

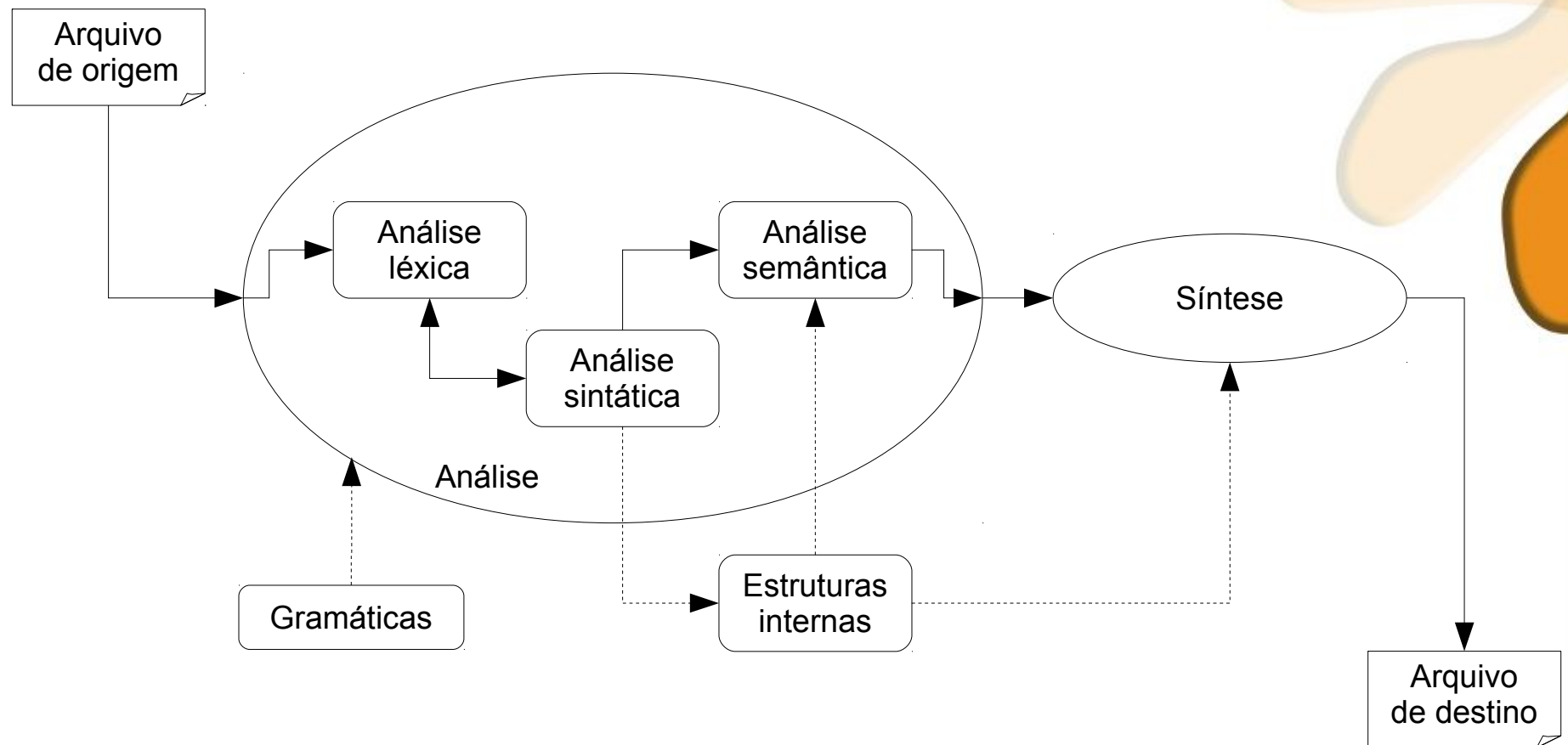
Compiladores

Análise Semântica

Cristiano Lehrer, M.Sc.



Atividades do Compilador



Introdução (1/2)

- Embora a análise sintática consiga verificar se uma expressão obedece às regras de formação de uma dada gramática:
 - Seria muito difícil expressar por meio de gramáticas livres de contexto algumas regras usuais em linguagens de programação:
 - Todas as variáveis devem ser declaradas.
 - Atribuição entre variáveis de tipos compatíveis.
 - Operação entre variáveis de tipos compatíveis.
 - Utilização de variáveis inicializadas.
 - Situações nas quais o contexto em que ocorre a expressão ou o tipo da variável deve ser verificado.

Introdução (2/2)

- O objetivo da análise semântica é trabalhar nesse nível de inter-relacionamento entre partes distintas do programa.
- Essa fase utiliza informações geradas na análise sintática, como a **árvore sintática**, e outras estruturas auxiliares que podem ser criadas durante a análise sintática, como a **tabela de símbolos**, para verificar a validade dessas regras não representadas pelas gramáticas livres do contexto.

Tabela de Símbolos

- A **tabela de símbolos** é uma estrutura auxiliar criada pelo compilador para apoiar as atividades da análise semântica do código.
- Um compilador usa uma tabela de símbolos para guardar informações sobre os nomes declarados em um programa.
- A tabela de símbolos é pesquisada cada vez que um nome é encontrado no programa fonte.
- Alterações são feitas na tabela de símbolos sempre que um novo nome ou nova informação sobre um nome já existente é obtida.
- A gerência da tabela de símbolos de um compilador deve ser implementada de forma a permitir inserções e consultas da forma mais eficiente possível, além de permitir o crescimento dinâmico da mesma.

Entradas na Tabela de Símbolos

- Cada entrada na tabela de símbolos é a declaração de um nome.
- O formato das entradas podem não ser uniforme porque as informações armazenadas para cada nome podem variar de acordo com o tipo/uso do nome:
 - Embora seja mais fácil manipular entradas uniformes.
- Cada entrada na tabela de símbolos pode ser implementada como um registro contendo campos que a qualificam:
 - Campos como nome, tipo, classe, tamanho, escopo, etc.

NOME	ATRIBUTOS
Identificador 01	
Identificador 02	
Identificador 03	

Implementação da Tabela de Símbolos (1/2)

```
program main;  
    var  a,  b,  c  :  
integer;  
begin  
    b := 10;  
    c := 15;  
    a := b + c;  
end.
```

NOME	ATRIBUTOS
a	integer
b	integer
c	integer

Implementação da Tabela de Símbolos (2/2)

```
program main;  
  var a, b, c : integer;  
  procedure sub1;  
    var b, c, d : real;  
  begin  
    d := b + c;  
  end;  
begin  
  b := 10;  
  c := 15;  
  a := b + c;  
end.
```

NOME	ATRIBUTOS
a	integer
b	integer
c	integer
sub1	procedure



NOME	ATRIBUTOS
b	real
c	real
d	real

Escopo (1/5)

- O **escopo** de uma variável de programa é a faixa de instruções na qual a variável é visível:
 - Uma variável é **visível** em uma instrução se puder ser referenciada nessa instrução.
 - As **variáveis não-locais** de uma unidade ou de um bloco de programa são as visíveis dentro daquela ou deste, mas não são declaradas aí.
 - As regras de escopo de uma linguagem determinam como uma ocorrência particular de um nome está associada à variável.

Escopo (2/5)

- No **escopo estático** (*static scoping*) o escopo de uma variável pode ser determinado estaticamente, ou seja, antes da execução do programa:
 - Conceito introduzido pelo ALGOL 60 e adotado pela maioria das linguagens de programação.
 - Processo para localizar a declaração de uma variável:
 - Iniciar a pesquisa no escopo atual e ir subindo na hierarquia de escopos até que seja encontrado uma declaração com o nome desejado.
 - Variáveis podem estar escondidas de uma unidade quando existe uma outra variável “mais próxima” com o mesmo nome.

Escopo (3/5)

- Considere o seguinte procedimento em Pascal:

```
procedure big;
```

```
    var x : integer;
```

```
    procedure sub1;
```

```
    begin {sub1}
```

```
        ...x...           {x declarado em big}
```

```
    end; {sub1}
```

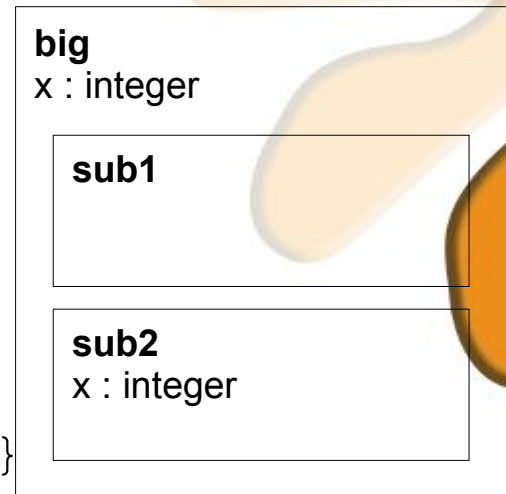
```
    procedure sub2;
```

```
        var x : integer;
```

```
    begin {sub2}
```

```
        ...x...           {x declarado em sub2}
```

```
    end; {sub2}
```



NOME	ATRIBUTOS
x	integer
sub1	procedure
sub2	procedure

NOME	ATRIBUTOS
x	integer

NOME	ATRIBUTOS
x	integer

Escopo (4/5)

- O **escopo dinâmico** baseia-se na sequência de chamada de subprogramas, não em suas relações espaciais um com o outro:
 - Escopo das variáveis em APL, SNOBOL4 e nas primeiras versões de LISP.
- Os programas em linguagens de escopo estático são de leitura mais fácil, mais confiáveis e sua execução é mais rápida do que programas equivalentes em linguagens de escopo dinâmico!

Escopo (5/5)

- Considere o seguinte procedimento em Pascal, e a seguinte chamada de procedimentos:

- $\text{big} \rightarrow \text{sub2} \rightarrow \text{sub1}$

```
procedure big;
```

```
  var x : integer;
```

```
  procedure sub1;
```

```
  begin {sub1}
```

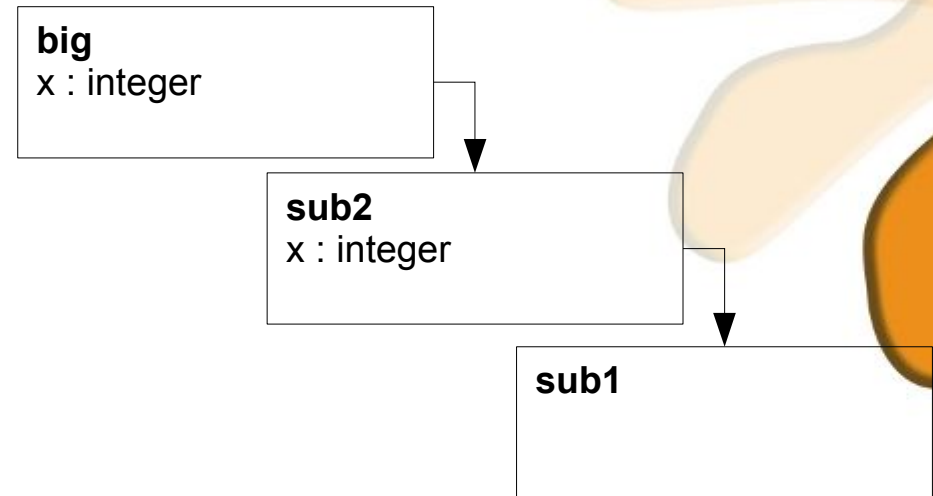
```
    ...x...
```

```
  end; {sub1}
```

```
procedure sub2;
```

```
  var x : integer;
```

```
  begin {sub2}
```



{x declarado em sub2}

NOME	ATRIBUTOS
x	integer
sub2	procedure

NOME	ATRIBUTOS
x	integer
sub1	procedure

NOME	ATRIBUTOS

Ambientes de Referenciamento (1/3)

- O **ambiente de referenciamento** de uma instrução é o conjunto de todos os nomes visíveis na instrução:
 - Numa linguagem de escopo estático, é formado pelas variáveis locais e todas as variáveis visíveis em todos os escopos superiores (mais externos).
 - Numa linguagem de escopo dinâmico, o ambiente de referência é formado pelas variáveis locais e todas as variáveis visíveis em todos os subprogramas ativos:
 - Um subprograma é **ativo** se sua execução já começou mas ainda não terminou.

Ambientes de Referenciamento (2/3)

Ambiente de Referenciamento – Escopo Estático

```
void sub1 () {  
    int a, b;  
    ... ← a e b de sub1  
}  
  
void sub2 () {  
    int b, c;  
    ... ← b e c de sub2  
    sub1 ();  
}  
  
void main () {  
    int c, d;  
    ... ← c e d de main  
    sub2 ();  
}
```

Ambientes de Referenciamento (3/3)

Ambiente de Referenciamento – Escopo Dinâmico

```
void sub1 () {  
    int a, b;  
    ... ←  
}
```

a e b de sub1, c de sub2, d de main
(c de main e b de sub2 estão ocultos)

```
void sub2 () {  
    int b, c;  
    ... ←  
    sub1 ();  
}
```

b e c de sub2, d de main
(c de main está oculto)

```
void main () {  
    int c, d;  
    ... ←  
    sub2 ();  
}
```

c e d de main