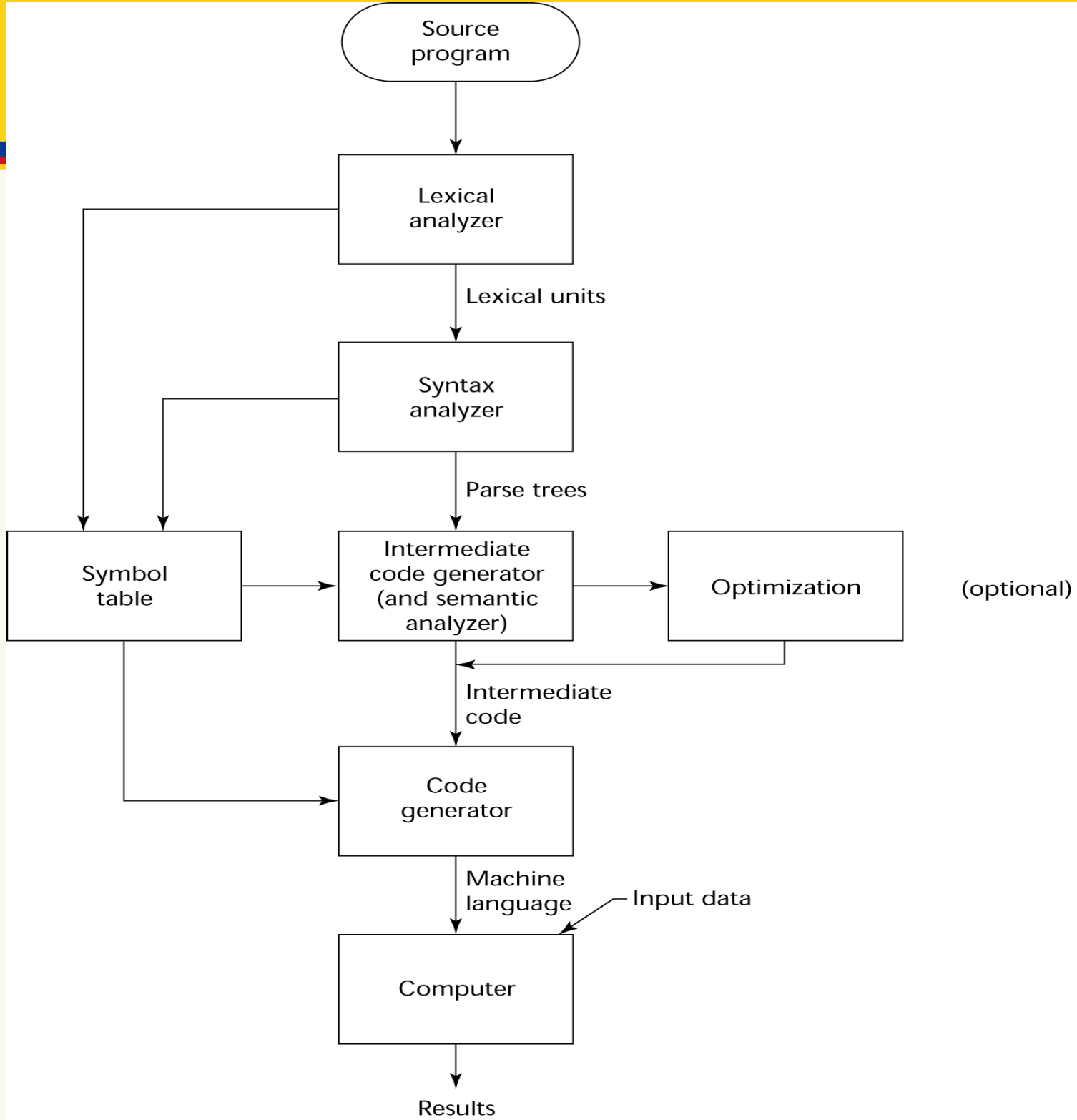
ICC043/IEC582 – Paradigmas de Linguagens de Programação

O processo de compilação: Análise léxica e sintática

Prof. Dr. Rafael Giusti
rgiusti@icomp.ufam.edu.br

Leitura recomendada

- SEBESTA. *Concepts of Programming Languages*.
  - Capítulo 4: Análise léxica e sintática


Source program
Lexical analyzer
Lexical units
Syntax analyzer
Parse trees
Symbol table
Intermediate code generator (and semantic analyzer)
Optimization
(optional)
Intermediate code
Code generator
Machine language
Input data
Computer
Results

Análise léxica

- A primeira etapa do processo é a análise léxica
- O programa é uma sequência de símbolos que é inicialmente verificado pelo analisador léxico
  - Remove/processa espaços em branco
  - Remove/processa comentários
  - Separa os elementos léxicos do programa, identificando *tokens*
  - Substitui macros e constantes nomeadas do pré-processador (opcional)

Análise léxica

- Um programa em C

```c
#define OK 0

int main(void)
{
    return OK;
}
```


```c
int main(void)
{
    return 0;
}
```

Análise léxica

- Um programa em C

```
int main(void)
{
    return 0;
}
```

Tokenização

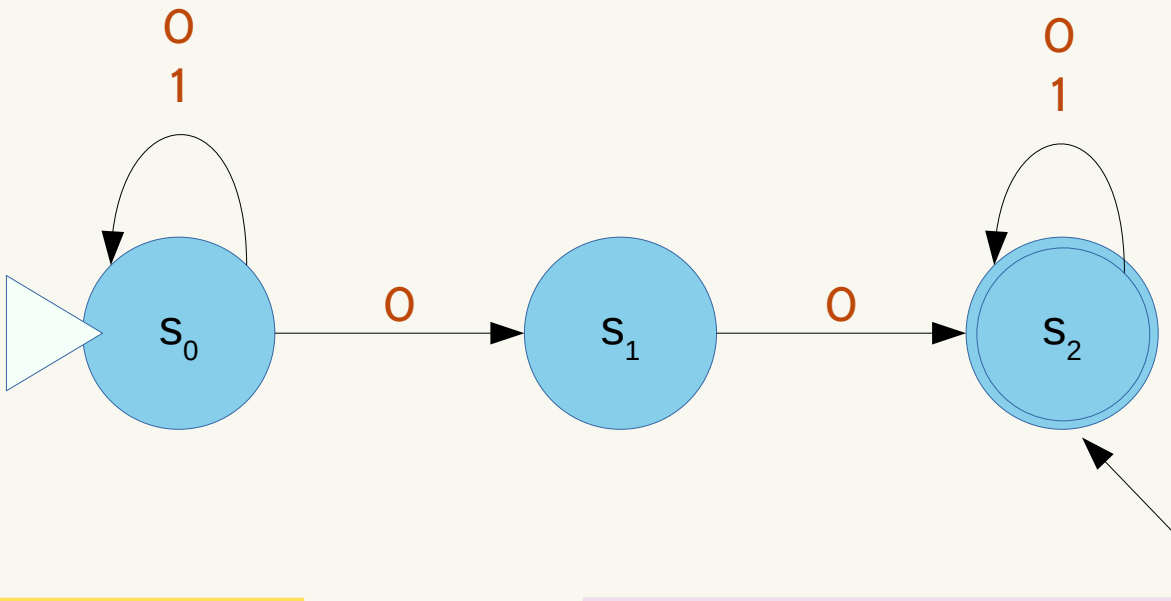
`int`	type_int
`main`	identifier
`(`	open_paren
`void`	type_void
`)`	close_paren
`{`	open_bracket
`return`	return_stmt
`0`	lit_int
`;`	semicolon
`}`	close_bracket

Análise léxica

- A análise léxica pode ser feita por um autômato finito
  - Autômatos finitos são equivalentes a gramáticas regulares e expressões regulares
  - São máquinas de estado que efetuam transições dependendo do próximo símbolo que aparece na cadeia
    - Podem ter um ou mais estados finais
    - Esses estados finais podem identificar quando um *token* é extraído

Autômato finito

- Um autômato finito para as cadeias binárias que contém pelo menos uma ocorrência de `00`



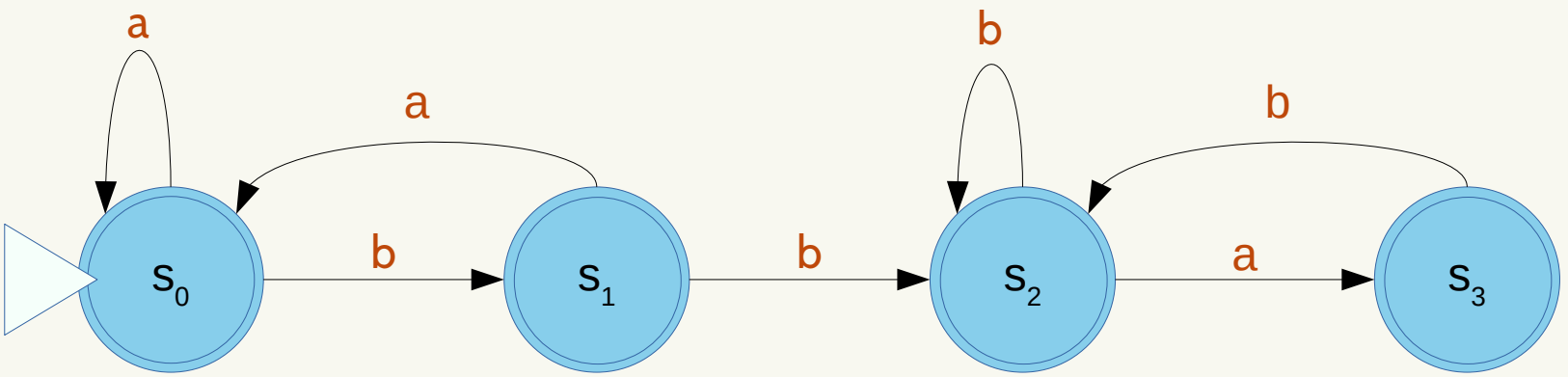
ER equivalente:

`(0+1)*00(0+1)*`

O estado final é indicado pelo círculo duplo; o autômato termina e aceita a cadeia se chegar ao estado final **e** não houver mais símbolos para serem lidos

Autômato finito

- Um autômato finito para as sequências em que "aa" não aparece depois de "bb"

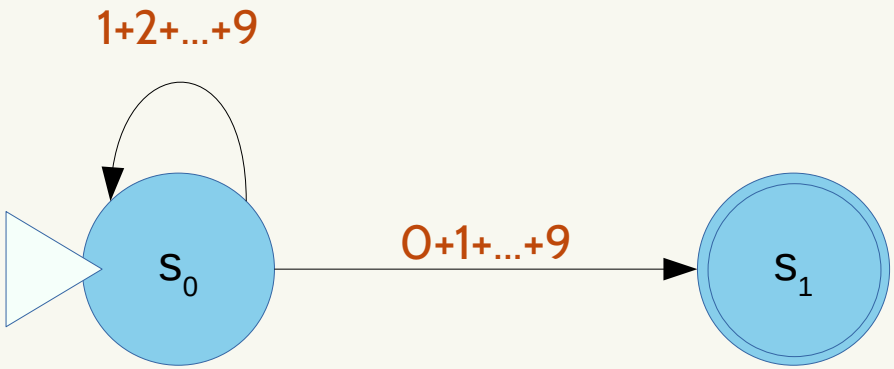


ER equivalente:

(a+ba)*(b+ba)*

Autômato finito

- Um autômato finito para literais inteiras decimais em Python 3



Autômato finito

Os estados de aceitação podem ser associados a *tokens*

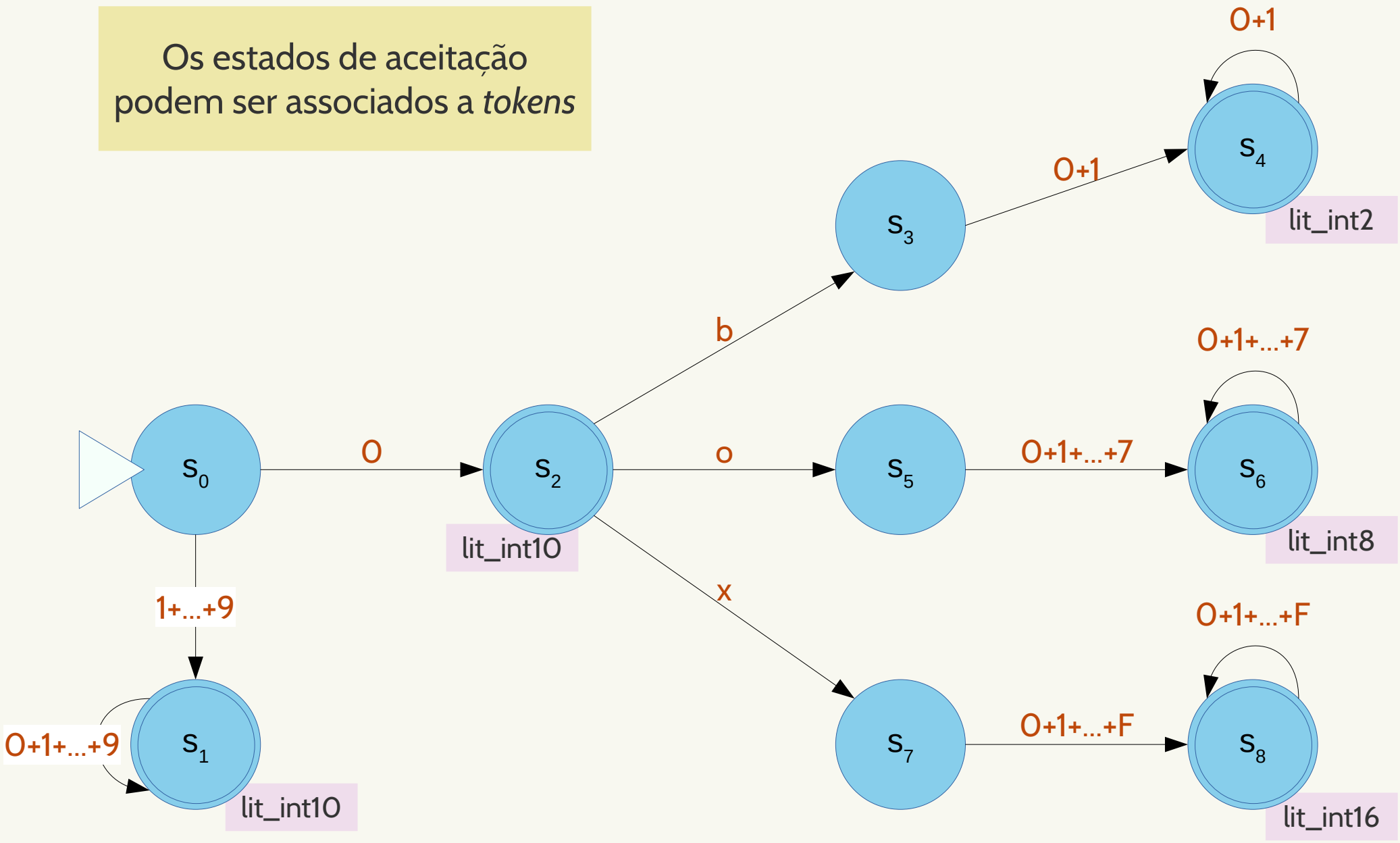


Tabela de símbolos

- Durante as etapas de análise léxica e sintática, o compilador (ou interpretador) faz uso de duas tabelas
  - A tabela de palavras reservadas é utilizada pelo analisador léxico para verificar se um lexema é um identificador ou alguma das palavras reservadas da linguagem
  - A tabela de símbolos é utilizada por ambos os analisadores para registrar nomes de variáveis, funções etc.

Tabela de símbolos

- Os atributos associados aos nomes na tabela de símbolo são chamados descritores
  - Os descritores registram escopo, tipo de dados, tipo de endereçamento etc.

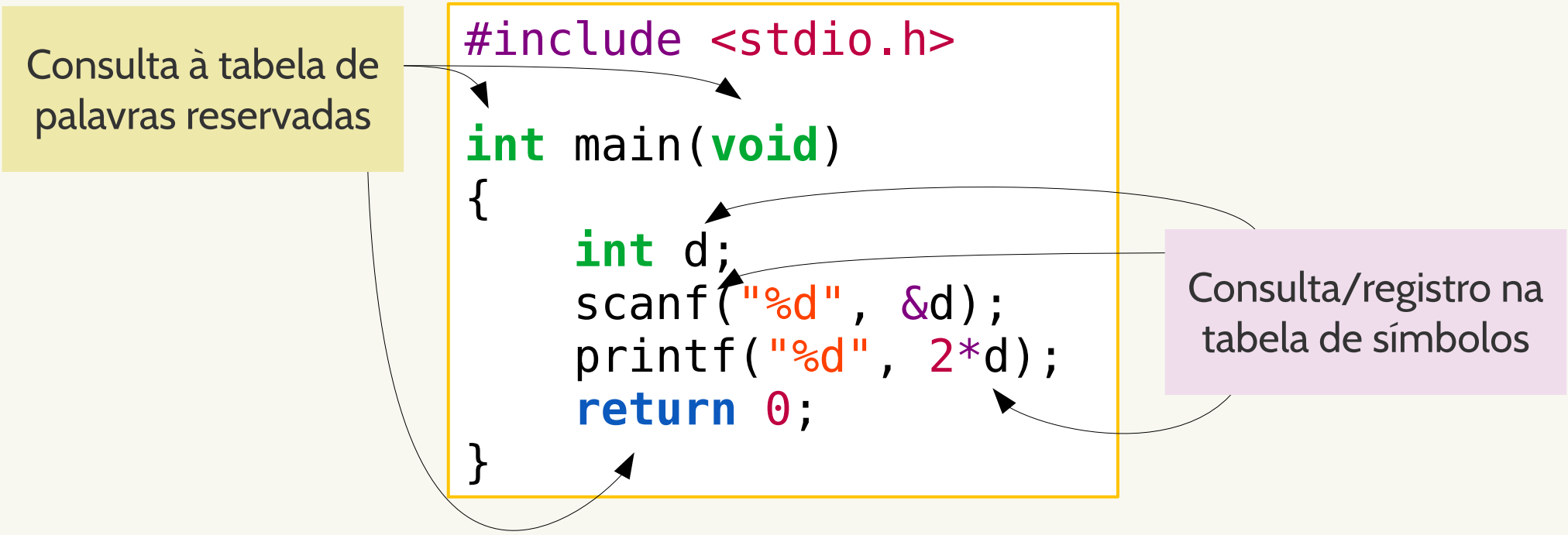


Tabela de símbolos

```
program ReadScores;
type
  TScore = range 0..100;
var
  Score: TScore;
  Students: Integer;
  Average: Float;
  i: Integer;
begin
  ReadLn(Students);
  for i := 1 to Students do
  begin
    ReadLn(Score);
    Average := Average + Score;
  end;
  Average := Average / Students;
  WriteLn(Average);
end.
```

Identificador	Tipo	Escopo
TScore	range 0..100	Arquivo
Score	range 0..100	Arquivo
Students	Integer	Arquivo
Average	Float	Arquivo
i	Integer	Arquivo
ReadLn	procedure	Global
WriteLn	procedure	Global

Tabela de símbolos hipotética para um programa em Pascal

Tabela de símbolos

- Em linguagens estáticas, a tabela de símbolos só é necessária durante a compilação
  - Após a compilação, nomes são substituídos por endereços de memória
  - A tabela de símbolos pode ser descartada
    - Os atributos são utilizados para verificar se o programa é válido e para instruir o compilador na geração do código
      - Conversões de tipos de dados
      - Passagem de argumentos etc.

Tabela de símbolos

- Em linguagens dinâmicas, variáveis e funções podem ser declaradas em tempo de execução
- A tabela de símbolos deve estar disponível também em tempo de execução
  - Em Python, por exemplo, a tabela de símbolos pode ser acessada através das funções `globals()` e `locals()`

Análise sintática

- Uma vez identificados os lexemas e os *tokens* do programa, podemos realizar a análise sintática
- Na análise sintática, a estrutura do programa é identificada e representada como uma árvore sintática
  - Esse processo é realizado por um programa conhecido como *parser*
  - Ao mesmo tempo, o *parser* pode manipular a tabela de símbolos e identificar erros de semântica estática
    - Exemplo: variável não declarada

Análise sintática

- Existem diversas técnicas para desenvolver *parsers*, mas em geral a estrutura de um *parser* segue diretamente da gramática
  - Em geral são coleções de programas recursivos (funções)
  - O *parser* normalmente faz uma única passagem pelo código-fonte, consultando o *token* seguinte para identificar quais derivações devem ser realizadas

```
<stmt> ::= <if_stmt> | <assign_stmt>
<assign_stmt> ::= <var> := <expr>
<if_stmt> ::= if <expr> then <stmt> endif
```

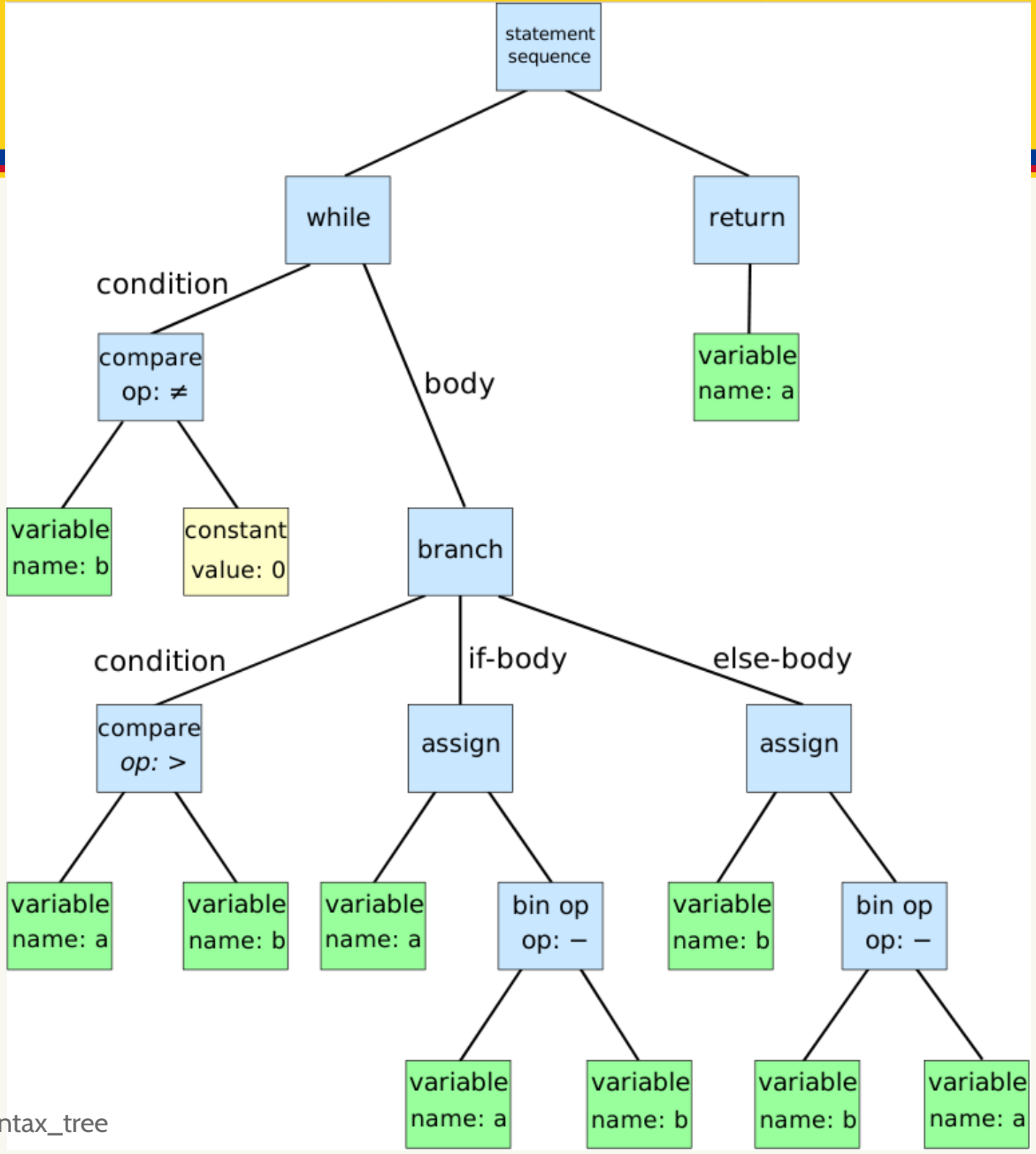
```
void stmt(Token next) {
    switch (next.token) {
        case IDENTIFIER: assign_stmt(next); break;
        case KEYWORD_IF: if_stmt(getNextToken()); break;
        default: raise_syntax_error(token);
    }
}

void assign_stmt(Token next) {
    Token left = next;
    if (!lookup(left.string)) raise_symbol_unknown(left);
    Token op = getNextToken();
    if (op.token != OPERATOR_ASSIGN) raise_syntax_error(token);
    expr();
}

void if_stmt(Token next) {
    expr();
    next = getNextToken();
    if (next.token != KEYWORD_THEN) raise_syntax_error(next);
    stmt();
    next = getNextToken();
    if (next.token != KEYWORD_ENDIF) raise_syntax_error(next);
}
```

Árvore sintática

- O objetivo do *parser* é construir a árvore sintática do programa
  - Ou uma representação simplificada da árvore, também conhecida como árvore sintática abstrata
- A árvore pode ser armazenada em memória como um grafo ou pode existir apenas implicitamente durante a execução do *parser*
- A árvore sintática pode ser utilizada para gerar o código objeto durante a compilação ou pode ser percorrida para interpretação do programa

Árvore sintática

```
while b ≠ 0
  if a > b
    a := a − b
  else
    b := b − a
return a
```



https://en.wikipedia.org/wiki/Abstract_syntax_tree