# COMPILER (CSE 4120)

(Lecture 11: Semantic Analysis 4)

Sungwon Jung, Ph.D.

Mobile Computing and Data Engineering Lab.
Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea
Tel: +82-2-705-8930
Email : jungsung@sogang.ac.kr

Dept. of Computer Science at Sogang University

## Type Inference and Type Checking

- Describe a type checker for a simple language in terms of semantic actions, based on a representation of types and a *typeEqual* operation
- A Simple grammar to illustrate type checking

*program* → *var-decls* **;** *stmts*
*var-decls* → *var-decls* **;** *var-decl* | *var-decl*
*var-decl* → **id** **:** *type-exp*
*type-exp* → **int** | **bool** | **array** **[num]** **of** *type-exp*
*stmts* → *stmts* **;** *stmt* | *stmt*
*stmt* → **if** *exp* **then** *stmt* | **id** **:=** *exp*

  - Assume the availability of a symbol table
    - Contains variable names and associated types
    - Have operations
      - ***insert*** – inserts a name and a type into the table
      - ***lookup*** – returns the associated type of a name

```
program → var-decls ; stmts
var-decls → var-decls ; var-decl | var-decl
var-decl → id : type-exp
type-exp → int | bool | array [num] of type-exp
stmts → stmts ; stmt | stmt
stmt → if exp then stmt | id := exp
```

# Type Inference and Type Checking

- Declarations
  - Causes the type of an identifier to be entered into the symbol table
    - the grammar rule

      *var-decl* → ***id*** **:** *type-exp*

      has the associated semantic action

      *insert*(***id***.*name*, *type-exp.type*)
    - int and bool

| | |
|---|---|
| *type-exp* → **int** | *type-exp.type* := *integer* |
| *type-exp* → **bool** | *type-exp.type* := *boolean* |

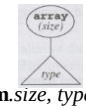
    - Types are assumed to be kept as some kind of tree structure
      - *makeTypeNode*(*array*, *size*, *type*) construct the following type node
      - *Grammar Rule:*

        $type\text{-}exp_1$ → **array [num] of** $type\text{-}exp_2$

      - *Associated Semantic rule:*

        $type\text{-}exp_1.type$ := *makeTypeNode*(*array*, **num**.*size*, $type\text{-}exp_2.type$)

      - The child of the array node is the type given by the type parameter









```
program → var-decls ; stmts
var-decls → var-decls ; var-decl | var-decl
var-decl → id : type-exp
type-exp → int | bool | array [num] of type-exp
stmts → stmts ; stmt | stmt
stmt → if exp then stmt | id := exp
```


# Type Inference and Type Checking

- Statements
  - Statements do not themselves have types, but substructures will need to be checked for type correctness
  - Examples:
    - *if-statement* :
      - The conditional expression must have Boolean type
      - Grammar rule: *stmt* → **if** *exp* **then** *stmt*
      - Semantic rule: **if not** *typeEqual(exp.type, boolean)* **then** *type-error(stmt)*
    - *assignment statement*
      - The variable being assigned must have the same type as the expression whose value it is to receive
      - Grammar rule: *stmt* → **id :=** *exp*
      - Semantic rule:

        **if not** *typeEqual*(*lookup*(**id**.*name*), *exp.type*) **then** *type-error(stmt)*

```
program → var-decls ; stmts
var-decls → var-decls ; var-decl | var-decl
var-decl → id : type-exp
type-exp → int | bool | array [num] of type-exp
stmts → stmts ; stmt | stmt
stmt → if exp then stmt | id := exp
```

# Type Inference and Type Checking

- Expressions
  - The missing grammar rules for the above grammar:
    - exp → exp + exp | exp or exp | exp [exp] | num | true | false | od
  - Constant expressions, such as numbers and the Boolean value true and false, have implicitly defined *integer* and *Boolean* types

| | |
|---|---|
| $exp_1 \rightarrow exp_2$ **+** $exp_3$ | **if not** (*typeEqual*($exp_2$.*type*, *integer*) **and** *typeEqual*($exp_3$.*type*, *integer*)) **then** *type-error*($exp_1$) ; $exp_1$.*type* := *integer* |
| $exp_1 \rightarrow exp_2$ **or** $exp_3$ | **if not** (*typeEqual*($exp_2$.*type*, *boolean*) **and** *typeEqual*($exp_3$.*type*, *boolean*)) **then** *type-error*($exp_1$) ; $exp_1$.*type* := *boolean* |
| $exp_1 \rightarrow exp_2$ **[** $exp_3$ **]** | **if** *isArrayType*($exp_2$.*type*) **and** *typeEqual*($exp_3$.*type*, *integer*) **then** $exp_1$.*type* := $exp_2$.*type.child1* **else** *type-error*($exp_1$) |
| *exp* → **num** | *exp.type* := *integer* |
| *exp* → **true** | *exp.type* := *boolean* |
| *exp* → **false** | *exp.type* := *boolean* |
| *exp* → **id** | *exp.type* := *lookup*(**id**.*name*) |