# Sacco Alessandro Compilatori - Esercizio5

# SymbolTable

La Symbol Table è stata implementata come una classe che contiene uno Stack di Scope (stack di lavoro). La classe Scope è una classe che contiene un riferimento ad una HashMap<Key,Type> ed un riferimento all’eventuale Scope padre (implementazione di uno stack implicito utile durante il Type Checking e la verifica della dichiarazione di variabili/procedure, null se lo scope è quello relativo al nodo ProgramOp).

# Scoping rules

## ProgramOp

(Essendo il primo nodo che si visita ottiene il riferimento alla SymbolTable con il primo livello di scoping già inizializzato).

Si lega il riferimento allo scope al nodo ProgramOp

Una volta visitato tutti i figli del nodo poi si rimuove lo scope dal top dello stack

## ProcDeclOp

Se la procedura è stata già dichiarata all’interno dello scope attuale viene lanciata una MultipleDelcarationException.

Altrimenti, la dichiarazione della procedura viene aggiunta allo scope, si effettua la creazione del nuovo livello di scoping, lo si inserisce al top dello stack e si lega lo scope al nodo.

Dopo aver visitato tutti i figli del nodo ProcDeclOp ed essere tornato al padre, si rimuove lo scope dal top dello stack.

## VarOp

Cosi come ProcDeclOp, se il nodo VarOp compare come figlio di un VarDeclOp (si sta dichiarando variabile)

Se all’interno dello scope attuale la variabile è stata già dichiarata viene lanciata una MultipleDeclarationException.

Altrimenti, la variabile viene aggiunta nella SymbolTable insieme alle informazioni sul tipo.

Si lega lo scope attuale al nodo

## Altri tipi di nodi

Si lega semplicemente lo scope attuale al nodo

# Type-Check

## VarOp e ProcCallOp

Se il nodo è relativo ad un identificatore (sia esso una variabile o una procedura), si effettua la lookup dell’identificativo associato all’interno della **SymbolTable**:

## Lookup

Si preleva lo scope legato al nodo considerato, si ricerca l’identificatore all’interno dello scope e si inserisce il risultato della ricerca in ***returned***

Finchè *returned* è *null*

Inserisci in scope il riferimento allo scope padre  
 Se lo scope padre è null (Abbiamo raggiunto il bottom dello stack implicito)

Restituisci null (L’identificatore non è stato trovato in nessuno scope)

Altrimenti, si ricerca l’identificatore all’interno del nuovo scope e si inserisce il risultato in returned

Se il risultato della lookup è null si lancia una **UndeclaredIdentifierException** con un opportuno messaggio di errore

Altrimenti, si restituisce il tipo memorizzato all’interno della **SymbolTable**

## ConstOp

Se il nodo è un ConstOp, restitusici come tipo il tipo della costante stessa (*Boolean, Integer).*

## Altri nodi

Seil nodo è legato ad un costrutto riguardante operatori di espressioni o istruzioni (qui bisogna  
fare riferimento al Type System di sotto) allora

controlla se i tipi dei nodi figli, restituiti dalla tabella dei simboli al top dello stack, rispettano le  
indicazioni del type system  
 se il controllo ha avuto successo assegna al nodo il tipo indicato nel type system altrimenti lancia una TypeMismatchException con un opportuno messaggio

## Type System

### WhileOp

**IF** il tipo del nodo figlio Expression è *Boolean o Integer* **THEN** nodo.type = null  
**ELSE** lancia una TypeMismatchException con messaggio "You are trying to evaulate an expression which is not INTEGER or BOOLEAN"

### UnaryMinusOp

**IF** il tipo del nodo figlio Expression è *Boolean o Integer***THEN** nodo.type = *Integer* **ELSE** lancia una TypeMismatchException con messaggio "The UnaryMinus operator requires, as argument, a Boolean or an integer"

## RelationalOp

**IF** il tipo dei nodi figli SimpleExpression è lo stesso (*Boolean**o Integer* per entrambi)  
**THEN** nodo.type = *Boolean***ELSE** lancia una TypeMismatchException con messaggio "You can’t compare a *Tipo Prima Espressione* and a *Tipo Seconda Espressione* At instruction: *istruzione che ha generato l’errore*"

### NotOp

**IF** il tipo del nodo figlio Expression è *Boolean o Integer***THEN** nodo.type = *Boolean* **ELSE** lancia una TypeMismatchException con messaggio "The Not operator requires, as argument, a Boolean or an integer"

### MulOp

**IF** il tipo dei nodi figli SimpleExpression è per entrambi *Boolean* o *Integer***THEN IF** l’operatore usato è “&&” nodo.type = *Boolean* **ELSE** nodo.type = *Integer* **ELSE** lancia una TypeMismatchException con messaggio "You can’t use this operator *Operatore utilizzato* with a *Tipo Prima Espressione* and a *Tipo Seconda Espressione* At instruction: *istruzione che ha generato l’errore* "

## IfThenOp e IfThenElseOp

**IF** il tipo del nodo figlio Expression è *Boolean o Integer***THEN** nodo.type = *null* **ELSE** lancia una TypeMismatchException con messaggio "You are trying to evaulate an expression which is not INTEGER or BOOLEAN"

### AssingOp

**IF** il tipo del nodo relativo alla parte sinistra dell’assegnazione (identifier) è uguale al tipo del figlio Expression, parte destra dell’assegnazione **THEN** nodo.type =tipo dei due figli **ELSE** lancia una TypeMismatchException con messaggio "You can’t assign a *Tipo espressione*to a *Tipo Identificatore* variable. At instruction: *istruzione che ha generato l’errore* "

### AddOp

**IF** il tipo dei nodi figli SimpleExpression è per entrambi *Boolean* o *Integer***THEN IF** l’operatore usato è “||” nodo.type = *Boolean* **ELSE** nodo.type = *Integer* **ELSE** lancia una TypeMismatchException con messaggio "You can’t use this operator *Operatore utilizzato* with a *Tipo Prima Espressione* and a *Tipo Seconda Espressione* At instruction: *istruzione che ha generato l’errore* "