# Names, Bindings, and Scopes

국민대학교 컴퓨터공학부
강 승 식

# 명령형(절차적) 언어와 변수

- Imperative language
  - *abstraction of von Neumann computer architecture*
    - Memory : instructions and data
    - Processor : operations for modifying the contents of memory

- Variable
  - *abstraction of the memory cells*
    - Characterized by a collection of properties(attributes)
  - Type, scope rule, lifetime, type checking, initialization

- Name(identifier) 명칭
  - variable, procedure, type, and constant

    변수명 함수명 타입명칭 상수명

- Location 대표적 변수 = location
  - Places(addresses) where values can be stored

- Value
  - storable quantities: integers, reals, array values, etc

# Name: design issues

- The maximum length of a name
  - 6(Fortran I, Fortran77), 31(Fortran90, C)
  - no limit(Ada, Java), C++ don’t specify a length limit
- Connector characters : ’_’
- Case sensitive : C, C++, Java

대소문자 구분// 나머지는 대부분 대소문자 동일함 구분안함

- Keywords or reserved words(predefined name)
  - C, C++ : many names are predefined in libraries

예약되어 있는 단어는 변수명으로 사용 할 수 없음 ex) if, for while 등
그러나 다른 언어들은 가능함

# Name: attributes

- Example code in Pascal

```
const n = 5;
var x: integer;
function f(m:integer): boolean
  begin
    . . .
  end
```

c언어는 {}

- What are the attributes of names ?
  - n :
  - x :
  - f :

# Name: attributes

- What kinds of attributes?
  - *Name, address(location), value, type, size, scope(static-nesting-level), lifetime, …*
  - The meaning of a name is determined by attributes (properties)

- How can we associate attributes to names?
  - By declarations, assignment, …

시험범위 여기까지..???

# Address or Location

- **The memory address with which it is associated**

- **For the same name,**
  - Different address at different places and at different times

- ***l*-value** : the address of a variable

# Alias

- Multiple identifiers reference the same address

- Implementation in programming languages
  - Fortran : EQUIVALENCE statement
  - C, C++ : union types, pointer, reference
  - Pascal, Ada : variant record

- Side effect is a critical problem!

# Type

- Determines *the range of values* and *the set of operations*

- Example
  - 16-bit integer
    - Range of values: -32,768 ~ + 32,767
    - Set of operations: ‘+’, ‘-’, ‘*’, ‘/’, mod, …

# Binding

- *The process of associating an attribute to a name*

- Binding time
  - Compile time -- Static binding
    - Static attributes
      - *int n = 100;*
      - *double pi = 3.14;*
  - Run time (Execution time) -- Dynamic binding
    - Dynamic attributes
      - *C x = new C();              // Java -- object*
      - *int *p = malloc(100);   // C -- memory allocation*

# Binding Times

- Language definition time
  - *boolean, true, false, char, integer, maxint*
- Language implementation time
  - *integer, maxint*
- Compile-time
- Link/load time
  - external definition/the location of a global variable
- Runtime

- *‘*’* is bound to the multiplication operation at language design time.
- *INTEGER type of Fortran* is bound to a range of possible values at language implementation time.
- *Java variable* is bound to a data type at compile time.
- *A call to a subprogram* is bound to the code at link time
- *A variable* may be bound to a storage cell when the program is loaded into memory

# Type Binding

- Variable declaration
  - Explicit declaration : most programming languages
  - Implicit declaration : Fortran
  - Perl: $(scalar), @(array), %(hash structure)

- Dynamic type binding
  - Type is not specified by a declaration statement.
  - Variable is bound to a type when it is assigned a value.

  - Provides a great deal of programming flexibility.
  - APL, SNOBOL4, JavaScript, LISP, Python : interpreter
  - Error detection capability of a compiler is diminished.

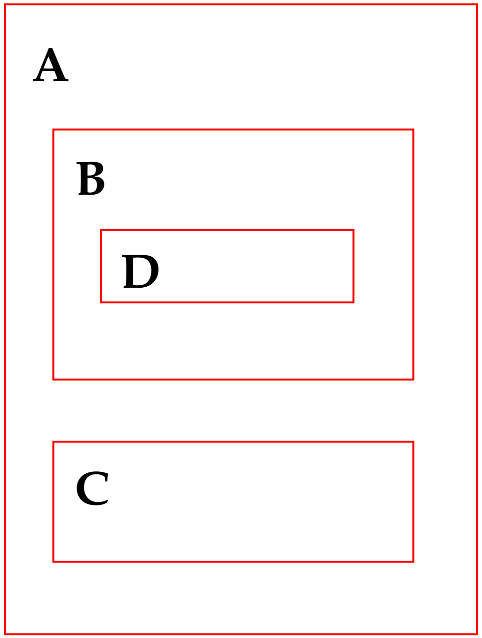
# Blocks and Scope

# Blocks

- Pascal
  - Procedure
  - Function
- C, C++, Java
  ```
  {
      ...
  }
  ```
- Ada
  ```
  declare
      ...
  begin
      …
  end
  ```
- ML
  ```
  let
      …
  in
      …
  end
  ```

# Block Structured Languages

- Blocks can be nested
- Algol, Pascal, Modula, Ada, C, ...



# Global/Local declaration

```
program ex;
        var x: integer; (* global declarations *)
        procedure p;
                var y: boolean;  (* local declarations of p *)
                begin
                  ...
                end;
        procedure q;
                var z: real;  (* local declarations of q *)
                begin
                  ...
                end;
         ...
  begin (* main  *)
         ...
  end. (* main *)
```

# Scope

- The region of a program in which an identifier is valid(accessible)

- Static scope(Lexical scope)
    - the scope of a declaration is limited to the block in which it is declared
    - the standard scope rule in most languages

- Dynamic scope in Lisp and SNOBOL
    - Scope of an identifier depends on runtime execution
    - Dynamic languages

- Pascal, C/C++
  - Declaration before use
  - The scope of declaration is the block *from the point of  a declaration*

- Algol, Ada     선언 블락 시작전 int i=j;     int j=100; 해도 아무런 문제 없음
  - The scope of declaration is the block *from the beginning*

```
program symtabex;
        var  x, z : integer;
        procedure p;    함수 선언문
                var x: boolean;
                procedure q;
                        var y: integer;
                        begin (* q *)
                           y := x + z;
                        end; (* q *)
                begin (* p *)
                  . . .
                end (* p *)
begin (* main *)
  . . .
end.
```

# Dynamic Scope

- Declarations are processed as they are encountered along an *execution path through the program*

  (assume that the symbol table is managed dynamically)
- A dynamically-scoped variable *refers to the closest enclosing binding* in the execution of the program.

- Example: h() → f(), g()→ f()

```
fun h(y)      { int a = 100;   f(3)  }
fun g(y)      { int a = 200;   f(3)  }
fun f(x)      { print a }
```

# Memory Allocation

- Static allocation (at compile-time)
- Dynamic allocation (at run-time)

- FORTRAN
    - All locations are bound statically
- LISP, Python
    - All locations are bound dynamically
- Pascal, C, Modula-2, Java
    - Some allocation statically, others dynamically

- Global variables
  - static allocation at compile-time

- Local variables
  - dynamic allocation on runtime stack when execution reaches the block

- Dynamic memory allocation
  - malloc() in C, new() in Pascal, Java
  - dynamic allocation on heap when executing the function