Semantic Analysis

Type Checking

הצהרה על משתנים והתאמות טיפוסים

$int\ a, b, c;$

$int\ a;$ ✗ **Already declared**

$int\ A[100];$

$real\ x, y;$

$struct\ T\ t, t2;$

….

$a = 1 + d;$ ✗ **Undeclared identifier d**

$A = 5;$ ✗ **Type mismatch**

$a[17] = 0;$ ✗ **Type mismatch**

$t = 15;$ ✗ **Type mismatch**

$a = 3.14;$ ✗ **Type mismatch**

$x = 3.14;$ ✓

$b = x;$ ✗ **Type mismatch**

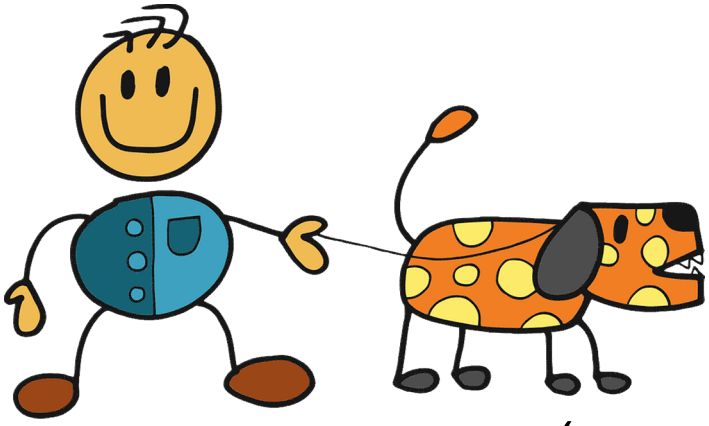
$y = a;$ ✓

טבלת הסמלים

- מבנה נתונים שמכיל אינפורמציה על כל המזהים בתכנית (למעשה רק ב-scope הנוכחי. נדבר על טווחי הכרה בהמשך הקורס).
- האינפורמציה כוללת:
  - טיפוס בסיסי: int, real, char, *void* וכד'.
  - טיפוס מורכב: מערך, מצביע, struct, פונקציה וכד'.
  - לטיפוסים מורכבים דרושה אינפורמציה נוספת.
    - Array(size, type) // מערך חד-מימדי
    - Pointer(type)
    - פונקציה- הטיפוס המוחזר, מספר הפרמטרים וטיפוסיהם.
    - ועוד

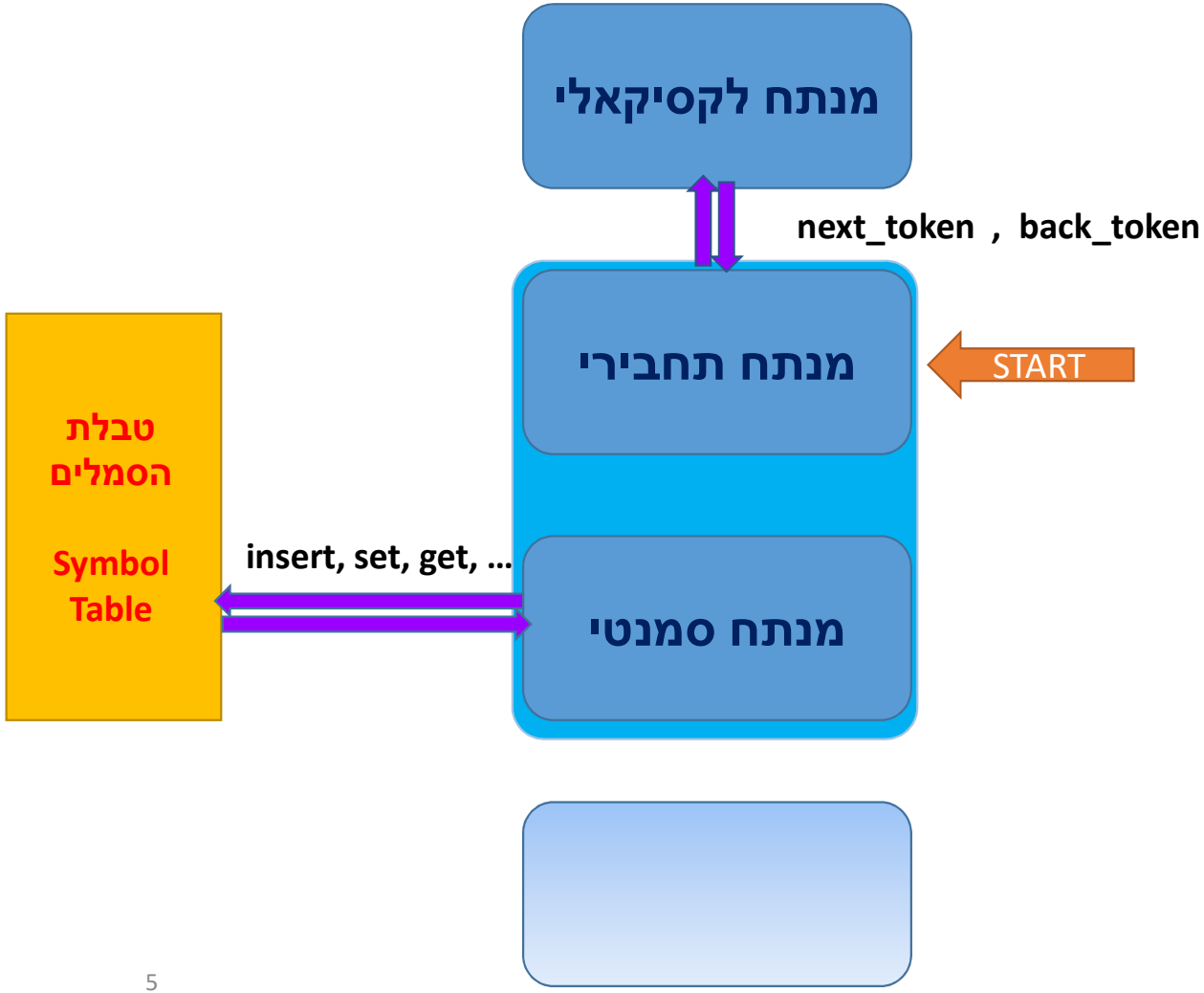
יחסי גומלין בין המנתח התחבירי והמנתח הסמנטי

- המנתח התחבירי והמנתח הסמנטי פועלים למעשה יחד ולא אחד אחרי השני.



- פונקציה $Parse\_X$:
- **תקבל** כפרמטר תכונות נורשות של X
- **תחזיר** תכונות נוצרות של אבא/אח של X (אשר תלויות בו)

- ניתן להשתמש במבנה attributes.


מנתח לקסיקאלי
next_token , back_token
מנתח תחבירי
START
טבלת הסמלים
Symbol Table
insert, set, get, ...
מנתח סמנטי

דוגמא לשילוב של פעולות סמנטיות בפונקציות של המנתח התחבירי

- דוגמא: סכימת תרגום מונחית תחביר להצהרות משתנים

$\boldsymbol{D} \rightarrow \boldsymbol{T} \quad \{L.type = T.type\} \quad \boldsymbol{L}$
$\boldsymbol{T} \rightarrow \boldsymbol{int} \quad \{T.type = int\}$
$\boldsymbol{T} \rightarrow \boldsymbol{real} \quad \{T.type = real\}$
$\boldsymbol{L} \rightarrow \boldsymbol{id} \quad \{id.type = L.type;\ insert(id.name, id.type)\}$
$\boldsymbol{L} \rightarrow \boldsymbol{id} \quad \{id.type = L.type\ ; insert(id.name, id.type); L_1.type = L.type\} \quad , \boldsymbol{L_1}$

- נוח לתכנן את סכימת התרגום מונחית התחביר עבור הדקדוק המקורי.
- לצורך כתיבת הפרסר בשיטת recursive descent יש להסיר רקורסיה שמאלית וגורמים שמאליים משותפים.

- הסכימה המעודכנת:

$\boldsymbol{D} \rightarrow \boldsymbol{T}\ \{L.type = T.type\}\ \ \boldsymbol{L}$
$\boldsymbol{T} \rightarrow \boldsymbol{int}\ \ \{T.type = int\}$
$\boldsymbol{T} \rightarrow \boldsymbol{real}\ \ \{T.type = real\}$
$\boldsymbol{L} \rightarrow \boldsymbol{id}\ \{id.type = L.type\ ; insert(id.name, id.type);\ \ L'.type = L.type\}\ \ \boldsymbol{L'}$
$\boldsymbol{L'} \rightarrow \boldsymbol{\varepsilon}\ \{\ \ \}$
$\boldsymbol{L'} \rightarrow \{L.type = L'.type\}\ \ ,\ \ \ \boldsymbol{L}$

$\boldsymbol{D} \rightarrow \boldsymbol{T}\ \{L.type = T.type\}\ \ \boldsymbol{L}$
$\boldsymbol{T} \rightarrow \boldsymbol{int}\ \ \{T.type = int\}$
$\boldsymbol{T} \rightarrow \boldsymbol{real}\ \ \{T.type = real\}$
$\boldsymbol{L} \rightarrow \boldsymbol{id}\ \{id.type = L.type\ ; insert(id.name, id.type);\ \ L'.type = L.type\}\ \ \boldsymbol{L'}$
$\boldsymbol{L'} \rightarrow \boldsymbol{\varepsilon}\ \{\ \ \}$
$\boldsymbol{L'} \rightarrow \{L.type = L'.type\}\ \ ,\ \ \ \boldsymbol{L}$

- $T.type$ - תכונה נוצרת - ערך מוחזר מ-$Parse\_T$
- $L.type$ – תכונה נורשת - מועברת כפרמטר ל- $Parse\_L$
- $L'.type$ – תכונה נורשת - מועברת כפרמטר ל- $Parse\_L'$
- $id.type$ – תכונה נורשת

$\boldsymbol{D} \rightarrow \boldsymbol{T}\ \{L.type = T.type\}\ \ \boldsymbol{L}$
$\boldsymbol{T} \rightarrow \boldsymbol{int}\ \ \{T.type = int\}$
$\boldsymbol{T} \rightarrow \boldsymbol{real}\ \ \{T.type = real\}$
$\boldsymbol{L} \rightarrow \boldsymbol{id}\ \{id.type = L.type\ ; insert(id.name, id.type);\ \ L'.type = L.type\}\ \ \boldsymbol{L'}$
$\boldsymbol{L'} \rightarrow \boldsymbol{\varepsilon}\ \ \{\ \ \}$
$\boldsymbol{L'} \rightarrow \{L.type = L'.type\}\ \ ,\ \ \ \ \boldsymbol{L}$

```
void Parse_D()
{
    attr T_type, L_type;

    T_type = Parse_T();

    L_type = T_type;

    Parse_L(L_type);
}
```

$\boldsymbol{D} \rightarrow \boldsymbol{T}\ \{L.type = T.type\}\ \ \boldsymbol{L}$
$\boldsymbol{T} \rightarrow \boldsymbol{int}\ \ \{T.type = int\}$
$\boldsymbol{T} \rightarrow \boldsymbol{real}\ \ \{T.type = real\}$
$\boldsymbol{L} \rightarrow \boldsymbol{id}\ \{id.type = L.type\ ; insert(id.name, id.type);\ \ L'.type = L.type\}\ \ \boldsymbol{L'}$
$\boldsymbol{L'} \rightarrow \boldsymbol{\varepsilon}\ \ \{\ \ \}$
$\boldsymbol{L'} \rightarrow \{L.type = L'.type\}\ \ \boldsymbol{L}$

```
attr Parse_T()
{
    attr T_type;

    t=next_Token();

    switch(t.kind) {
        case int:  T_type = int; break;
        case real: T_type = real; break;
        default: error();
        }
    return T_type;
}
```

$\boldsymbol{D} \rightarrow \boldsymbol{T}\ \{L.type = T.type\}\ \ \boldsymbol{L}$
$\boldsymbol{T} \rightarrow \boldsymbol{int}\ \ \{T.type = int\}$
$\boldsymbol{T} \rightarrow \boldsymbol{real}\ \ \{T.type = real\}$
$\boldsymbol{L} \rightarrow \boldsymbol{id}\ \{id.type = L.type\ ; insert(id.name, id.type);\ \ L'.type = L.type\}\ \ \boldsymbol{L'}$
$\boldsymbol{L'} \rightarrow \boldsymbol{\varepsilon}\ \ \{\ \ \}$
$\boldsymbol{L'} \rightarrow \{L.type = L'.type\}\ \ ,\ \ \ \ \boldsymbol{L}$

```
void Parse_L(attr L_type)
{
    attr L'_type;
    match(id);

    id.type = L_type;

    insert(id.name, id.type);

    L'_type = L_type;

    Parse_L'(L'_type);
}
```

$\boldsymbol{D} \rightarrow \boldsymbol{T}\ \{L.type = T.type\}\ \ \boldsymbol{L}$
$\boldsymbol{T} \rightarrow \ \boldsymbol{int}\ \ \{T.type = int\}$
$\boldsymbol{T} \rightarrow \boldsymbol{real}\ \ \{T.type = real\}$
$\boldsymbol{L} \rightarrow \ \boldsymbol{id}\ \{id.type = L.type\ ; insert(id.name, id.type);\ \ L'.type = L.type\}\ \ \boldsymbol{L'}$
$\boldsymbol{L'} \rightarrow \ \boldsymbol{\varepsilon}\ \{\ \ \}$
$\boldsymbol{L'} \rightarrow \ \{L.type = L'.type\}\ \ ,\ \ \ \ \boldsymbol{L}$

```
void Parse_L'(attr L'_in_type)
{
    attr L_type;

    t=next_Token();

    switch(t.kind) {
       case , :  L_type = L'_in_type;  Parse_L(L_type); break;
       case t in Follow(L') : back_token(); break;

       default: error();
       }
}
```

התאמת טיפוסים בפקודת השמה

- כלל גזירה של פקודת השמה: $\boldsymbol{ASSIGN\_STMT \rightarrow id = EXPR}$
- תכונות סמנטיות רלוונטיות: -- *שתיהן תכונות נוצרות. ידועות רק אחרי הגזירה*
- $id.type$
- $EXPR.type$
- הפעולה הסמנטית עבור כלל הגזירה תהיה בדיקת התאמת טיפוסים.
  - אם אותו טיפוס- יש התאמה.
  - אם לא אותו טיפוס- תלוי... גם כאשר יש התאמה, לפעמים פקודת ההשמה דורשת פעולות נוספות כמו casting.

טיפוס של ביטוי

- בהינתן ביטוי "בסיסי" (int_num, $real\_num$ או id): ניתן לדעת מהו הטיפוס בקלות.
  - int_num – טיפוס $integer$
  - $real\_num$- טיפוס $real$
  - id – ע"י בדיקה בטבלת הסמלים
- בהינתן ביטוי מורכב- יש לחשב מהו הטיפוס של הביטוי. **הטיפוס לא נתון באופן מפורש.**

טיפוס של ביטוי : דוגמאות

- מה הטיפוס של הביטוי a + b? (בהנחה שיש רק int ו-real)
  - אם a ו-b מטיפוס int : int
  - אם לפחות אחד מהם הוא real : real

- מה לגבי a / b?
  - אם שניהם int אז מדובר בפעולת חלוקה בשלמים ("div")
  - אם לפחות אחד מהם הוא real אז הטיפוס הוא real והאופרציה היא חלוקה.

דוגמא: הצהרות וביטויים פשוטים

- $P \rightarrow D; E$
- $D \rightarrow D; D \mid id: T$
- $T \rightarrow$ char | integer | real | array[num] of T | * T
- $E \rightarrow$ int_num | real_num | id | E[E] | E* | E mod E | E + E | E/E

$P \rightarrow D; E$

$D \rightarrow D; D$

$D \rightarrow id: T$ {inseret(id.name, T.type) }

$T \rightarrow char$ {T.type = char;}

$T \rightarrow integer$ {T.type = integer;}

$T \rightarrow real$ {T.type = real;}

טיפוס "מערך" כולל מידע
על גודל וטיפוס האיברים

$T \rightarrow$ array[num] of T1 {T.type = **array**(num.value, T1.type);}

$T \rightarrow$ *T1 {T.type = **pointer**(T1.type);}

טיפוס "מצביע" כולל מידע
על הטיפוס המוצבע

E→ int_num {E.type = integer;}

E→ real_num {E.type = real;}

חיפוש הכניסה המתאימה בטבלה

E→ id { entry = find(id.name);
E.type =
if (entry == null)
then type-error;
else get_type(entry);
}

הטיפוס בכניסה בטבלה

```
E → E1[ E2]      { E.type =
                        if ( (E2.type == integer)  &&  (E1.type == array(s,t) )
                              then  t;
                        else type-error;
                 }

E →  E1*         { E.type =
                      if (E1.type == pointer(t))
                            then  t ;
                      else type-error;
                 }
```

```
E → E1 mod E2     {E.type =
                        if ( (E1.type == integer) && (E2.type == integer) )
                              then integer;
                        else type-error;
                  }
```

```
E → E1 + E2     {E.type =
                  if ( (E1.type == type-error) || (E2.type == type-error) )
                        then type-error;
                  elseif ( (E1.type == real) || (E2.type == real) )
                        then real;
                  else
                        integer;
                }
```

```
E → E1 / E2     {E.type =
                    if ( (E1.type == type-error) || (E2.type == type-error) )
                        then type-error;
                    elseif ( (E1.type == integer) && (E2.type == integer) )
                        then integer;     // div
                    else
                        real;
                    }
```

דוגמא בקובץ נפרד