40-414 Compiler Design

Semantic Analysis
&
Symbol Table Management

Lecture 7

Static versus Dynamic Checking

- *Static checking*: the compiler enforces programming language's *static semantics*
  - Program properties that can be checked at compile time

- *Dynamic semantics*: checked at run time
  - Compiler generates verification code to enforce programming language's dynamic semantics

Static Checking Examples

- Type checks: *in A := B + C, all operands should have the same type*
- Flow-of-control checks: *check whether a e.g. break statement has somewhere to return control.*
- Uniqueness checks: *In some languages names must be unique*
- Named-related checks: *In ADA for loops can have name, and it must appear twice (before the for keyword and before the end statement).*

Type Checks, Overloading, and Coercion

```
int op(int), op(float);
int f(float);
int a, c[10], d;

d = c+d;        error: invalid conversion from ‘int*’ to ‘int’
                d = d + c;
                      ↑
*d = a;         error: invalid type argument of unary ‘*’
                *d = a;
                 ↑
a = op(d);              // OK: overloading (C++)

a = f(d);               // OK: coercion of d to float
```

Flow-of-Control Checks

```
myfunc()
{ …
  break; // ERROR
}
```

```
myfunc()
{ …
  while (n)
  { …
    if (i>10)
      break; // OK
  }
}
```

```
myfunc()
{ …
  switch (a)
  { case 0:
      …
      break; // OK
    case 1:
      …
  }
}
```

Uniqueness Checks

```
myfunc()
{ int i, j, i; // ERROR
  …
}
```

```
struct myrec
{ int name;
};
struct myrec // ERROR
{ int id;
};
```

```
myfunc(int a, int a) // ERROR
{    …
}
```

Nested Related Checks

```
LoopB: for (int J = 0; J < m; J++)
        {
                LoopA: for (int I = 0; I < n; I++)
                 { ...
                        if (a[I] == 0)
                         break LoopB; // Java labeled loop
          ...
                 }
        }
```

One-Pass versus Multi-Pass Static Checking

- One-pass compiler: static checking in C, Pascal, Fortran, and many other languages is performed in one pass while intermediate code is generated
  (Influences design of a language: placement constraints)

- Multi-pass compiler: static checking in Ada, Java, and C# is performed in a separate phase, sometimes by traversing a syntax tree multiple times

Dynamic Checking

- A piece of object code is added to the compiled program to perform the checking in the execution time

- Example:
  var a[10] int; …; read (I); a(I) := 0;

- Generated code is as such the last statement would have been:

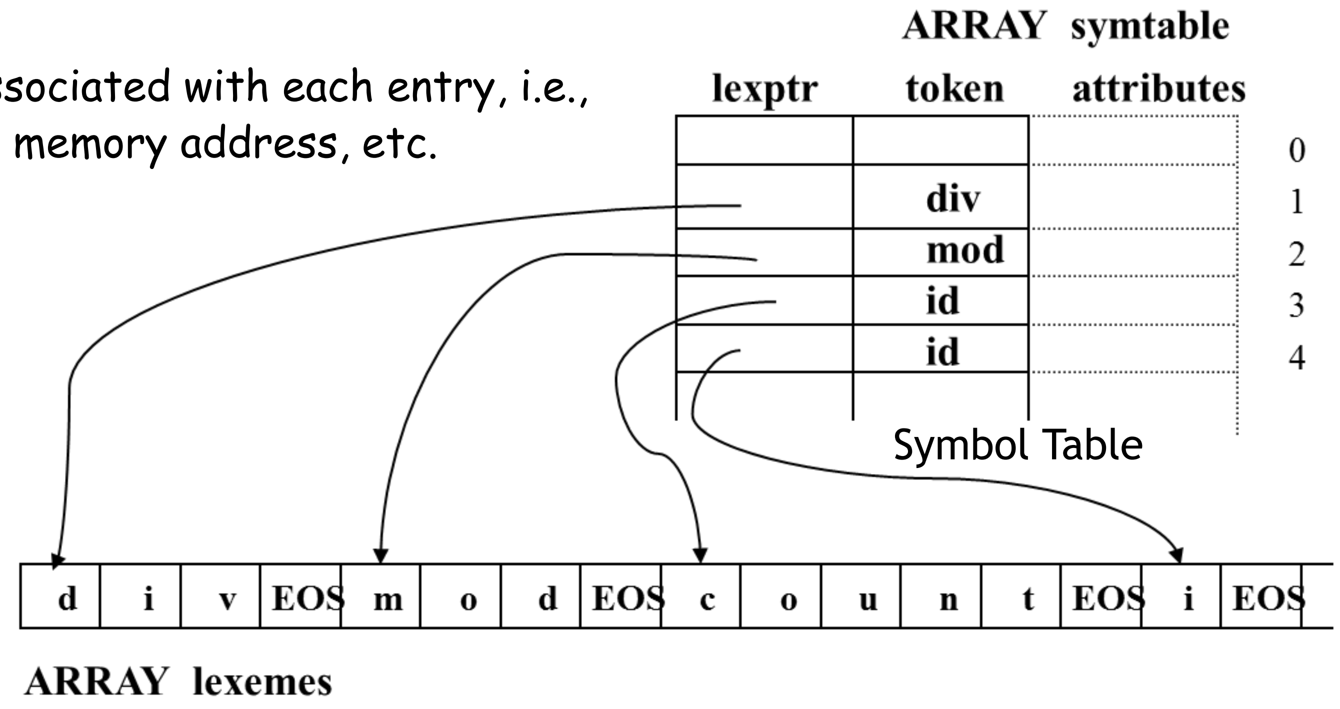
  If I <= 10 then a(I) := 0 else print (“subscript out of range error”)

Symbol Table Management

OPERATIONS: Insert (string, token_ID)
Lookup (string)

NOTICE: Reserved words are placed into symbol table for easy lookup

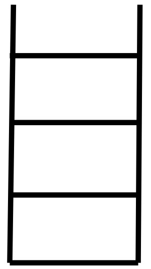
Attributes may be associated with each entry, i.e., typing info, size info, memory address, etc.



Example

```
program sort(input, output);
     var a : array [0 .. 10] of integer; x : integer;

     procedure readarray;
          var i : integer;
          begin … a … end;

     procedure exchange( i, j, : integer);
          begin
                  x := a[i]; a[i] := a[j]; a[j] := x
          end

     procedure quicksort(m, n: integer);
          var k, v : integer;

          function partition(y, z: integer) : integer;
          var i, j : integer;
          begin   … a …
                  … v …
                  … exchange(i, j); …
          end { partition };
          begin … end { quicksort }
     begin … end { sort }.
```

Example (Cont.)



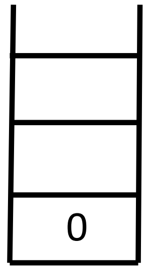
Scope Stack

symbol table

lexeme	type	attributes	
			0
			1
			2
			3
			4
			5
			6
			7
			8
			9
			10
			11
			12

```
program sort(input, output);
        var a : array [0 .. 10] of integer; x : integer;
        procedure readarray;
                var i : integer;
                begin … a … end;
        procedure exchange( i, j, : integer);
                begin
                            x := a[i]; a[i] := a[j]; a[j] := x
                end
        procedure quicksort(m, n: integer);
                var k, v : integer;
                function partition(y, z: integer) : integer;
                            var i, j : integer;
                            begin       … a …
                                        … v …
                                        … exchange(i, j); …
                end { partition };
                begin … end { quicksort }
        begin … end { sort }.
```

Example (Cont.)



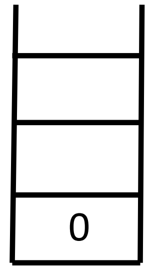
Scope Stack

symbol table

lexeme	type	attributes	
sort	-		0
			1
			2
			3
			4
			5
			6
			7
			8
			9
			10
			11
			12

```
program sort(input, output);
        var a : array [0 .. 10] of integer; x : integer;
        procedure readarray;
                var i : integer;
                begin … a … end;
        procedure exchange( i, j, : integer);
                begin
                            x := a[i]; a[i] := a[j]; a[j] := x
                end
        procedure quicksort(m, n: integer);
                var k, v : integer;
                function partition(y, z: integer) : integer;
                            var i, j : integer;
                            begin        … a …
                                         … v …
                                         … exchange(i, j); …
                end { partition };
                begin … end { quicksort }
        begin … end { sort }.
```

Example (Cont.)



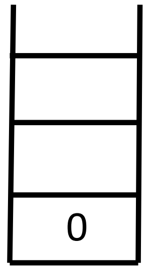
Scope Stack

symbol table

	lexeme	**type**	**attributes**
0	sort	-	
1	a	int	
2	x	int	
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

```
program sort(input, output);
        var a : array [0 .. 10] of integer; x : integer;
        procedure readarray;
                var i : integer;
                begin … a … end;
        procedure exchange( i, j, : integer);
                begin
                             x := a[i]; a[i] := a[j]; a[j] := x
                end
        procedure quicksort(m, n: integer);
                var k, v : integer;
                function partition(y, z: integer) : integer;
                             var i, j : integer;
                             begin        … a …
                                          … v …
                                          … exchange(i, j); …
                end { partition };
                begin … end { quicksort }
        begin … end { sort }.
```

Example (Cont.)



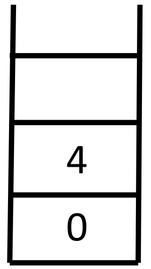
Scope Stack

symbol table

lexeme	**type**	**attributes**	
sort	-		0
a	int		1
x	int		2
readarray			3
			4
			5
			6
			7
			8
			9
			10
			11
			12

```
program sort(input, output);
        var a : array [0 .. 10] of integer; x : integer;
        procedure readarray;
                var i : integer;
                begin … a … end;
        procedure exchange( i, j, : integer);
                begin
                             x := a[i]; a[i] := a[j]; a[j] := x
                end
        procedure quicksort(m, n: integer);
                var k, v : integer;
                function partition(y, z: integer) : integer;
                             var i, j : integer;
                             begin       … a …
                                         … v …
                                         … exchange(i, j); …
                end { partition };
                begin … end { quicksort }
        begin … end { sort }.
```

Example (Cont.)



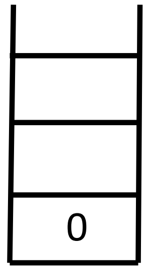
Scope Stack

symbol table

lexeme	**type**	**attributes**	
sort	-		0
a	int		1
x	int		2
readarray			3
i	int		4
			5
			6
			7
			8
			9
			10
			11
			12

```
program sort(input, output);
    var a : array [0 .. 10] of integer; x : integer;
    procedure readarray;
        var i : integer;
        begin … a … end;
    procedure exchange( i, j, : integer);
        begin
              x := a[i]; a[i] := a[j]; a[j] := x
        end
    procedure quicksort(m, n: integer);
        var k, v : integer;
        function partition(y, z: integer) : integer;
              var i, j : integer;
              begin    … a …
                       … v …
                       … exchange(i, j); …
        end { partition };
        begin … end { quicksort }
    begin … end { sort }.
```

Example (Cont.)



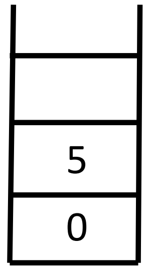
Scope Stack

symbol table

lexeme	type	attributes	
sort	-		0
a	int		1
x	int		2
readarray			3
exchange			4
			5
			6
			7
			8
			9
			10
			11
			12

```
program sort(input, output);
        var a : array [0 .. 10] of integer; x : integer;
        procedure readarray;
                var i : integer;
                begin … a … end;
        procedure exchange( i, j, : integer);
                begin
                        x := a[i]; a[i] := a[j]; a[j] := x
                end
        procedure quicksort(m, n: integer);
                var k, v : integer;
                function partition(y, z: integer) : integer;
                        var i, j : integer;
                        begin       … a …
                                    … v …
                                    … exchange(i, j); …
                end { partition };
                begin … end { quicksort }
        begin … end { sort }.
```

Example (Cont.)



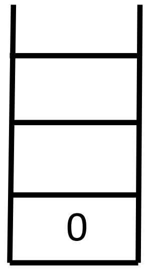
Scope Stack

symbol table

lexeme	**type**	**attributes**	
sort	-		0
a	int		1
x	int		2
readarray			3
exchange			4
i	int		5
j	int		6
			7
			8
			9
			10
			11
			12

```
program sort(input, output);
    var a : array [0 .. 10] of integer; x : integer;
    procedure readarray;
        var i : integer;
        begin … a … end;
    procedure exchange( i, j, : integer);
        begin
              x := a[i]; a[i] := a[j]; a[j] := x
        end
    procedure quicksort(m, n: integer);
        var k, v : integer;
        function partition(y, z: integer) : integer;
              var i, j : integer;
              begin       … a …
                          … v …
                          … exchange(i, j); …
        end { partition };
        begin … end { quicksort }
    begin … end { sort }.
```

Example (Cont.)



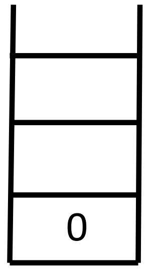
Scope Stack

symbol table

lexeme	type	attributes	
sort	-		0
a	int		1
x	int		2
readarray			3
exchange			4
			5
			6
			7
			8
			9
			10
			11
			12

```
program sort(input, output);
    var a : array [0 .. 10] of integer; x : integer;
    procedure readarray;
        var i : integer;
        begin … a … end;
    procedure exchange( i, j, : integer);
        begin
                x := a[i]; a[i] := a[j]; a[j] := x
        end
    procedure quicksort(m, n: integer);
        var k, v : integer;
        function partition(y, z: integer) : integer;
                var i, j : integer;
                begin       … a …
                            … v …
                            … exchange(i, j); …
        end { partition };
        begin … end { quicksort }
    begin … end { sort }.
```

Example (Cont.)



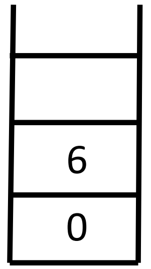
Scope Stack

symbol table

lexeme	type	attributes	
sort	-		0
a	int		1
x	int		2
readarray			3
exchange			4
quicksort			5
			6
			7
			8
			9
			10
			11
			12

```
program sort(input, output);
        var a : array [0 .. 10] of integer; x : integer;
        procedure readarray;
                var i : integer;
                begin … a … end;
        procedure exchange( i, j, : integer);
                begin
                             x := a[i]; a[i] := a[j]; a[j] := x
                end
        procedure quicksort(m, n: integer);
                var k, v : integer;
                function partition(y, z: integer) : integer;
                             var i, j : integer;
                             begin        … a …
                                          … v …
                                          … exchange(i, j); …
                end { partition };
                begin … end { quicksort }
        begin … end { sort }.
```

Example (Cont.)



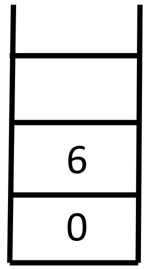
Scope Stack

symbol table

	lexeme	**type**	**attributes**
0	sort	-	
1	a	int	
2	x	int	
3	readarray		
4	exchange		
5	quicksort		
6	m	int	
7	n	int	
8	k	int	
9	v	int	
10			
11			
12			

```
program sort(input, output);
        var a : array [0 .. 10] of integer; x : integer;
        procedure readarray;
                var i : integer;
                begin … a … end;
        procedure exchange( i, j, : integer);
                begin
                             x := a[i]; a[i] := a[j]; a[j] := x
                end
        procedure quicksort(m, n: integer);
                var k, v : integer;
                function partition(y, z: integer) : integer;
                             var i, j : integer;
                             begin        … a …
                                          … v …
                                          … exchange(i, j); …
                end { partition };
                begin … end { quicksort }
        begin … end { sort }.
```

Example (Cont.)



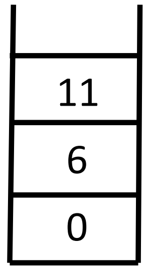
Scope Stack

symbol table

lexeme	**type**	**attributes**	
sort	-		0
a	int		1
x	int		2
readarray	-		3
exchange	-		4
quicksort	-		5
m	int		6
n	int		7
k	int		8
v	int		9
partition			10
			11
			12

```
program sort(input, output);
    var a : array [0 .. 10] of integer; x : integer;
    procedure readarray;
        var i : integer;
        begin … a … end;
    procedure exchange( i, j, : integer);
        begin
                x := a[i]; a[i] := a[j]; a[j] := x
        end
    procedure quicksort(m, n: integer);
        var k, v : integer;
        function partition(y, z: integer) : integer;
                var i, j : integer;
                begin      … a …
                           … v …
                           … exchange(i, j); …
        end { partition };
        begin … end { quicksort }
    begin … end { sort }.
```

Example (Cont.)



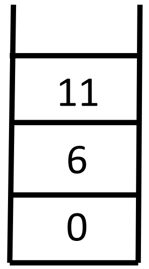
Scope Stack

symbol table

lexeme	**type**	**attributes**	
sort	-		0
a	int		1
x	int		2
readarray	-		3
exchange	-		4
quicksort	-		5
m	int		6
n	int		7
k	int		8
v	int		9
partition			10
y	int		11
z	int		12

```
program sort(input, output);
        var a : array [0 .. 10] of integer; x : integer;
        procedure readarray;
                var i : integer;
                begin … a … end;
        procedure exchange( i, j, : integer);
                begin
                            x := a[i]; a[i] := a[j]; a[j] := x
                end
        procedure quicksort(m, n: integer);
                var k, v : integer;
                function partition(y, z: integer) : integer;
                            var i, j : integer;
                            begin       … a …
                                        … v …
                                        … exchange(i, j); …
                end { partition };
                begin … end { quicksort }
        begin … end { sort }.
```

Example (Cont.)



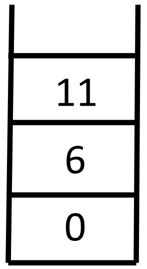
Scope Stack

symbol table

	lexeme	**type**	**attributes**
0	sort	-	
1	a	int	
2	x	int	
3	readarray	-	
4	exchange	-	
5	quicksort	-	
6	m	int	
7	n	int	
8	k	int	
9	v	int	
10	partition	int	
11	y	int	
12	z	int	

```
program sort(input, output);
    var a : array [0 .. 10] of integer; x : integer;
    procedure readarray;
        var i : integer;
        begin … a … end;
    procedure exchange( i, j, : integer);
        begin
                x := a[i]; a[i] := a[j]; a[j] := x
        end
    procedure quicksort(m, n: integer);
        var k, v : integer;
        function partition(y, z: integer) : integer;
                var i, j : integer;
                begin      … a …
                           … v …
                           … exchange(i, j); …
        end { partition };
        begin … end { quicksort }
    begin … end { sort }.
```

Example (Cont.)



Scope Stack

symbol table

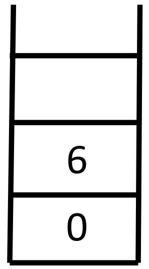
lexeme	type	attributes	
sort	-		0
a	int		1
x	int		2
readarray	-		3
exchange	-		4
quicksort	-		5
m	int		6
n	int		7
k	int		8
v	int		9
partition	int		10
y	int		11
z	int		12

```
program sort(input, output);
    var a : array [0 .. 10] of integer; x : integer;
    procedure readarray;
        var i : integer;
        begin … a … end;
    procedure exchange( i, j, : integer);
        begin
                x := a[i]; a[i] := a[j]; a[j] := x
        end
    procedure quicksort(m, n: integer);
        var k, v : integer;
        function partition(y, z: integer) : integer;
                var i, j : integer; ←
                begin      … a …
                           … v …
                           … exchange(i, j); …
        end { partition };
        begin … end { quicksort }
    begin … end { sort }.
```

i	int		13
j	int		14

25

Example (Cont.)



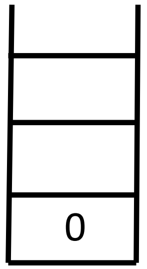
Scope Stack

symbol table

lexeme	type	attributes	
sort	-		0
a	int		1
x	int		2
readarray	-		3
exchange	-		4
quicksort	-		5
m	int		6
n	int		7
k	int		8
v	int		9
partition	int		10
			11
			12

```
program sort(input, output);
          var a : array [0 .. 10] of integer; x : integer;
          procedure readarray;
                    var i : integer;
                    begin … a … end;
          procedure exchange( i, j, : integer);
                    begin
                                    x := a[i]; a[i] := a[j]; a[j] := x
                    end
          procedure quicksort(m, n: integer);
                    var k, v : integer;
                    function partition(y, z: integer) : integer;
                                    var i, j : integer;
                                    begin       … a …
                                                … v …
                                                … exchange(i, j); …
                    end { partition };
                    begin … end { quicksort }
          begin … end { sort }.
```

Example (Cont.)



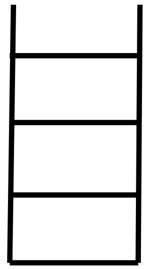
Scope Stack

symbol table

lexeme	**type**	**attributes**	
sort	-		0
a	int		1
x	int		2
readarray	-		3
exchange	-		4
quicksort	-		5
			6
			7
			8
			9
			10
			11
			12

```
program sort(input, output);
        var a : array [0 .. 10] of integer; x : integer;
        procedure readarray;
                var i : integer;
                begin … a … end;
        procedure exchange( i, j, : integer);
                begin
                            x := a[i]; a[i] := a[j]; a[j] := x
                end
        procedure quicksort(m, n: integer);
                var k, v : integer;
                function partition(y, z: integer) : integer;
                            var i, j : integer;
                            begin       … a …
                                        … v …
                                        … exchange(i, j); …
                end { partition };
                begin … end { quicksort } ←
        begin … end { sort }.
```

Example (Cont.)



Scope Stack

symbol table

lexeme	type	attributes	
			0
			1
			2
			3
			4
			5
			6
			7
			8
			9
			10
			11
			12

```
program sort(input, output);
        var a : array [0 .. 10] of integer; x : integer;
        procedure readarray;
                var i : integer;
                begin … a … end;
        procedure exchange( i, j, : integer);
                begin
                            x := a[i]; a[i] := a[j]; a[j] := x
                end
        procedure quicksort(m, n: integer);
                var k, v : integer;
                function partition(y, z: integer) : integer;
                            var i, j : integer;
                            begin       … a …
                                        … v …
                                        … exchange(i, j); …
                end { partition };
                begin … end { quicksort }
        begin … end { sort }.
```

Question?

Which one of the modules detects the error in the given Pascal piece of code, and when?

```
type a = array[1..10] of integer;
var i : integer; b : a;
i : = 11;
b[i] = 25;
```

○ Lexical Analysis in Compile time

○ Syntax Analysis in Compile time

○ Semantic Analysis in Compile time

○ Generated Code in Runtime

Question?

What is the state of symbol table and scope stack at the time of compiling lines 7 and 13?

```
1    Program S()
2              Var a[1..5], c, real
3              Procedure R(m: integer)
4                      Var b[1..5] integer
5                      Procedure E()
6                               Var I, c[1..3] integer
7                               c(3) := a (2) + b (1)
8                      End E
9                      Function Q(n: integer): integer
10                              Var a integer
11                              Procedure P()
12                                       Var b real
13                                       b := a + c
14                              End P
15                     End Q
16             End R
17   End S
```

Question?

Which one of the modules detects the error in the given Pascal piece of code, and when?

```
type a = array[1..10] of integer;
var i : integer; b : a;
i : = 11;
b[i] = 25;
```

- ○ Lexical Analysis in Compile time
- ○ Syntax Analysis in Compile time
- ○ Semantic Analysis in Compile time
- ○ Generated Code in Runtime

Question?

What is the state of symbol table and scope stack at the time of compiling lines 7 and 13?

```
1   Program S()
2           Var a[1..5], c, real
3           Procedure R(m: integer)
4                   Var b[1..5] integer
5                   Procedure E()
6                           Var I, c[1..3] integer
7                           c(3) := a (2) + b (1)
8                   End E
9                   Function Q(n: integer): integer
10                          Var a integer
11                          Procedure P()
12                                  Var b real
13                                  b := a + c
14                          End P
15                  End Q
16          End R
17  End S
```