UNIVERSIDADE EDUARDO MONDLANE

FACULDADE DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA

COMPILADORES

Estrutura Típica de um Compilador

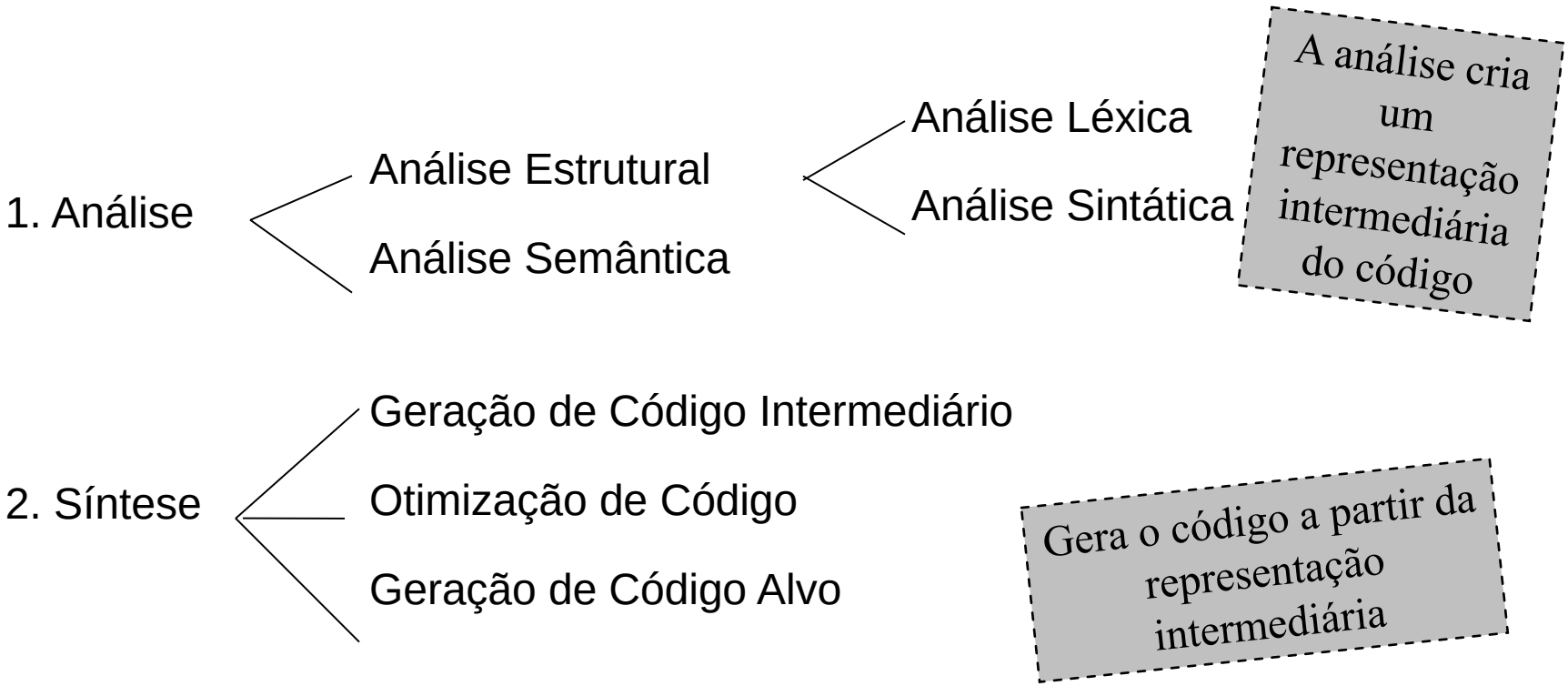
Docentes: Ruben Manhiça
Cristiliano Maculuve

Maputo, 2/28/2024

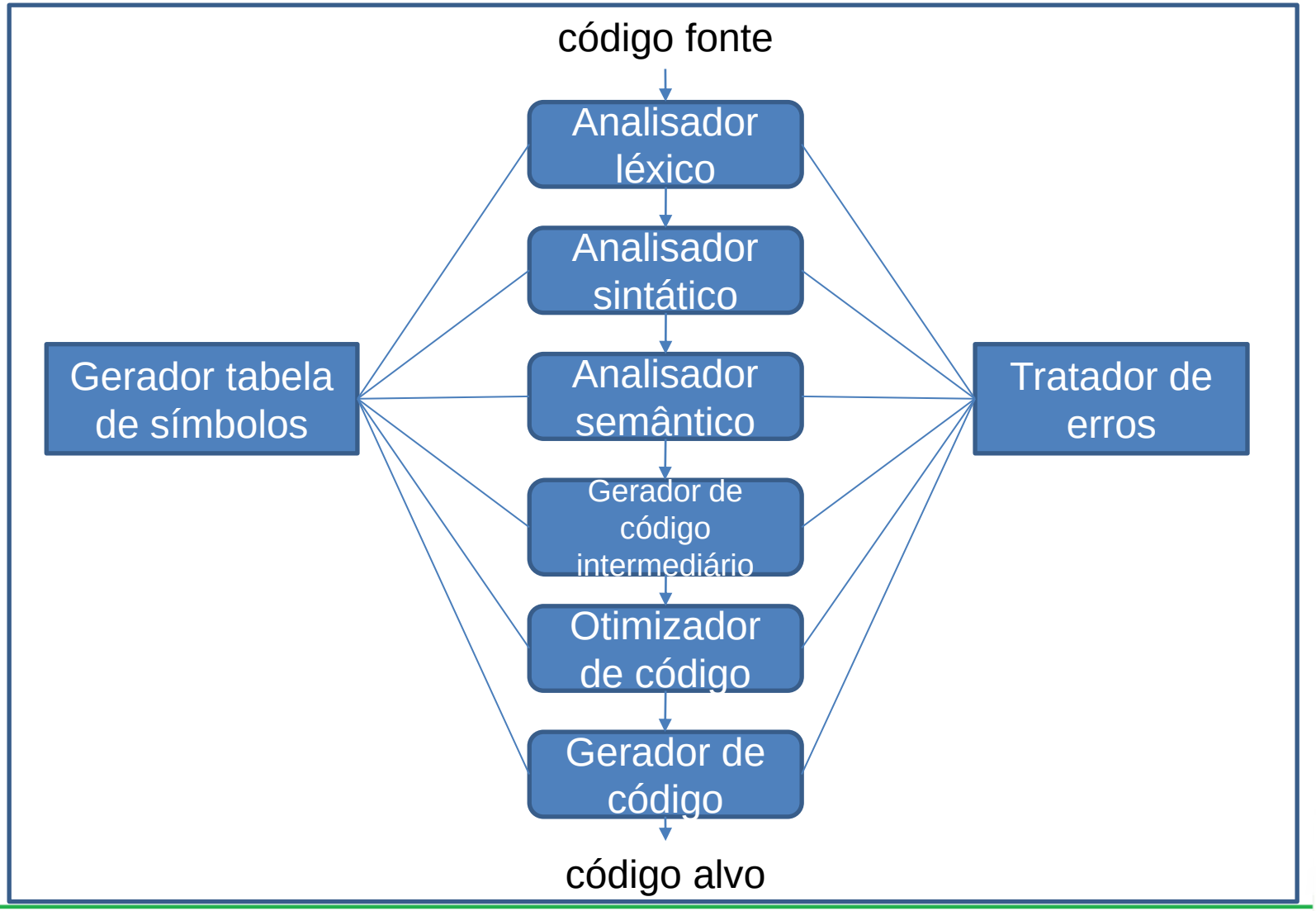
Conteúdo da Aula

1. Estrutura típica de um compilador
2. Fases do Processo de Compilação

Estrutura Geral de um Compilador
(Modelo de compilação de Análise e Síntese)



Compiladores - Fases



Análise Léxica

- Também chamada de scanner
- Agrupa caracteres em símbolos (ou tokens)
- Entrada: fluxo de caracteres
- Saída: fluxo de símbolos
- Símbolos são:
  - Palavras reservadas, identificadores de variáveis e procedimentos, operadores, pontuação,...
- Uso de expressões regulares no reconhecimento
- Lex/Flex são ferramentas que geram scanners.

Análise Léxica

Dado os caracteres da instrução

montante := saldo + taxa_de_juros * 30;

São identificados os seguintes *tokens*:

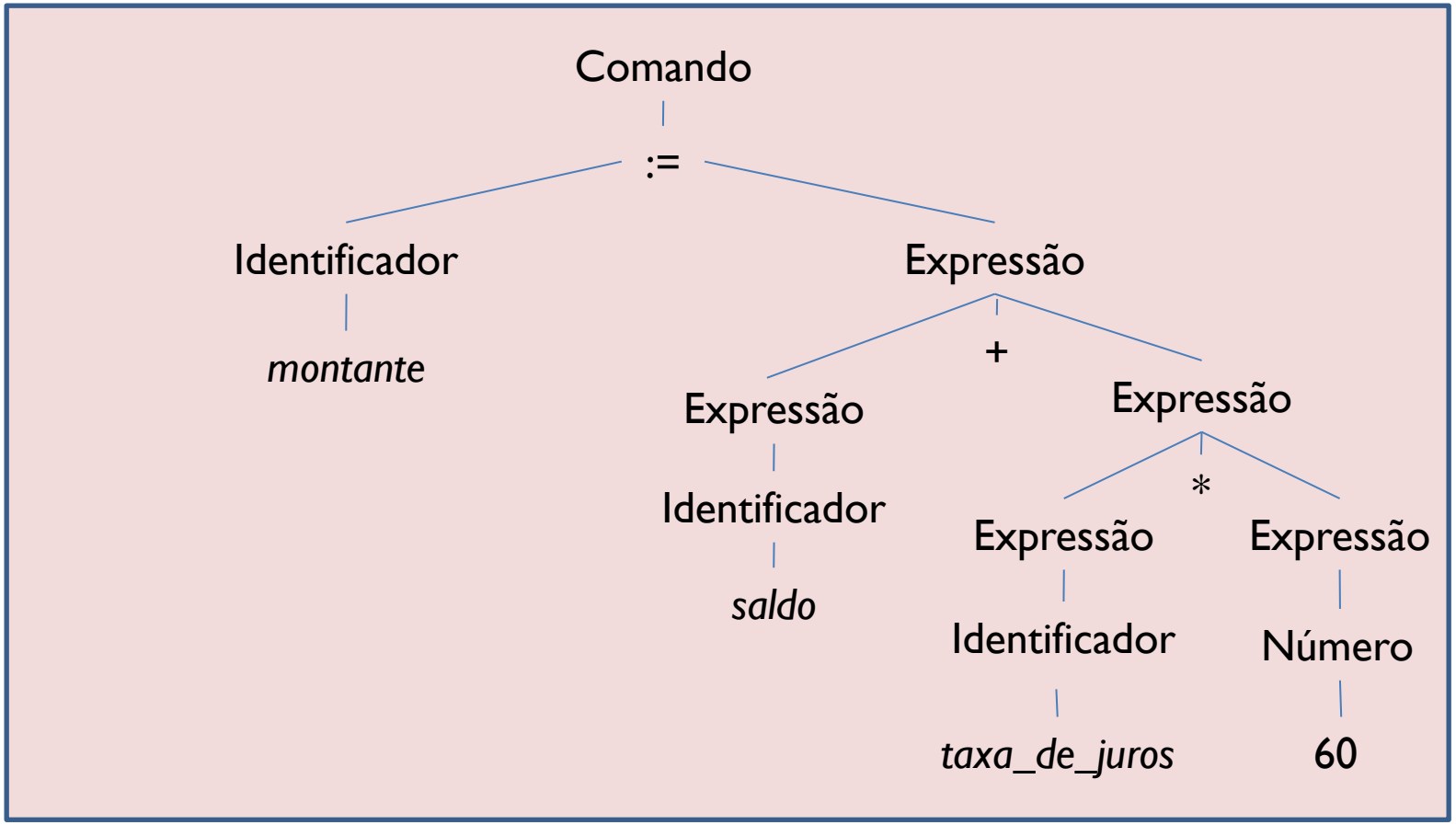
- ✓ **Identificador ->** *montante*
- ✓ **Símbolo de atribuição** -> *:=*
- ✓ **Identificador ->** *saldo*
- ✓ **Símbolo de adição ->** +
- ✓ **Identificador ->** *taxa_de_juros*
- ✓ **Símbolo de multiplicação ->** *
- ✓ **Número ->** *30*

```
java;

String a,b;

public void lancarPraga(){
      variavel = a+” , ”+b;
      system.out.println(“texto”+ifcoisa);
}

public double encontraArroba(String s){
    return s.indexOf(“@”);
}
```

Análise Sintática

- Também chamada de *parser*
- Agrupa símbolos em unidades sintáticas
- Ex.: os 3 símbolos **A+B** podem ser agrupados em uma estrutura chamada de *expressão.*
- Expressões depois podem ser agrupados para formar comandos ou outras unidades**.**
- Saída: representação árvore de parse do programa
- **Gramática livre de contexto** é usada para definir a estrutura do programa reconhecida por um parser
- **Yacc/Bison** são ferramentas para gerar parsers

Análise Sintática



Árvore gerada para: **montante := saldo + taxa_de_juros * 60**

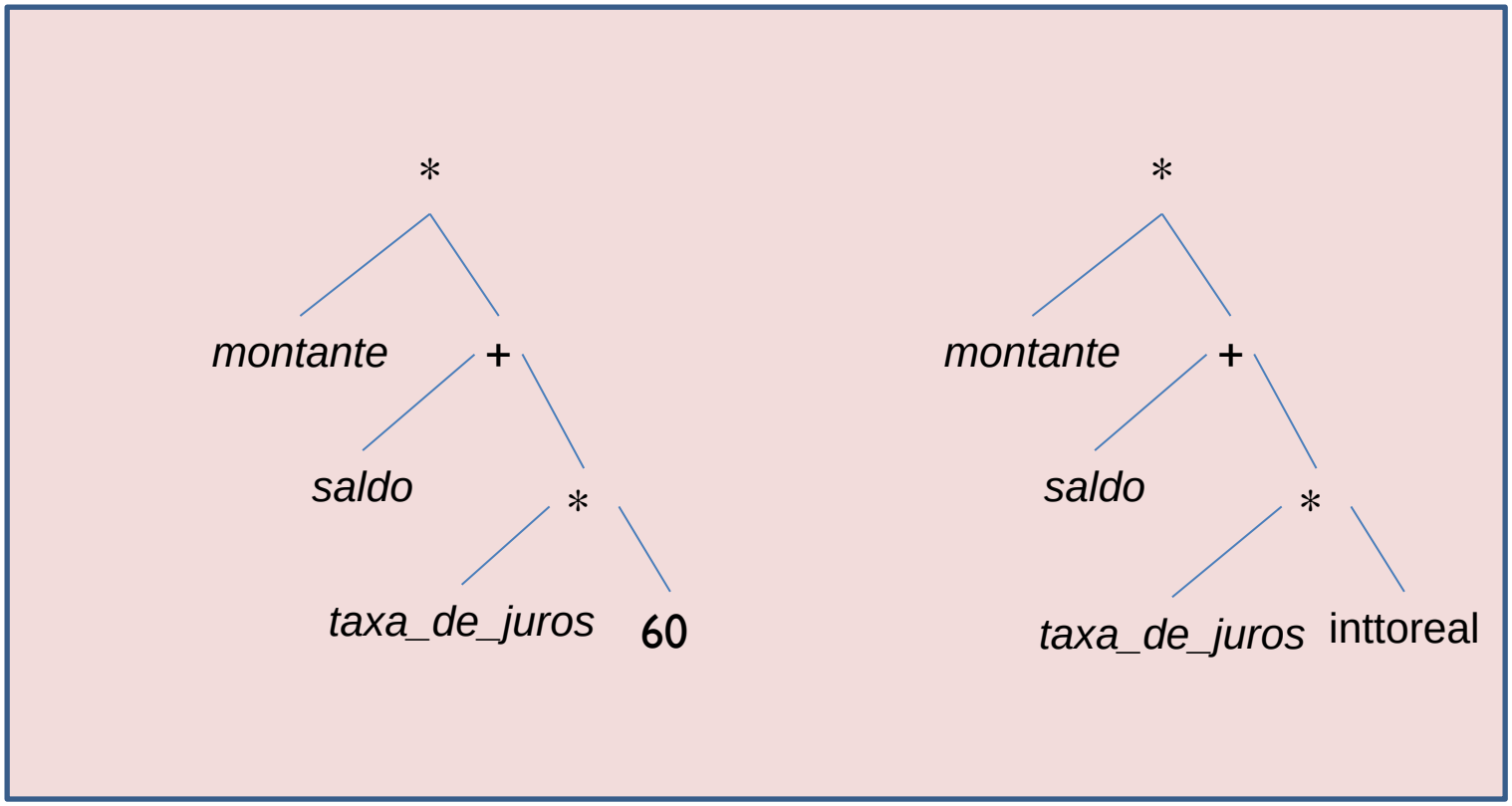
Java

```
public class run {
        private int a,c
        private string b;

            public static void main (String[]args) {
            if(a>b)  return a+b;
                else if(a<c)    system.out.print(“something);
                 system.out.print(anything);
            }

         public void doSomething(int a) {
                          while(a==true){
                        a++
                    }
                }
             }
```

Análise Semântica

- Verifica se estruturas sintáticas, embora corretas sintaticamente, têm **significado admissível na** linguagem.
- Por exemplo, não é possível representar em uma gramática livre de contexto uma regra como ***"todo identificador deve ser declarado antes de ser usado"***
- Um importante componente é **verificação de tipos.**
- Utiliza informações recolhidas anteriormente e armazenadas na tabela de símbolos
- Considerando "**A + B", quais os possíveis** problemas semânticos?
- Saída: **árvore de parse anotada**

Análise Semântica



Conversão de inteiro para real inserida pela análise semântica

Gerador de Código Intermediário

- Usa as estruturas produzidas pelo analisador sintático e verificadas pelo analisador semântico para criar uma **sequência de instruções simples** (código intermediário)
- Está **entre** a linguagem de *alto nível e a* linguagem de *baixo nível*

Gerador de Código Intermediário

- Considere que temos um único registro **acumulador.**
- Considere o comando de atribuição

  x := a + b * c

  pode ser traduzido em:
  - **t1:=b*c**
  - **t2:=a+t1**
  - **x:=t2**
- Pode-se fazer um gerador de código relativamente simples usando regras como:

Gerador de Código Intermediário

Toda operação aritmética (binária) gera 3 instruções.
Por exemplo, para *b*c* temos:

1. Carga do primeiro operando no acumulador
   load b
2. Executa a operação correspondente com o segundo operando, deixando o resultado no acumulador
   mult c
3. Armazena o resultado em uma nova variável temporária
   store t1

- Um comando de atribuição gera sempre duas instruções. Para *x:= t2*

1. Carrega o valor da expressão no acumulador
   load t2
2. Armazena o resultado na variável
   store x

Gerador de Código Intermediário

Para o comando de atribuição

x := a + b * c;

é gerado o código intermediário:

1. Load b { t1 := b * c }
2. Mult c
3. Store t1
4. Load a { t2 := a + t1 }
5. Add t1
6. Store t2
7. Load t2
8. Store x { x := t2 }

Otimizador de Código

- **Independente** da máquina
- **Melhora o código intermediário** de modo que o programa objeto seja **menor** (*ocupe menos espaço de memória) e/ou* **mais rápido** (*tenha tempo de execução menor)*
- A saída do otimizador de código é um **novo código** intermediário

Otimizador de Código

1. Load b
2. Mult c
3. Add a
4. Store x

Otimizador de Código

Fonte	código intermediário original	código intermediário otimizado
w:=(a+b)+c;	t1:=a+b t2:=t1+c w:=t2	t1:=a+b t2:=t1+c w:=t2
x:=(a+b)*d;	t3:=a+b t4:=t3*d x:=t4	 t4:=t1*d x:=t4
y:=(a+b)+c;	t5:=a+b t6:=t5+c y:=t6	 y:=t2
z:=(a+b)*d+e;	t7:=a+b t8:=t7*d t9:=t8+e z:=t9	 t9:=t4+e z:=t9

Gerador de Código

- Produz o código objeto final
- Toma decisões com relação à:
  - Alocação de espaço para os dados do programa;
  - Seleção da forma de acessá-los;
  - Definição de quais registradores serão usados, etc.
- Projetar um gerador de código que produza programas objecto **eficientes** é uma das tarefas mais **difíceis** no projeto de um compilador

Gerador de Código

- Várias considerações têm que ser feitas:
  - Há vários tipos de instruções correspondendo a vários tipos de dados e a vários modos de endereçamento;
  - Há instruções de soma específicas, por exemplo para incrementar/decrementar de 1;
  - Algumas somas não foram especificadas explicitamente:
  - Cálculo de endereço de posições em vetores;
  - Incremento/decremento registrador de topo pilha
  - Local onde armazenar variáveis;
  - Alocação de registradores

TPC

- Ler sobre Analise Léxica (Capitulo 1 do livro)
  - AHO, Alfred V., M. SETHI, Ravi, ULLMAN, Jeffrey J., Compiladores: Princípios, Técnicas, e Ferramentas, 2ª Edição, Addison Wesley 2007.

FIM!!!

Duvidas e Questões?