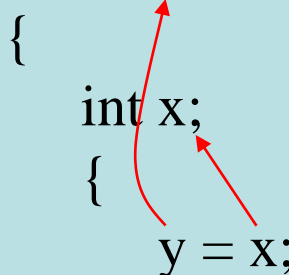


טבלאות סמלים

Static Scoping

- המופע של המשתנה אליו יש לגשת הוא המופע הקרוב ביותר בשרשרת הקינון הסטטי.

```
{
  int x,y;
  {
    int x;
    {
      y = x;
    }
    { ... }
  }
}
```



- ניתן לשייך כל גישה למופע מסוים.
- מבוצע בזמן קומפילציה.

טבלת סמלים

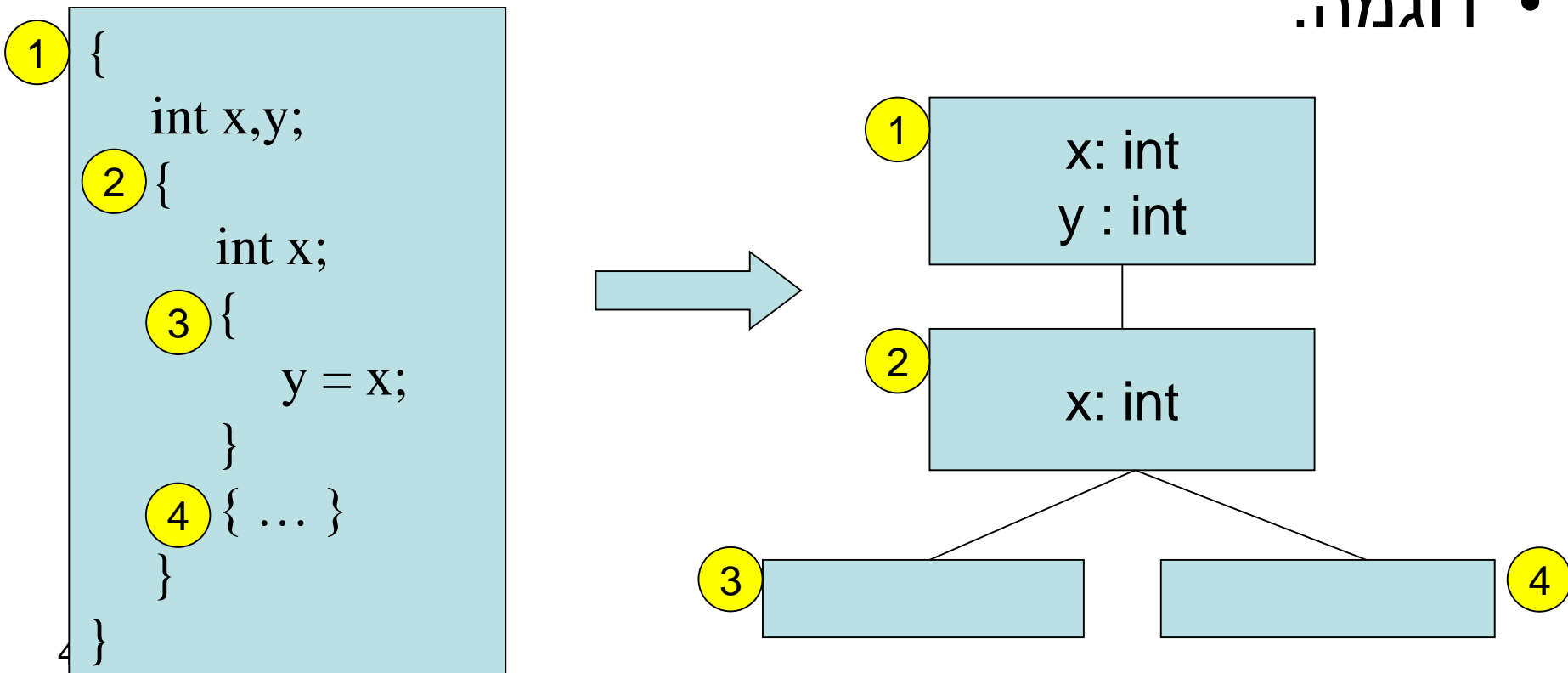
- מבנה נתונים המשמש את הקומפיילר.
- מחזיק מידע על הסמלים בתכנית:
 - **סמלים**: שמות משתנים, פונקציות, ...
 - צריך לתמוך ב-static scoping.
 - צריך לאפשר חיפוש יעיל של סמלים.

מימוש טבלאות הסמלים

- הרעיון: לכל scope ניצור טבלה נפרדת

– המבנה המתקבל הוא עץ של טבלאות.

- דוגמה:



מימוש טבלאות הסמלים - המשך

- כדי למצוא משתנה א:

- חפש בטבלת הסמלים של ה-scope הנוכחי. אם מצאת, עצור. אחרת,

- חפש בטבלת הסמלים של האבא של ה-scope הנוכחי. אם מצאת, עצור. אחרת,

- המשך את החיפוש עד לשורש

- מה מכילה טבלת הסמלים בשורש העץ?

בניית עץ טבלאות הסמלים

- הבנייה מתבצעת בזמן קומפילציה .
- נשתמש במחסנית שמחזיקה את שרשרת הטבלאות מה-
scope הנוכחי עד לשורש.
 - על מנת למצוא משתנה, מספיק לבדוק רק טבלאות במחסנית
 - מדוע?
 - בכניסה ל-scope ניצור טבלה חדשה ונדחוף למחסנית.
 - ביציאה מ-scope נוציא טבלה מראש המחסנית.

סכימת תרגום

- נניח שמבנה הרשומה בטבלת הסמלים הוא:

name type offset

המיקום היחסי
באיזור ה-locals
של רשומת
ההפעלה

- נשתמש בשתי מחסניות:

– **tables**: מחסנית טבלאות הסמלים.

– **offsets**: מחסנית של ה-offset-ים הנוכחיים
בכל scope במסלול לשורש.

דוגמה

```
1 {  
    int x,y;  
2 {  
    int x;  
3 {  
    bool w;  
    y = x;  
    }  
4 {  
    bool z;  
    }  
    }  
    int f;  
}
```

- עבור קטע הקוד הבא:

נניח לשם פשטות הדוגמה, כי משתנים מסוג
INT, BOOL תופסים 1 בתים בזיכרון.

- נראה כיצד ייראו מחסניות הטבלאות

וה- offsets בכל שלב.


```

1 {
  → int x,y;
2 {
  int x;
3   bool w;
  y = x;
4 }
  bool z;
}
int f;
}

```

דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
offsets בכניסה ל- scope1:

Name	Type	Offset

tables stack

0	

offsets stack

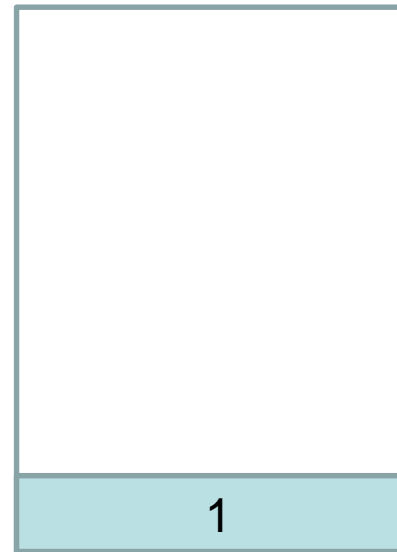
ה offset אליו
ייכנס המשתנה
הבא ב scope1

דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
offsets ב-scope1:

Name	Type	Offset
x	int	0

tables stack



offsets stack

ה offset אליו
ייכנס המשתנה
הבא ב scope1

```

1 {
  → int x,y;
2 {
  int x;
3   bool w;
  y = x;
4 }
  bool z;
}
int f;
}

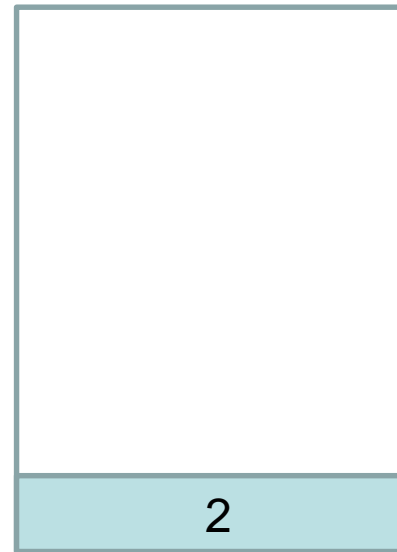
```

דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
offsets לפני scope2:

Name	Type	Offset
x	int	0
y	int	1

tables stack



offsets stack

ה offset אליו
ייכנס המשתנה
הבא ב scope1

דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
offsets בכניסה ל- scope2:

```
1 {  
  int x,y;  
2 {  
  → int x;  
3   bool w;  
   y = x;  
4 }  
  bool z;  
}  
int f;  
}
```

Name	Type	Offset
x	int	0
y	int	1

tables stack

2
2

offsets stack

ה offset אליו
ייכנס המשתנה
הבא ב scope2

ה offset אליו
ייכנס המשתנה
הבא ב scope1

דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
:scope3 offsets

```
1 {  
  int x,y;  
2 {  
  → int x;  
3   bool w;  
   y = x;  
4 }  
  bool z;  
}  
int f;  
}
```

Name	Type	Offset
x	int	2

Name	Type	Offset
x	int	0
y	int	1

tables stack

3
2

offsets stack

דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
offsets בכניסה ל- scope3:

Name	Type	Offset
x	int	2

Name	Type	Offset
x	int	0
y	int	1

tables stack

3
3
2

offsets stack

דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
offsets לפני היציאה מ-scope3:

Name	Type	Offset
w	bool	3

Name	Type	Offset
x	int	2

Name	Type	Offset
x	int	0
y	int	1

tables stack

4
3
2

offsets stack

דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
offsets לאחר היציאה מ-scope3:

```
1 {  
  int x,y;  
2 {  
  int x;  
3  bool w;  
  y = x;  
4 }  
  bool z;  
}  
int f;  
}
```

Name	Type	Offset
x	int	2

Name	Type	Offset
x	int	0
y	int	1

tables stack

3
2

offsets stack

דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
offsets בכניסה ל- scope4:

```
1 {  
  int x,y;  
2 {  
  int x;  
3  bool w;  
  y = x;  
4 }  
  bool z;  
}  
int f;  
}
```

Name	Type	Offset
x	int	2

Name	Type	Offset
x	int	0
y	int	1

tables stack

3
3
2

offsets stack

דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
offsets לפני היציאה מ-scope4:

```
1 {  
  int x,y;  
2 {  
  int x;  
3  bool w;  
  y = x;  
4 {  
  bool z;  
  }  
  int f;  
}
```

Name	Type	Offset
z	bool	3

Name	Type	Offset
x	int	2

Name	Type	Offset
x	int	0
y	int	1

tables stack

4
3
2

offsets stack

דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
offsets לאחר היציאה מ-scope4:

```
1 {  
  int x,y;  
2 {  
  int x;  
3  bool w;  
  y = x;  
4 }  
  bool z;  
}  
→ int f;  
}
```

Name	Type	Offset
x	int	2

Name	Type	Offset
x	int	0
y	int	1

tables stack

3
2

offsets stack

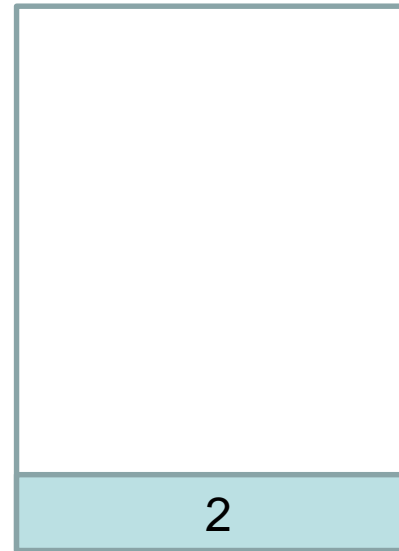
דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
offsets לאחר היציאה מ-scope2:

```
1 {  
  int x,y;  
2 {  
  int x;  
3  bool w;  
  y = x;  
4 }  
  bool z;  
}  
int f;  
}
```

Name	Type	Offset
x	int	0
y	int	1

tables stack



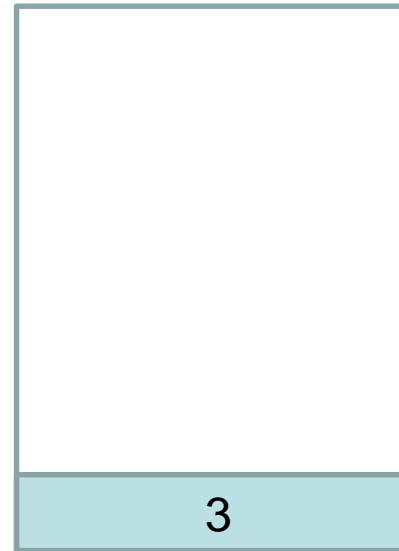
offsets stack

דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
offsets לפני היציאה מ-scope1:

Name	Type	Offset
x	int	0
y	int	1
f	int	2

tables stack

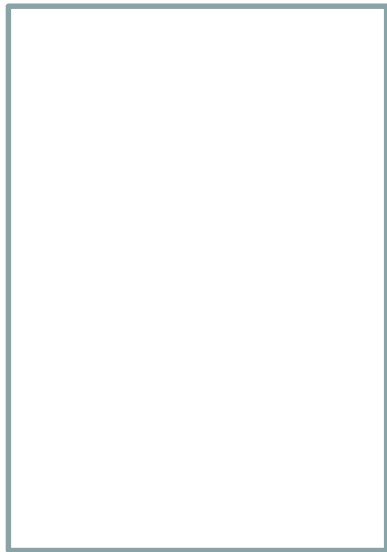



offsets stack

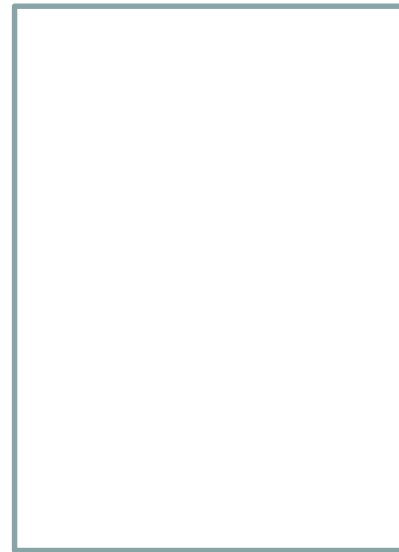
דוגמה - המשך

- מצב מחסנית טבלאות הסמלים ומחסנית ה-
offsets לאחר היציאה מ-scope1:

```
1 {  
  int x,y;  
2 {  
  int x;  
3 {  
  bool w;  
  y = x;  
4 {  
  bool z;  
  }  
  }  
  int f;  
}
```



tables stack



offsets stack

סכימת תרגום - המשך

- נניח שקיימות הפונקציות הבאות:

– **Maketable** (parent) :

יוצרת טבלה חדשה ריקה שהיא בת של parent בעץ.

– **Insert** (table, name, type, offset) :

מכניסה משתנה לטבלת הסמלים.

– **push, pop, top** :

הפעולות הסטנדרטיות למחסנית.

סכימת תרגום - המשך

$\text{Prog} \rightarrow \underline{\text{int}} \underline{\text{main}} () \{ \text{St} \}$

$\text{St} \rightarrow \underline{\text{vartype}} \underline{\text{id}} ;$

$\text{St} \rightarrow \{ \text{St} \}$

$\text{St} \rightarrow \underline{\text{print}} \underline{\text{id}} ;$

- Prog : המשתנה התחילי - גוזר תוכנית שלמה.

- St : גוזר פקודה או רצף פקודות.

– הערה: הדקדוק המוצג הינו חלקי ביותר.


```
Prog → int main ( ) { St }  
St → vartype id ;  
St → { St }  
St → print id ;
```

כיצד להתחיל את המבנים?

- Prog → int main () { **M** St }
 { pop(tables);
 pop(offsets); }

- **M** → ϵ
 { t = maketable(null);
 push(t, tables);
 push(0, offsets); }

```
Prog → int main ( ) { St }  
St → vartype id ;  
St → { St }  
St → print id ;
```

פתיחת Scope

- $St \rightarrow \{ N St \}$
 { pop(**tables**);
 pop(**offsets**); }
- $N \rightarrow \varepsilon$
 { t = maketable (top(**tables**));
 push (t, **tables**);
 push (top(**offsets**), **offsets**); }

```
Prog → int main ( ) { St }  
St → vartype id ;  
St → { St }  
St → print id ;
```

הוספת משתנה

- St → vartype id ;
 { **insert** (top(**tables**), id.name, vartype.type, top(**offsets**));
 top (**offsets**) += vartype.size; }

```
Prog → int main ( ) { St }  
St → vartype id ;  
St → { St }  
St → print id ;
```

הדפסת משתנה

• היינו רוצים לבדוק האם משתנה הוגדר לפני הדפסה.

- St → print id ;
 { found = false;
 for (int i = 0; i < tables.size(); i++) {
 t = tables.get(i); // get i'th table from top
 if (t.contains(id.name)) {
 found = true; break;
 }
 }
 }
 if (found == false)
 error("use of undef variable", id.name);
 }

Scoping של פונקציות

- משתנים הם לא הדבר היחיד שצריך לוודא את הטיפוס שלו

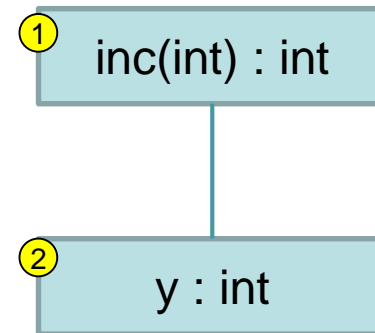
```
doSomething(x,y+2,89);
```

- נרצה לוודא:
 - doSomething קיימת
 - יש לה שלושה ארגומנטים
 - הארגומנטים שהועברו תואמים את הטיפוסים המוצהרים

פונקציות בטבלת הסמלים

- זה אומר שנרצה לרשום פונקציות בסקופ בו הן מוצהרות.

```
① int inc(int x)
② {
    int y = x + 1;
    return y;
}
```



דוגמה עם פונקציות

```
➔ ① int inc(int x)
    ② {
        int y = x + 1;
        return y;
    }

① int add(int x, int y)
    ③ {
        int inc = x;
        inc = inc + y;
        return inc;
    }
```

Name	Type	Offset

tables stack

0

offsets stack

דוגמה עם פונקציות

```
➔ ① int inc(int x)
  ② {
    int y = x + 1;
    return y;
  }

① int add(int x, int y)
  ③ {
    int inc = x;
    inc = inc + y;
    return inc;
  }
```

Name	Type	Offset
inc	int -> int	n/a

tables stack

0

לפונקציה אין
offset אז לא
מקדמים את המונה

offsets stack

דוגמה עם פונקציות

① `int inc(int x)`

➔ ② {
 `int y = x + 1;`
 `return y;`
}

① `int add(int x, int y)`

③ {
 `int inc = x;`
 `inc = inc + y;`
 `return inc;`
}

Name	Type	Offset
x	int	-1
Name	Type	Offset
inc	int -> int	n/a

tables stack

פרמטרים יושבים
לפני המשתנים
ברשימת ההפעלה

0

0

offsets stack

דוגמה עם פונקציות

```
①int inc(int x)
②{
  int y = x + 1;
  return y;
}

①int add(int x, int y)
③{
  int inc = x;
  inc = inc + y;
  return inc;
}
```

Name	Type	Offset
x	int	-1
y	int	0
Name	Type	Offset
inc	int -> int	n/a

tables stack

1
0

offsets stack

דוגמה עם פונקציות

```
①int inc(int x)
②{
    int y = x + 1;
    return y;
}
→
①int add(int x, int y)
③{
    int inc = x;
    inc = inc + y;
    return inc;
}
```

Name	Type	Offset
inc	int -> int	n/a

tables stack

0

offsets stack

דוגמה עם פונקציות

```
①int inc(int x)
②{
    int y = x + 1;
    return y;
}

➡①int add(int x, int y)
③{
    int inc = x;
    inc = inc + y;
    return inc;
}
```

Name	Type	Offset
inc	int -> int	n/a
add	int, int -> int	n/a

tables stack

0

offsets stack

דוגמה עם פונקציות

```
①int inc(int x)
②{
    int y = x + 1;
    return y;
}

①int add(int x, int y)
➔ ③{
    int inc = x;
    inc = inc + y;
    return inc;
}
```

Name	Type	Offset
x	int	-1
y	int	-2
Name	Type	Offset
inc	int -> int	n/a
add	int, int -> int	n/a

tables stack

0
0

offsets stack

דוגמה עם פונקציות

```
①int inc(int x)
②{
    int y = x + 1;
    return y;
}

①int add(int x, int y)
③{
    int inc = x;
    inc = inc + y;
    return inc;
}
```

Name	Type	Offset
x	int	-1
y	int	-2
inc	int	0
Name	Type	Offset
inc	int -> int	n/a
add	int, int -> int	n/a

tables stack

1
0


מה יקרה עכשיו אם
תופיע הקריאה
?inc(8)

offsets stack

דוגמה עם פונקציות

```
①int inc(int x)
②{
    int y = x + 1;
    return y;
}

①int add(int x, int y)
③{
    int inc = x;
    inc = inc + y;
    return inc;
}
```



Name	Type	Offset
inc	int -> int	n/a
add	int, int -> int	n/a

tables stack



offsets stack

בשבוע הבא

- ייצור קוד ביניים