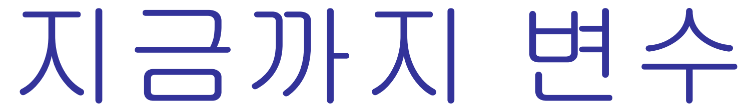
$\quad (s3, \text{while } \ldots) \rightarrow s''$ $\quad [x \mathrel{!=} 1]s3 = T$

$\qquad (s3, y = x*y; x = x-1) \rightarrow s4$ $\quad s4 = \{x \mapsto 1, y \mapsto 6\}$

$\qquad (s4, \text{while } \ldots) \rightarrow s''$ $\quad [x \mathrel{!=} 1]s4 = F$

$s'' = s4$

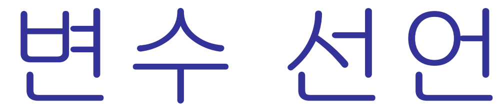
5.2 변수 선언 및 유효범위



- Variable x ⇔ Locations in Memory
  - 1:1 대응

- 변수의 유효범위(scope)
  - 프로그램 전체

- 변수 x의 의미
  - $[x]s = s(x)$ 메모리 위치에 저장된 값(r-value)
  - $(s, x = E) \rightarrow s[x \mapsto [E]s]$ 메모리 위치(l-value)



- Syntax

```
S → …
      | let x = e in S end
      | let int x = e in S end
```

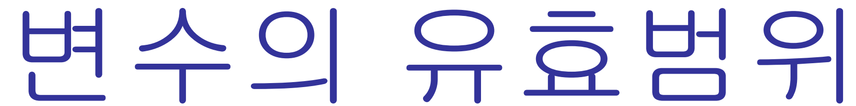
- Example

```
let x = 1 in
       let y = 2 in
               x = x + y end;
       let x = 5 in
               x = x + 1 end;
       x = x * 2
 end
```

- Semantics
  - 변수 x는 선언된 블록 내에서만 유효하다.
  - 이름 x가 여러 location를 나타낼 수 있다.

블록(Blocks)

- Pascal
  - procedure or function
- C, C++, Java
  - { . . . }
- Ada
  - declare
    . . .
    begin ... end
- ML
  - let ... in ... end



- 변수의 유효범위
  - 변수가 사용될 수 있는 프로그램 내의 범위

- Variable x ⇔ Locations in Memory
  - 1: n 대응

  - x → loc1
  - x → loc2
  - x → loc3

시맨틱스

- 수식 혹은 문장의 의미
  - 변수가 나타내는 location이 여러 개 이므로
  - 변수의 location 정보가 필요하다.

- Environment
  - 각 변수가 가리키는 location(address)을 표현
  - $\sigma \in Env = Var \rightarrow Loc$

- Memory
  - 각 메모리 위치에 저장된 값을 표현
  - $M \in Memory = Loc \rightarrow Value$

- 참고
  - $State = Var \rightarrow Value$

- Env σ, Memory M에서 수식 E의 의미
  - σ,M에서 수식 E의 값
  - [E]σ,M

  - $\sigma = \{x \mapsto l_1,\ y \mapsto l_2\}$
  - $M = \{l_1 \mapsto 2,\ l_2 \mapsto 3\}$
  - $[x+y]\sigma;M = [x]\sigma;M + [y]\sigma;M = M(\sigma(x)) + M(\sigma(y)) = 2+3$

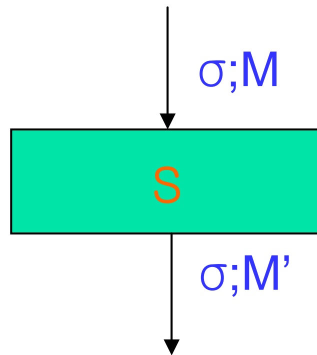
수식의 의미

- Env σ, Memory M에서 수식 E의 의미
  - [true]σ;M = T   [false]σ;M = F
  - [n]σ;M = n
  - [x]σ;M = M(σ(x))
  - [E1 + E2]σ;M = [E1]σ;M + [E2]σ;M
  - [E1 == E2]σ;M = T if [E1]σ;M == [E2]σ;M
F otherwise
  - …

문장 S의 의미

- Env σ, Memory M에서 문장 S의 실행

  (σ;M, S) → M'



- Assignment $(\sigma;M,\ x = E) \to M[\sigma(x) \mapsto [E]\sigma;M]$

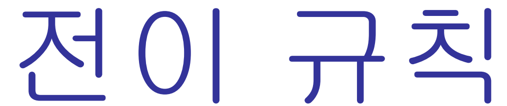
- sequence

$$\frac{(\sigma;M,\ S1) \to M',\ (\sigma;M',\ S2) \to M''}{(\sigma;M,\ S1;\ S2) \to M''}$$

- if-then-else

$$\frac{(\sigma;M,\ S1) \to M'}{(\sigma;M,\ \text{if } E \text{ then } S1 \text{ else } S2) \to M'},\quad \text{if } [E]\ \sigma;M = T$$

$$\frac{(\sigma;M,\ S2) \to M'}{(\sigma;M,\ \text{if } E \text{ then } S1 \text{ else } S2) \to M'},\quad \text{if } [E]\ \sigma;M = F$$



- while

$$\frac{(\sigma;M,\ S) \rightarrow\ M',\ \ (\sigma;M',\ \text{while } E \text{ do } S) \rightarrow M''}{(\sigma;M,\ \text{while } E \text{ do } S) \rightarrow M''} \quad \text{if } [E]\ \sigma;M = T$$

$$(\sigma;M,\ \text{while } E \text{ do } S) \rightarrow M \quad \text{if } [E]\ \sigma;M = F$$

- read

$$\frac{\text{READ } n}{(\sigma;M,\ \text{read } x) \rightarrow (M[\sigma(x) \mapsto n])}$$

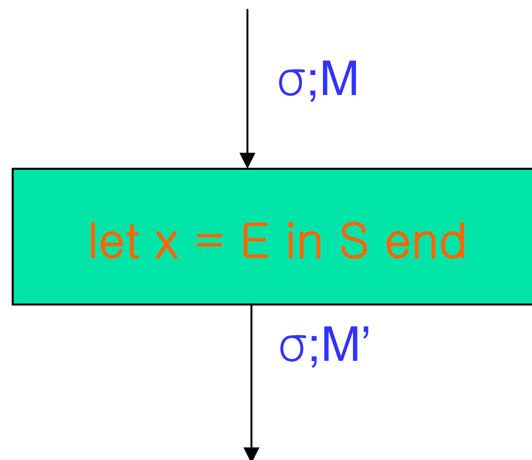
- print

$$\frac{\text{PRINT } [E]\ \sigma;M}{(\sigma;M,\ \text{print } E) \rightarrow M}$$

전이 규칙

- let … in … end

$$\frac{(\sigma[x \mapsto l];M[l \mapsto v],\ S) \rightarrow M' \quad \text{where new } l,\ v=[E]\sigma;M}{(\sigma;M,\ \text{let } x = E \text{ in } S \text{ end}) \rightarrow M'}$$

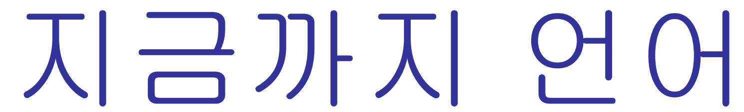


- let 블록 내에서 환경이 바뀌고
- 바뀐 환경에서 S를 실행하고
- let 블록이 끝나면 다시 시작 환경과 같아진다.

Example

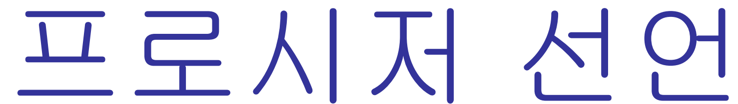
```
0 let x = 1 in
1      let y = 2 in
2                x = x + y end;
3      let x = 5 in
4                x = x + 1 end;
5      x = x * 3
6 end
```

- $\sigma_0 = \{\ \}$ $M_0 = \{\ \}$
- $\sigma_1 = \{x \mapsto l_1\}$ $M_1 = \{l_1 \mapsto 1\}$
- $\sigma_2 = \{x \mapsto l_1, y \mapsto l_2\}$ $M_2 = \{l_1 \mapsto 1, l_2 \mapsto 2\}$
- $\sigma_3 = \sigma_1 = \{x \mapsto l_1\}$ $M_3 = \{l_1 \mapsto 3, l_2 \mapsto 2\}$
- $\sigma_4 = \{x \mapsto l_3\}$ $M_4 = \{l_1 \mapsto 3, , l_2 \mapsto 2, l_3 \mapsto 5\}$
- $\sigma_5 = \sigma_3 = \sigma_1 = \{x \mapsto l_1\}$ $M_5 = \{l_1 \mapsto 3, l_2 \mapsto 2, l_3 \mapsto 6\}$
- $\sigma_6 = \sigma_5 = \sigma_3 = \sigma_1 = \{x \mapsto l_1\}$ $M_6 = \{l_1 \mapsto 9, l_2 \mapsto 2, l_3 \mapsto 6\}$



- 변수와 문장만 있고
- 프로시저(함수, 메쏘드)가 없다.

프로시저 선언



- Syntax

  S → …

  | let proc f(x) : S in S end

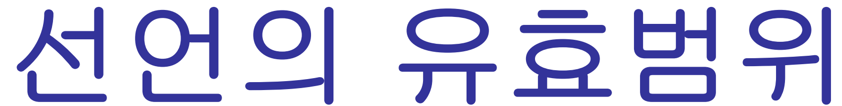
  | call f(E)

- Example 1

```
let sum = 0 in
      let proc add(x) : sum = sum + x in
          call add(9)
      end
end
```

- Semantics
  - 이름 f는 선언된 블록 내에서만 유효하다.
  - 이름 f는 정의된 프로시저를 나타낸다.

선언의 유효범위



- 선언된 식별자(이름)의 유효범위(Scope)
  - 선언된 이름이 유효한(사용 가능한) 프로그램의 범위 혹은 영역
  - 변수 이름뿐 아니라 함수 이름 등도 생각해야 한다.
  - 유효범위 규칙
- 정적 유효범위(Static scope)
  - 선언된 이름은 선언된 블록 내에서만 유효함
  - 대부분 언어에서 표준 규칙으로 사용됨
- 동적 유효범위(Dynamic scope)
  - 선언된 이름은 선언된 블록의 실행이 끝날 때까지 유효함
  - 실행 경로에 따라 유효범위가 달라질 수 있다.
  - Old Lisp와 SNOBOL에서 사용됨

Example 2

```
0  let x = 0 in
1       let proc add(y) : x = x + y in
2           let x = 2 in
3                call add(9)
4           end
5       end
6  end
```

환경 집합 Env

- 지금까지 환경 집합 Env
  - 변수 이름만 있었다.
  - Env = Var → Loc

- 이제부터 환경 집합 Env
  - Id = 변수 혹은 프로시저 이름 집합
  - Env = Id → Loc + Proc
  - 유효한 (변수,함수) 이름에 대한 정보를 유지한다.

- Proc
  - Proc = 프로시저 집합
  - Proc = Var × S (매개변수와 본체 문장)

- 하나의 프로시저
  - proc f(x): S_1
  - $\langle x, S_1 \rangle \in$ Proc = Var × S
  - f $\mapsto \langle x, S_1 \rangle$ in an env σ

프로시저 전이 규칙

- Dynamic Scope
  - 프로시저 선언을 만나면 환경(σ)을 바꾸고 $\sigma[f \mapsto <x, S_1>]$
  - 호출되면 호출된 환경(σ)에서 프로시저 문장 S_1을 실행한다.

- Proc

$$\frac{(\sigma[f \mapsto <x, S_1>];M, S_2) \rightarrow M'}{(\sigma;M, \text{let proc } f(x): S_1 \text{ in } S_2 \text{ end}) \rightarrow M'}$$

$$\frac{(\sigma[x \mapsto l];M[l \mapsto v], S_1) \rightarrow M' \quad \text{where new } l,\ v=[E]\sigma;M}{(\sigma;M, \text{call } f(E)) \rightarrow M'} \quad \text{if } \sigma(f)=<x, S_1>

Example 1

```
0    let x = 0 in
1        let proc add(y) : x = x + y in
2            call add(9)
3        end
4    end
```

- $\sigma_0 = \{\ \}$ $\quad M_0 = \{\ \}$
- $\sigma_1 = \{x \mapsto l_1\}$ $\quad M_1 = \{l_1 \mapsto 0\}$
- $\sigma_2 = \{x \mapsto l_1, add \mapsto \langle y, x=x+y \rangle\}$ $\quad M_2 = \{l_1 \mapsto 0\}$
- $\sigma_{add_entry} = \{x \mapsto l_1, add \mapsto \ldots, y \mapsto l_2\}$ $\quad M_{add_entry} = \{l_1 \mapsto 0, l_2 \mapsto 9\}$
- $\sigma_{add_exit} = \{x \mapsto l_1, add \mapsto \ldots, y \mapsto l_2\}$ $\quad M_{add_exit} = \{l_1 \mapsto 9, l_2 \mapsto 9\}$
- $\sigma_3 = \sigma_2 = \{x \mapsto l_1, add \mapsto \ldots\}$ $\quad M_3 = \{l_1 \mapsto 9, l_2 \mapsto 9\}$
- $\sigma_4 = \sigma_1 = \{x \mapsto l_1\}$ $\quad M_4 = \{l_1 \mapsto 9, l_2 \mapsto 9\}$

프로시저 전이 규칙

- Static Scope
  - 프로시저 선언을 만나면 환경(σ')을 바꾸고 $\sigma'[f \mapsto \langle x, S_1, \sigma' \rangle]$
  - 이때 선언된 환경(σ')도 함께 저장한다.
  - 호출되면 선언된 환경(σ')에서 프로시저 문장 S_1을 실행한다.

- Proc

$$\frac{(\sigma'[f \mapsto \langle x, S_1, \sigma' \rangle];M, S_2) \rightarrow M'}{(\sigma';M, \text{let proc } f(x)\text{: } S_1 \text{ in } S_2 \text{ end}) \rightarrow M'}$$

$$\frac{(\sigma'[x \mapsto l];M[l \mapsto v], S_1) \rightarrow M' \quad \text{where new } l,\ v = [E]\sigma;M}{(\sigma;M, \text{call } f(E)) \rightarrow M' \quad \text{if } \sigma(f) = \langle x, S1, \sigma' \rangle}$$

Example2

```
0    let x = 0 in
1       let proc add(y) : x = x + y in
2          let x = 2 in
3               call add(9)
4          end
5       end
6    end
```

- $\sigma_0 = \{\ \}$ $\quad M_0 = \{\ \}$
- $\sigma_1 = \{x \mapsto l_1\}$ $\quad M_1 = \{l_1 \mapsto 0\}$
- $\sigma_2 = \{x \mapsto l_1, \text{add} \mapsto <y, x=x+y, \sigma_1>\}$ $\quad M_2 = \{l_1 \mapsto 0\}$
- $\sigma_3 = \{x \mapsto l_2, \text{add} \mapsto \ldots\}$ $\quad M_3 = \{l_1 \mapsto 0, l_2 \mapsto 2\}$
- $\sigma_{add_entry} = \{x \mapsto l_1, \text{add} \mapsto \ldots, y \mapsto l_3\}$ $\quad M_{add_entry} = \{l_1 \mapsto 0, l_2 \mapsto 2, l_3 \mapsto 9\}$
- $\sigma_{add_exit} = \{x \mapsto l_1, \text{add} \mapsto \ldots, y \mapsto l_3\}$ $\quad M_{add_exit} = \{l_1 \mapsto 9, l_2 \mapsto 2, l_3 \mapsto 9\}$
- $\sigma_4 = \sigma_3 = \{x \mapsto l_2, \text{add} \mapsto \ldots\}$ $\quad M_4 = \{l_1 \mapsto 9, l_2 \mapsto 2, l_3 \mapsto 9\}$
- $\sigma_5 = \sigma_2 = \{x \mapsto l_1, \text{add} \mapsto \ldots\}$ $\quad M_5 = \{l_1 \mapsto 9, l_2 \mapsto 2, l_3 \mapsto 9\}$
- $\sigma_6 = \sigma_1 = \{x \mapsto l_1\}$ $\quad M_5 = \{l_1 \mapsto 9, l_2 \mapsto 2, l_3 \mapsto 9\}$

다른 언어에서 선언 및 유효범위

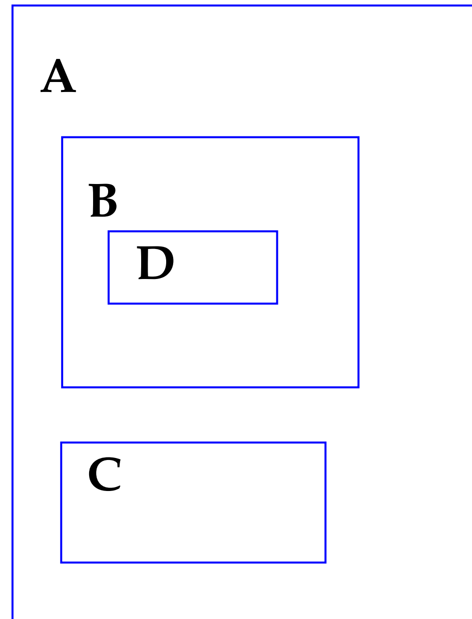
- 무엇을 선언하는가?
  - 상수, 변수, 함수, 클래스

- 어디에 선언하는가?
  - 블록 내 혹은 밖
  - 구조체(struct)
  - 클래스

- 어떻게 선언하는가?
  - 명시적 선언: Pascal,C, C++, Java, ...
  - 묵시적 선언: Lisp, partly in FORTRAN

블록 구조 언어

- 블록의 중첩을 허용하는 언어



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

C의 블록

```
void p (void)
{ double r, z;   /* p의 블록 */
   …
   { int x,y;       /* 또 하나의 중첩 블록 */
            x = 2;
            y = 0;
            x += 1;
    }
    …
}

int x;
double y;
main()
{  int i, j;          /* main 블록 내 */
   …
}
```

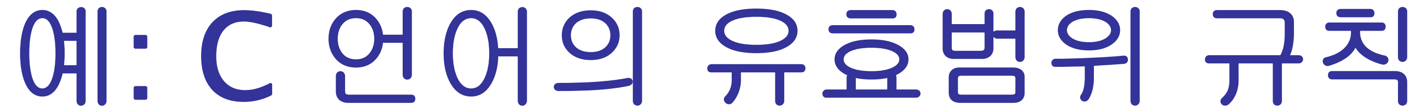
```
declare  x:integer;
         y: boolean;
begin
    x := 2;
    y := true;
    x := x+1;
    …
end
```

Pascal의 블록

```
program ex; (* main 프로그램 *)
    var x: integer;  (* 전역 선언 *)
    procedure p; (* 전역 선언 *)
        var y: boolean; (* p 내의 선언 *)
        begin
            if x = 2 then begin … end;
        end;
begin (* main *)
(* 선언할 수 없음 *)
…
end. (* 프로그램 *)
```

클래스 내의 선언

```
public class IntWithGcd
{
    private int value;    /* value 필트 지역 선언 */

    public int intValue()          /* intValue 메쏘드 지역 선언 */
    {
        return value;
    }

    public int gcd(int v)          /* gcd 메쏘드 지역 선언 */
    {
        ...
    }
}
```



예: 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

예제

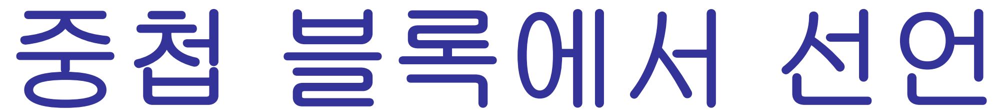
```
void p( )                    p
{    char y;                 y
     ...
}

int x;                       x

void q( )                    q
{    double z;               z
     ...
}

main()                       main
{    int w[10];              w
     ...
}
```

- 핵심 아이디어
  - 선언의 유효범위는 선언된 블록 내

- 예제

```
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 블록에서 선언



- 최중첩 우선 규칙(Most closely nested rule)
  - 한 식별자가 여러 번 선언된 경우 최중첩된 선언을 우선한다.

- 예

```
int x;
void p( )
{   char x;
    x = 'a';  /*                    */
    ...
}

main()
{   x = 2;  /*                      */
    ...
}
```

5.3 환경과 바인딩

프로그래밍 언어의 구현

- 프로그래밍 언어 어떻게 구현할까요?
  - 컴파일러
    - 입력 프로그램을 시맨틱스에 맞게 코드 생성
  - 인터프리터
    - 입력 프로그램을 시맨틱스에 맞게 실행
- 컴파일러 혹은 인터프리터
  - 컴파일 혹은 해석하기 위해 환경(Env)를 유지한다.
- 시맨틱스에서 환경(Env)를 생각해 보자.
  - 현재 유효한 이름을 유지한다.
  - 이름이 의미하는 대상(바인딩 정보)을 유지한다.
  - 변수, 프로시저, 상수, 타입 이름

- 바인딩은 무엇인가?
  - 이름이 의미하는 대상을 정하는 것
  - 이름에 따라 위치, 크기, 정의된 함수, 타입 등을 정한다.
    - 변수, 상수, 클래스, 메쏘드, 인터페이스, 매개변수 등.

- 이름의 의미
  - 의미하는 대상에 의해서 결정된다

- 바인딩 시간
  - 정적 바인딩
    - compile time
  - 동적 바인딩
    - execution time

- 정적 환경(Static Environment)
  - 컴파일러가 컴파일 하기 위해서는 각 지점에서 유효한
  - 정적 바인딩 정보를 유지해야 한다.

- 정적 바인딩(Static Binding)
  - 컴파일 시간에 이름이 의미하는 대상
  - 변수 이름 → 변수의 타입/위치
  - 함수 이름 → 매개변수 타입/반환 타입

- 아이디어
  - 블록 시작
    - 블록 시작 부분의 선언을 처리하여
    - 해당 바인딩을 정적 환경에 추가한다.
  - 블록 내
    - 블록 내의 문장들을 컴파일 한다.
  - 블록의 끝
    - 블록 시작에서 선언된 바인딩을 정적 환경에서 제거한다.
    - 더 이상 필요 없다.

정적 환경/바인딩 예1

```
0 let x = 1 in
1       let y = 2 in
2                 x = x + y end;
3       let x = 5 in
4                 x = x + 1 end;
5       x = x * 3
6 end
```

- $\sigma_0 = \{ \}$
- $\sigma_1 = \{x \mapsto <int, l_1> \}$
- $\sigma_2 = \{x \mapsto <int, l_1>, y \mapsto <int, l_2> \}$
- $\sigma_3 = \sigma_1 = \{x \mapsto <int, l_1>\}$
- $\sigma_4 = \{x \mapsto <int, l_1>, x \mapsto <int, l_3> \}$
- $\sigma_5 = \sigma_3 = \sigma_1 = \{x \mapsto <int, l_1> \}$
- $\sigma_6 = \sigma_5 = \sigma_3 = \sigma_1 = \{x \mapsto <int, l_1> \}$

정적 환경/바인딩 예2

```
0   let int x = e in
1         let proc add(y): entry x = x + y in exit
2             call add(9)
3         end
4   end
```

- $\sigma_0 = \{\ \}$
- $\sigma_1 = \{x \mapsto <int, l_1>\}$
- $\sigma_{add_entry} = \{x \mapsto <int, l_1>, add \mapsto <proc, int, void>, y \mapsto <int, l_2>\}$
- $\sigma_{add_exit} = \{x \mapsto <int, l_1>, add \mapsto <proc, int, void>\}$
- $\sigma_2 = \{x \mapsto <int, l_1>, add \mapsto <proc, int, void>\}$
- $\sigma_3 = \sigma_2 = \{x \mapsto <int, l_1>, add \mapsto <proc, int, void>\}$
- $\sigma_4 = \sigma_1 = \{x \mapsto <int, l_1>\}$

동적 바인딩(Dynamic Binding)

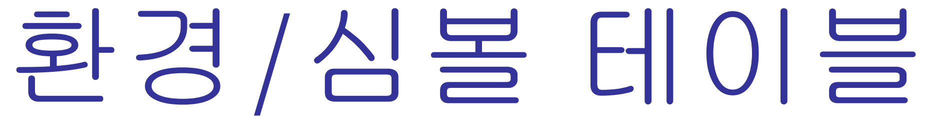
- 실행 시간에 이루어진 바인딩.
  - 이름이 의미하는 대상이 동적으로 결정

- Java 예제

```
C  x;
if (...) x = new C( );          /* memory allocation */
else x = new C1( );             /* C1 is a subclass of C */
x.m( ... );                     /* method dispatch */
```

- 동적으로 바인딩되는 것은 무엇인가?
  - x가 의미하는 대상
  - m이 의미하는 대상

- 언어 정의 시간(Language definition time)
  - boolean, true, false, char, integer,maxint
- 언어 구현 시간(Language implementation time)
  - integer, maxint
- 컴파일 시간(Compile- time)
- 링크/로드 시간(Link/load time)
  - external definition/the location of a global variable
- 실행 시간(Runtime)



- 환경(environment)
  - 현재 유효한 바인딩을 유지한다.

- 심볼 테이블
  - 현재 유효한 바인딩을 유지/관리하기 위한 자료 구조