Construção de Compiladores

Introdução

Professor: Luciano Ferreira Silva, Dr.

Programa

- Processadores de linguagem;
- Representações de linguagens;
- Análise léxica;
- Análise sintática;
- Análise semântica;
- Geração de código;

Bibliografia

1. RICARTE, I. E. Introdução à compilação. Rio de Janeiro: Campus/Elsevier Editora, 2008.

2. LOUDEN, K. C. Compiladores: Princípios e Práticas. 2a. ed. São Paulo: Editora Thomson. 2004.

3. Aho V.A., Sethi R., Ulman D.J., Compiladores - Princípios, Técnicas e Ferramentas, tradução de Daniel de Ariosto Pinto - Ed. Guanabara Koogan, Rio de Janeiro, 1995

4. Kowaltowisk T., Implementação de Linguagens de Programação - Ed. Guanabara Dois, 1993

5. Setzer W., Mello I. H. - A Construção de um Compilador - Ed. Campus, Rio de Janeiro, 1985

Sistema de Avaliação

- **Abordagem do Curso → Teoria + Prática**

  - ✓ **AC**: Avaliação Contínua, realizada gradualmente via resolução de listas de exercícios, implementação de trabalhos computacionais [10,0 pontos];
  - ✓ **AV1** e **AV2**: Provas escritas individuais = 10,0 pontos cada.

  +

  - ✓ **Dica**: não deixe de entregar os trabalhinhos e listas, além de lhe ajudarem com a nota, vão contribuir para você compreender melhor o conteúdo.

Sistema de Avaliação

- ✓ Trabalho Final (Individual) (**AF**): valor = 10 ptos;
- ✓ **Destaque**: perceba que o AF é composto por três elementos, dos quais você deve presar pela qualidade:
  1. **Compilador**: será avaliado em **50%** da nota
  2. **Relatório**: será avaliado em **25%** da nota
  3. **Apresentação/defesa**: será avaliado em **25%** da nota

- ✓ **Dicas**:
  1. Não deixe para última hora, o TF não é um trabalho de um (01) dia;
  2. Tenha capricho, faça o seu melhor.

- Nota Final = (AC + AV1 + AV2 + AF)/4;

Linguagens

- **Humano:**
  - ✓O uso da palavra articulada ou escrita como meio de expressão e de comunicação entre pessoas.
  - ✓A forma de expressão própria de um indivíduo, grupo, classe, etc.
  - ✓Tudo quanto serve para expressar ideias, sentimentos, modos de comportamento, etc.

Linguagens

- **Linguagem de máquina:** O código de máquina, considerado como linguagem de programação em que instruções e dados são representados como sequências de dígitos binários.

- **Linguagem de programação:** Conjunto de instruções e regras de composição e encadeamento, por meio do qual se expressam ações executáveis por um computador.

Linguagens

- **Linguagem de programação de baixo nível: Linguagem de programação em que as estruturas de controle e de dados derivam diretamente da arquitetura do processador, e cujos comandos são, portanto, mais próximos aos da linguagem de máquina e mais distantes da lógica e da linguagem humanas.**
  - ✓ Linguagem *assembly*. Linguagem de programação de baixo nível, específica para cada *hardware*, e na qual toda instrução simbólica corresponde a uma operação de máquina.

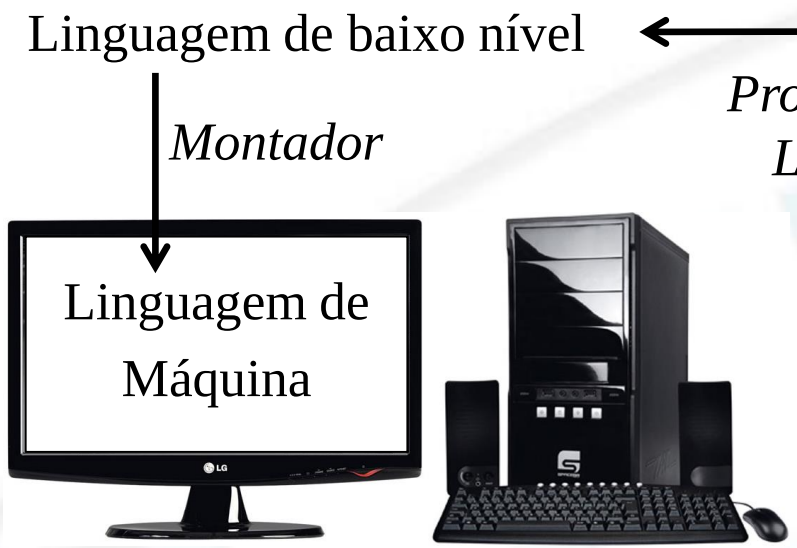
O que fazer?

Não!

Linguagens

- **Linguagem de programação de alto nível:** Linguagem de programação cuja estrutura busca aproximar-se do raciocínio humano, e que exige o uso de compilador ou de interpretador para sua execução, facilitando o processo de programação e permitindo maior complexidade desta, se comparada à linguagem de máquina ou à linguagem *assembly*.

Linguagens



- **Solução proposta:**
  - ✓ Uso de Linguagens de Alto Nível
    - usadas para produzir o programa fonte: sequência de caracteres que corresponde a uma frase, elaborada de acordo com as regras da linguagem fonte;
  - ✓ Uso de Processadores de Linguagens: Interpretador e/ou Compilador.

Interpretadores

- **Interpretador** → programa que executa diretamente as operações especificadas no programa fonte sobre as entradas fornecidas pelo usuário;

Interpretadores

- Um interpretador frequentemente oferece um melhor diagnóstico de erro do que um compilador, pois executa o programa fonte instrução por instrução.

- Os Interpretadores são em geral, menores que os Compiladores e facilitam as implementações mais completas da Linguagem de Programação.

- A principal desvantagem é que o tempo de execução de um programa interpretado é em geral, maior que o de um correspondente programa objeto compilado.

Compilador

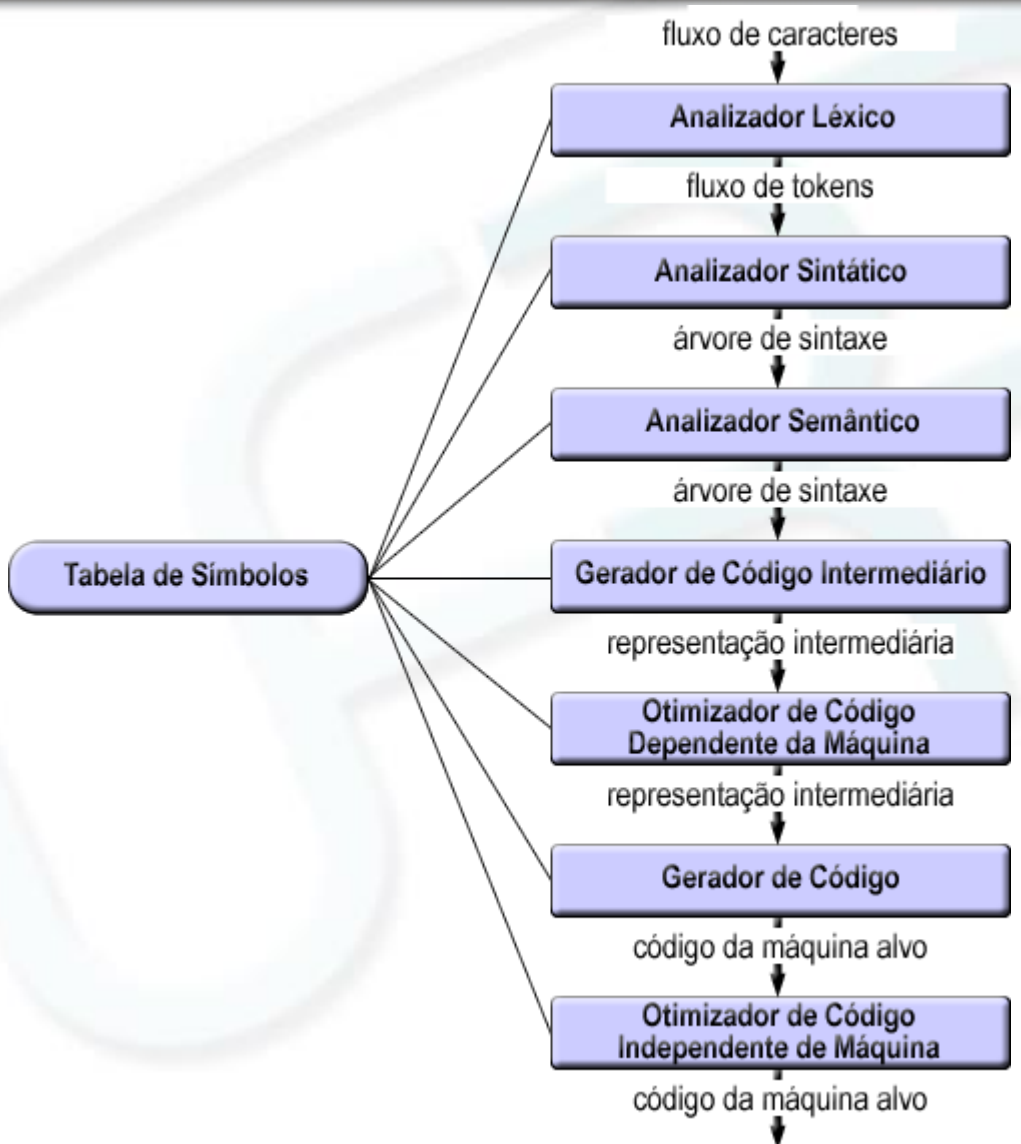
- **Compilador → é um programa que recebe como entrada um código em uma linguagem de programação – a linguagem fonte – e o traduz para um código equivalente em outra linguagem – a linguagem objeto.**

- **Programa objeto em linguagem de máquina é mais rápido no mapeamento de entradas para saídas do que um interpretador.**

Compilação + Interpretação

- **Os processadores da linguagem Java combinam compilação e interpretação.**
  - ✓ Um programa fonte em Java pode ser primeiro compilado para uma forma intermediária, chamada bytecodes (ou códigos de bytes) .
  - ✓ Os bytecodes são então interpretados por uma máquina virtual.
  - ✓ Como um benefício dessa combinação, os bytecodes compilados em uma máquina podem ser interpretados em outra máquina, talvez por meio de uma rede.

Diagrama de um Compilador



Fases de um Compilador

Análise Léxica ou scanning:

- ✓ O analisador léxico lê o fluxo de caracteres que compõem o programa fonte e os agrupa em seqüências significativas, chamadas *lexemas*.
- ✓ Para cada lexema, o analisador léxico produz como saída um *token* no formato: <*nome-token*, *valor-atributo*> que é passado para a fase subseqüente, a análise sintática.
  - *nome-token* → símbolo abstrato que é usado durante a análise sintática
  - *valor-atributo*, aponta para uma entrada na tabela de símbolos referente a esse token.
    - A informação da entrada da tabela de símbolos é necessária para a análise semântica e para a geração de código

Fase de um Compilador

- **Exemplo:**
  - ✓ suponha que um programa fonte contenha o comando de atribuição:

    *position = initial + rate * 60*

  - ✓ *position* é um lexema mapeado em um token <**id**, 1>, onde **id** é um símbolo abstrato que significa identificador e 1 aponta para a entrada da tabela de símbolos onde se encontra *position*;
  - ✓ O símbolo de atribuição = é um lexema mapeado para o token <=>;
  - ✓ *initial* é um lexema mapeado para o token <**id**, 2>;
  - ✓ + é um lexema mapeado para o token <+>;
  - ✓ *rate* é um lexema mapeado para o token <**id**, 3>;
  - ✓ * é um lexema mapeado para o token <*>;
  - ✓ 60 é um lexema normalmente mapeado para o token <**num**, 60>. A palavra **num** indica que 60 é uma constante numérica;
  - ✓ a expressão encontrada seria:

    <**id**, 1> <=> <**id**, 2> <+> <**id**, 3> <*> <**num**, 60>

Fases de um Compilador

Análise sintática:

- O analisador sintático utiliza os primeiros componentes dos *tokens* produzidos pelo analisador léxico para criar uma representação intermediária tipo árvore, que mostra a estrutura gramatical da seqüência de *tokens*.
  - Uma representação típica é uma árvore de sintaxe em que cada nó interior representa uma operação, e os filhos do nó representam os argumentos da operação

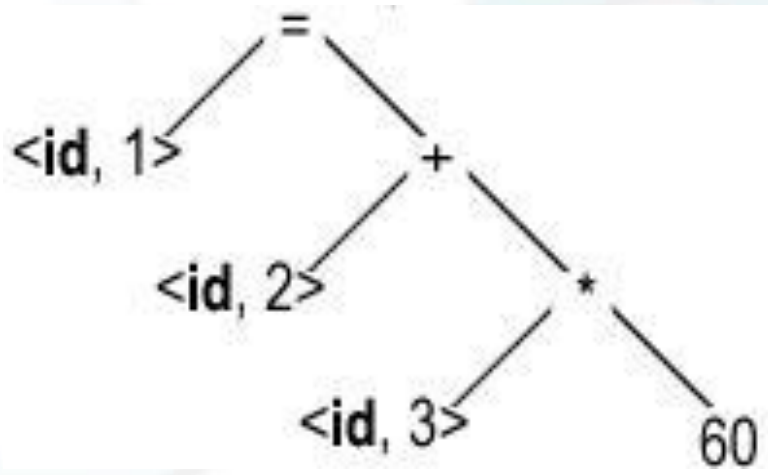
Fases de um Compilador

- **Exemplo:**

para

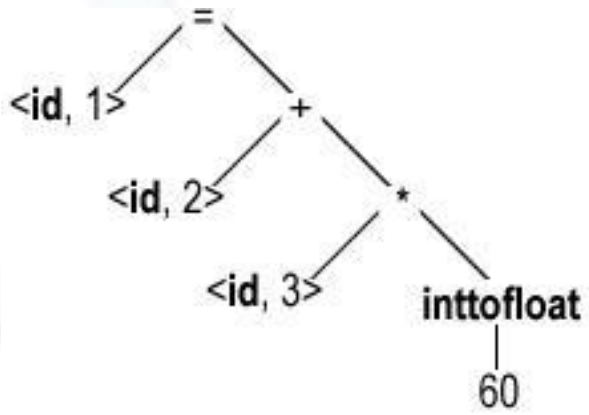
<**id**, 1> <=> <**id**, 2> <+> <**id**, 3> <*> <**num**, 60>

tem-se:



Fases de um Compilador

- **Análise Semântica:**
  - ✓ ocorre a *verificação de tipo*, em que o compilador verifica se cada operador possui operandos compatíveis;
    - Exemplo: muitas linguagens de programação exigem que um índice de arranjo seja um inteiro, portanto, o compilador precisa informar um erro de tipo se um número de ponto flutuante for usado para indexar um arranjo.
  - ✓ para o exemplo trabalhado até agora tem-se



Fases de um Compilador

Gerador de código intermediário:

- usa a estrutura produzida pelo analisador sintático para criar uma cadeia de instruções simples.
  - muitos estilos de código intermediário são possíveis. Um estilo comum usa instruções com um operador e um nº pequeno de operandos;
    - existe uma forma de representação intermediária, denominada *código de três endereços*, que consiste em uma seqüência de instruções do tipo assembler com no máximo três operandos por instrução;
    - exemplo:

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Fases de um Compilador

- **Otimização de Código (fase opcional):**
  - ✓ melhorar o código intermediário tal que o programa objeto seja mais rápido e/ou ocupe menos espaço. Sua saída é outro programa em código intermediário que faz a mesma tarefa do original;
  - ✓ exemplo:

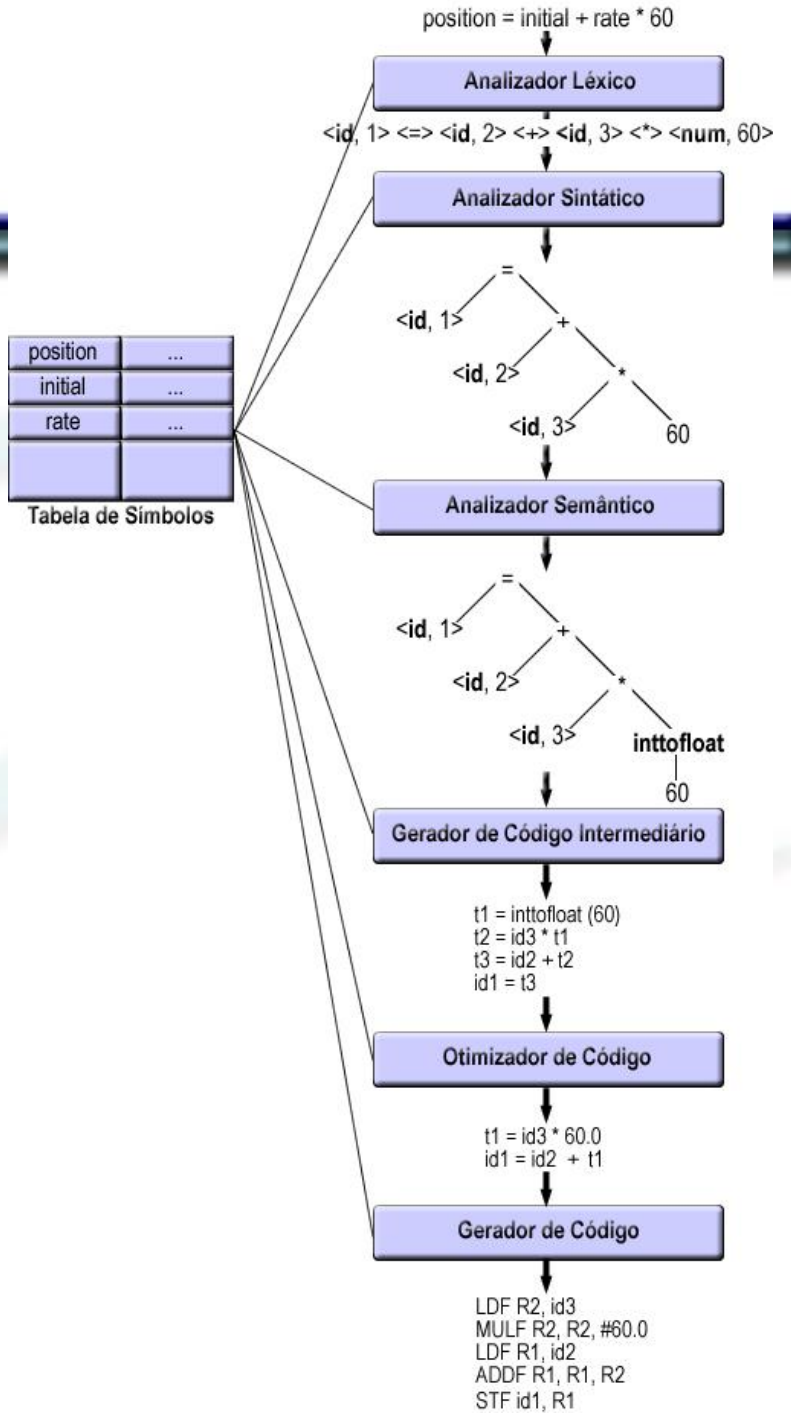
```
t1 = id3 * 60.0
id1 = id2 + t1
```

Fases de um Compilador

- **Gerador de código:**
  - ✓ gera o programa objeto. O código é gerado sempre para determinadas unidades sintáticas, sendo utilizadas informações fornecidas pelo analista de contexto.
  - ✓ exemplo:

```
LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2
ADD R1, R1, R2
STF id1, R1
```

Fases de um Compilador



1º Trabalho

- **Faça todo processo de compilação estudado na aula de hoje para as seguintes expressões matemáticas:**

✓ Atura = Cont + 3*H – 5

✓ Massa = (Altura/2) – (5*Dens + (Comp^2))

Sistema de Avaliação

- **Avaliação 01: prova individual + trabalhinhos (50% do conteúdo) →*(10,0 pontos)***

+

- **Avaliação 02: prova individual + trabalhinhos (restante do conteúdo) →*(10,0 pontos)***

+

- **Avaliação 03: Trabalhão final (todo o conteúdo) →*(10,0 pontos)***

Datas: a combinar

Dúvidas