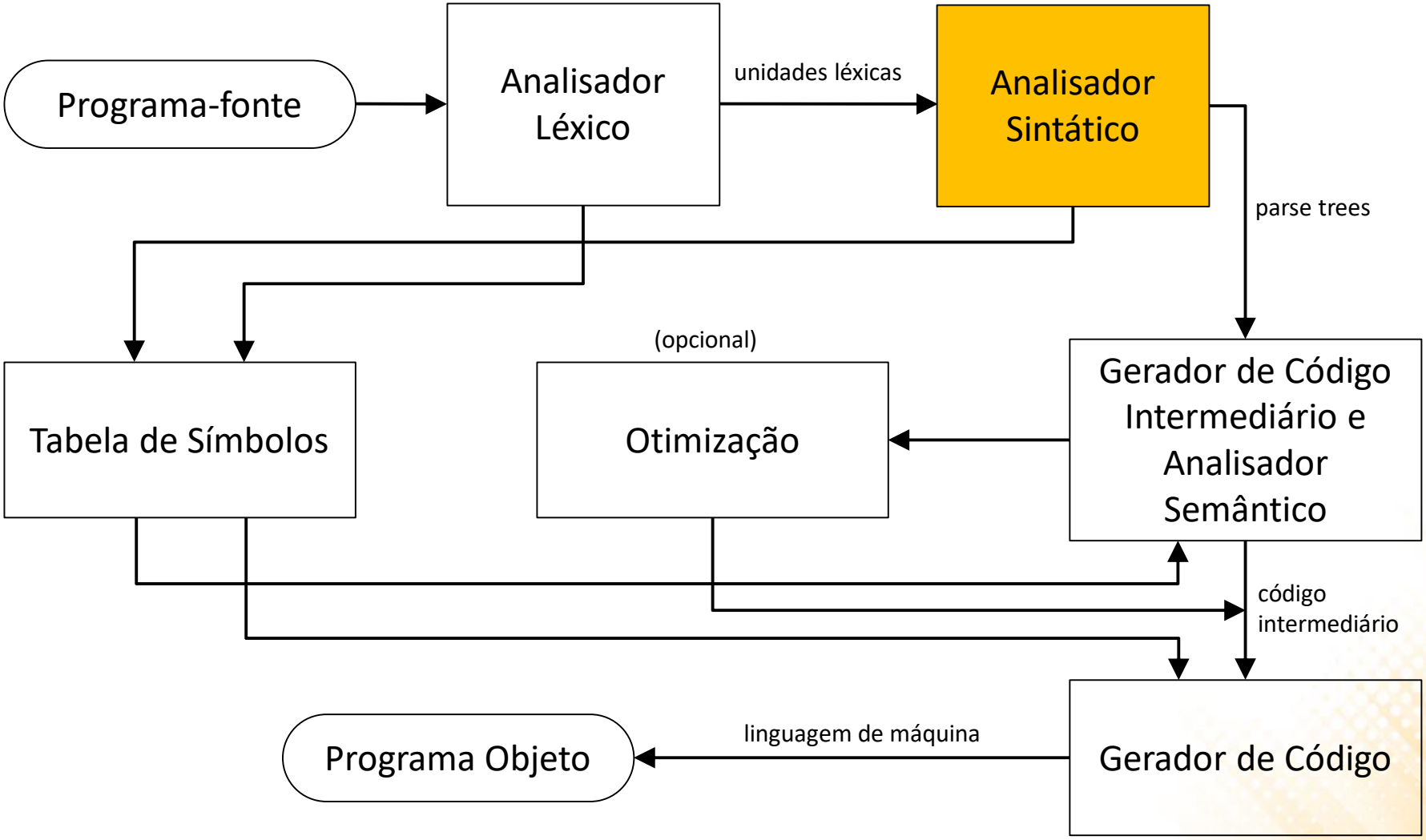
Compiladores

Aula 05 – Construção de Árvores Sintáticas (Implementação)

Edirlei Soares de Lima
<edirlei.lima@universidadeeuropeia.pt>

Processo de Compilação



Análise Sintática

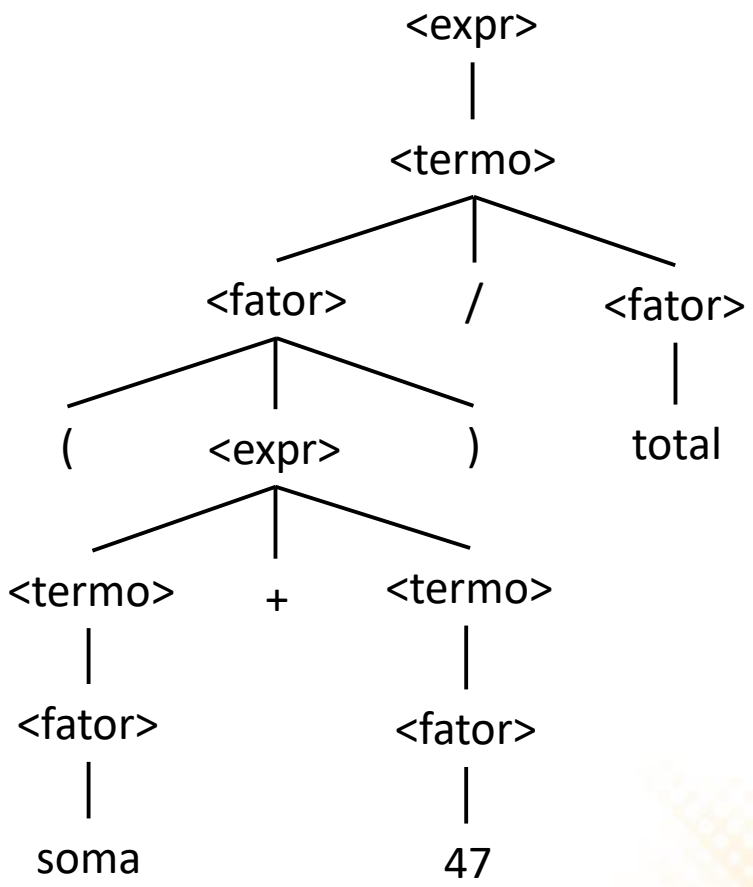
- Os analisadores sintáticos constroem árvores de análise sintática (**parse trees**) para os programas dados.
  - Em alguns casos, a parse tree é construída implicitamente (apenas o resultado de percorrer a árvore é gerado);

- Objetivos da Análise Sintática:
  - Verificar o programa de entrada para determinar se ele está sintaticamente correto;
  - Produzir árvores de análise (parse trees).

Análise Sintática

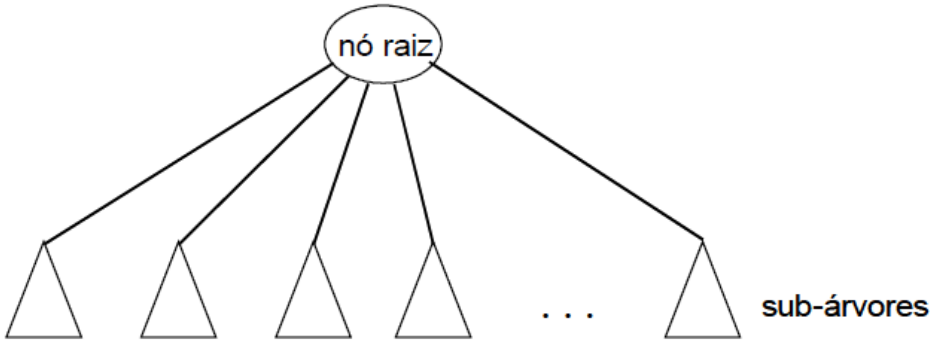
- Saída:

```
Token: 25, Lexema: (
Enter <expr>: 25
Enter <termo>: 25
Enter <fator>: 25
Token: 11, Lexema: soma
Enter <expr>: 11
Enter <termo>: 11
Enter <fator>: 11
Token: 21, Lexema: +
Exit <fator>: 11
Exit <termo>: 21
Token: 10, Lexema: 47
Enter <termo>: 10
Enter <fator>: 10
Token: 26, Lexema: )
Exit <fator>: 10
Exit <termo>: 26
Exit <expr>: 11
Token: 24, Lexema: /
Exit <fator>: 25
Token: 11, Lexema: total
Enter <fator>: 11
Token: -1, Lexema: EOF
Exit <fator>: 11
Exit <termo>: -1
Exit <expr>: 25
```



Árvores

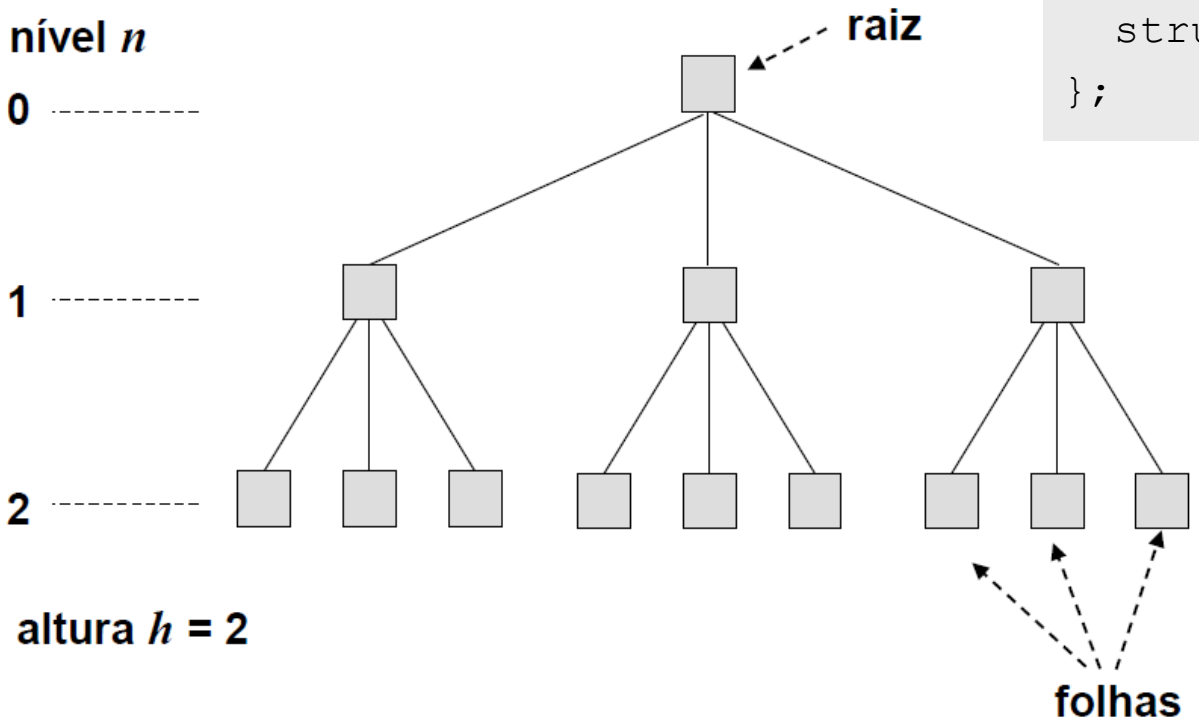
- Uma árvore é composta por um **conjunto de nós** tal que:
  - Existe um nó r, denominado **raiz**, com zero ou mais sub-árvores, cujas raízes estão ligadas a r;
  - Os nós raízes destas sub-árvores são os **filhos** de r;
  - Os **nós internos** da árvore são os nós com filhos;
  - As **folhas** ou **nós externos** da árvore são os nós sem filhos;



Árvores

- Estrutura de uma Árvore:

```
#define N 3
struct noArvore
{
  int info;
  struct noArvore* filhos[N];
};
```



Árvores – Implementação

- Representação de um nó da árvore:
  - Estrutura em C contendo:
    - A informação propriamente dita (exemplo: um caractere);
    - Um vetor de ponteiros para as sub-árvores;

```
#define MAX_CHILDREN 10

typedef struct node {
        char* info;
        int type;
        struct node* children[MAX_CHILDREN];
} Node;
```

Árvores – Implementação

- Função `CreateNode`:
  - Cria um nó raiz dadas a informação e inicializa as sub-árvores com NULL;
  - Retorna o endereço do nó raiz criado;

```
Node* CreateNode(char *value, int ntype)
{
  Node* p = (Node*)malloc(sizeof(Node));
  int i;
  p->info = value;
  p->type = ntype;
  for (i = 0; i < MAX_CHILDREN; i++)
    p->children[i] = NULL;
  return p;
}
```

Árvores – Implementação

- Função `AddChild`:
  - Adiciona/cria um nó filho ao nó recebido por parâmetro;
  - Retorna o endereço do nó criado;

```
Node* AddChild(Node* node, char *value, int ntype)
{
  int i = 0;
  while (node->children[i] != NULL)
    i++;
  if (i < MAX_CHILDREN)
    node->children[i] = CreateNode(value, ntype);
  else
    printf("Error: MAX_CHILDREN!\n");
  return node->children[i];
}
```

Árvores – Implementação

- Função `PrintTree:`
  - Imprime a árvore formatada na tela;
  - Função recursiva;

```
void PrintTree(Node *node, int tabs)
{
  int i = 0;
  int x = 0;
  for (x = 0; x < tabs; x++)
    printf("\t");
  printf("%s\n", node->info);
  while (node->children[i] != NULL)
  {
    PrintTree(node->children[i], tabs + 1);
    i++;
  }
}
```

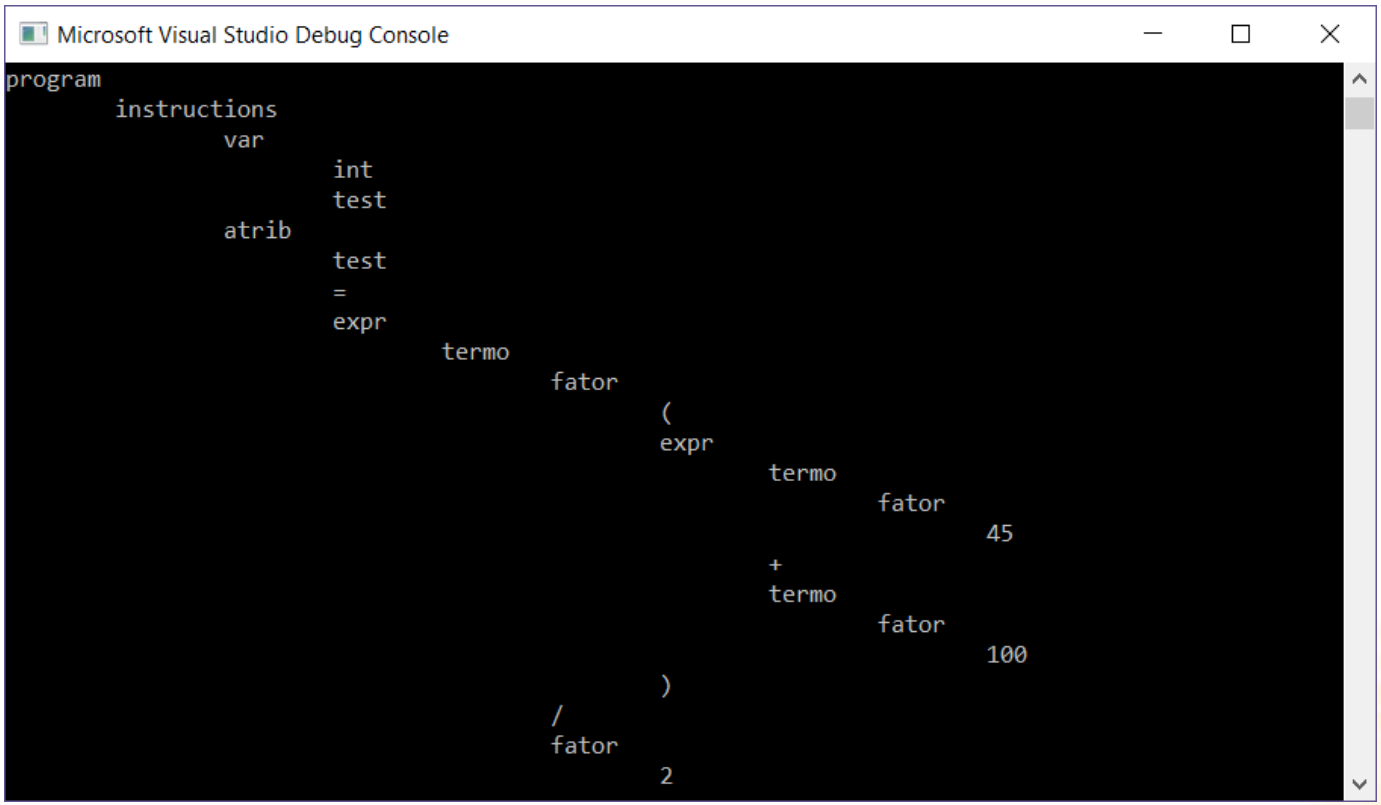
Construção da Árvore Sintática (Implementação)

```
...

int termo(FILE *code_file, int next_token, NextChar *next_char,
          Node *tree){
  printf("Enter <termo>\n");
  Node *termonode = AddChild(tree, "termo", -1);
  next_token = fator(code_file, next_token, next_char, termonode);
  while (next_token == MULT_OP || next_token == DIV_OP)
  {
    if (next_token == MULT_OP)
      AddChild(termonode, "*", next_token);
    else if (next_token == DIV_OP)
      AddChild(termonode, "/", next_token);
    next_token = lex(code_file, next_char);
    next_token = fator(code_file, next_token, next_char, termonode);
  }
  printf("Exit <termo>\n");
  return next_token;
}

...
```

Construção da Árvore Sintática (Implementação)

- Entrada:

```
int test;
test = (45 + 100)/2;
```



Exercício 01

1) Continue a implementação do analisador sintático, integrando-o com o com a gramatica desenvolvida nos exercícios anteriores. Ou seja, o analisador sintático deve ser capaz de construir árvores sintáticas para:

   a) Expressões de atribuição. Exemplo:

```
resultado = (soma + 87) / total
```

   b) Declarações de variáveis (int e float). Exemplo:

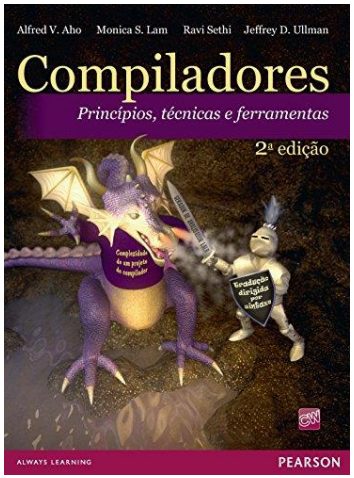
```
int a
float b
```

   c) Programas compostos por mais de uma expressão de atribuição e declarações de variáveis (separadas por ponto e vírgula). Exemplo:

```
int a;
float b;
a = 8 + 42;
b = a / 3;
```

Leitura Complementar

- Aho, A. V., Lam, M. S., Jeffrey, R. S. **Compiladores: Princípios, Técnicas e Ferramentas**. 2ª edição, Pearson, 2007. ISBN: 978-8588639249.
  - **Capítulo 3: Lexical Analysis**
  - **Capítulo 4: Syntax Analysis**



- Sebesta, R. W. **Conceitos de Linguagens de Programação**. 9ª edição Editora Bookman, 2011. ISBN: 978-8577807918.
  - **Capítulo 3: Descrevendo a Sintaxe e a Semântica**
  - **Capítulo 4: Análise Léxica e Sintática**