Judson Santos Santiago

Introdução

Compiladores

Introdução

- O mundo moderno é dependente de software

Previsão do Tempo

Sistemas Embarcados

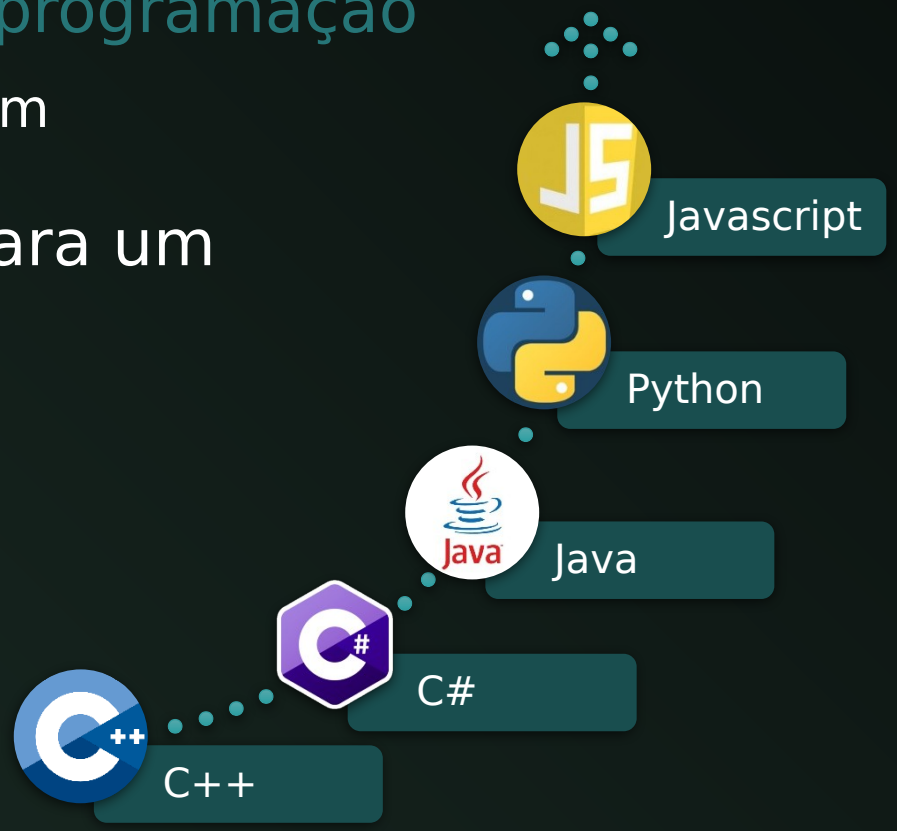
Comércio Eletrônico

Comunicação e Redes Sociais

Sincronização de Semáforos

Introdução

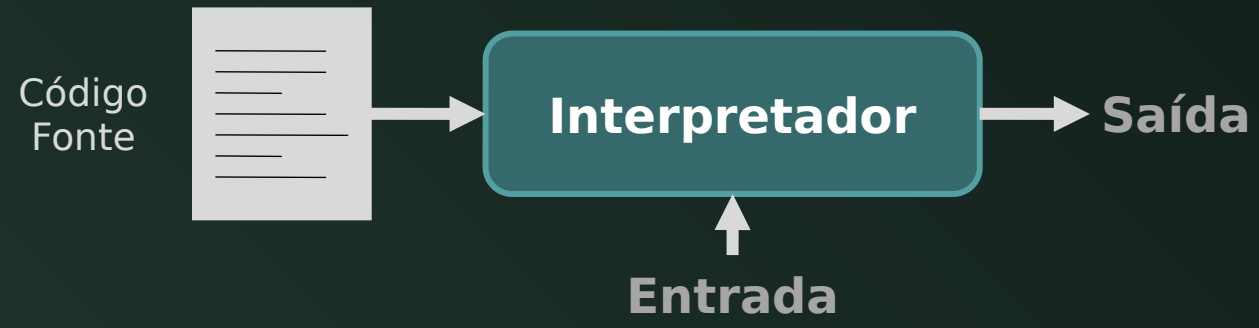
- Softwares dependem das linguagens de programação
  - Todo programa é escrito em alguma linguagem
- Os programas precisam ser traduzidos para um formato que permita a execução no computador
- Os sistemas de software que fazem essa tradução são chamados de compiladores



Processadores de Linguagem

- Existem dois tipos de processadores de linguagem

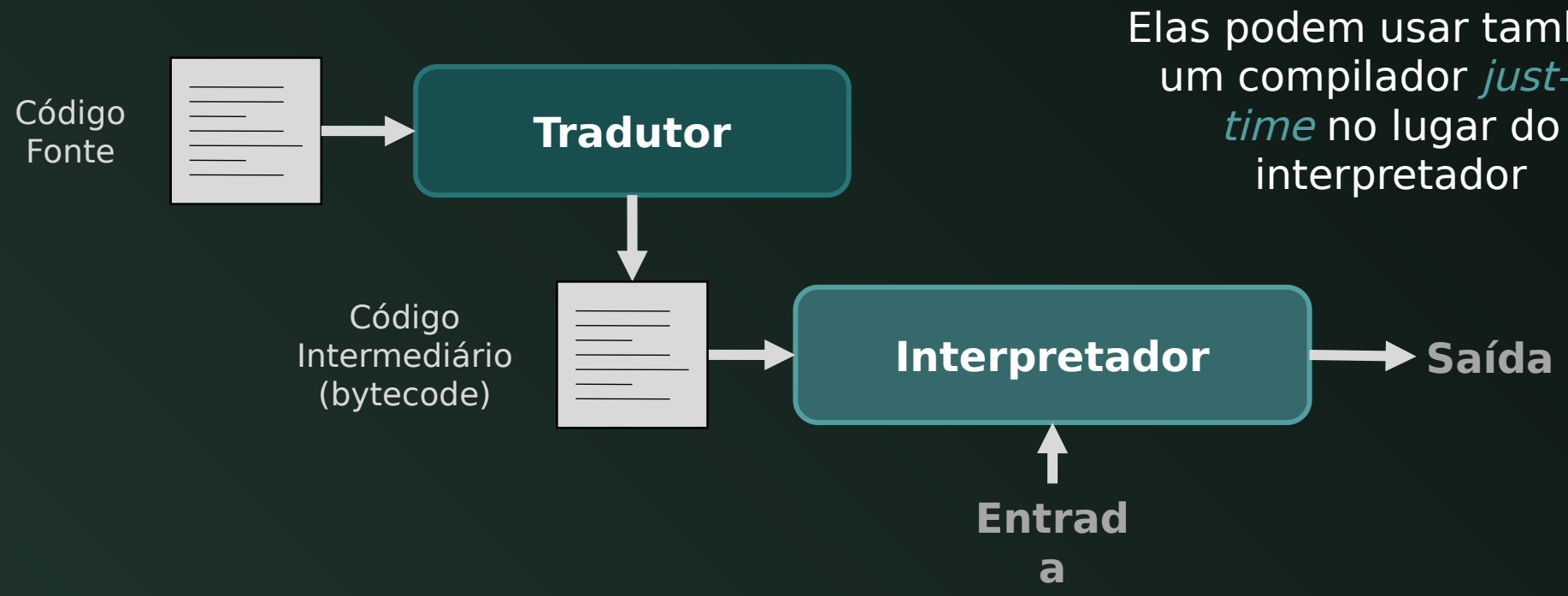
Execução mais rápida do código



Melhor diagnóstico de erros

Processadores de Linguagem

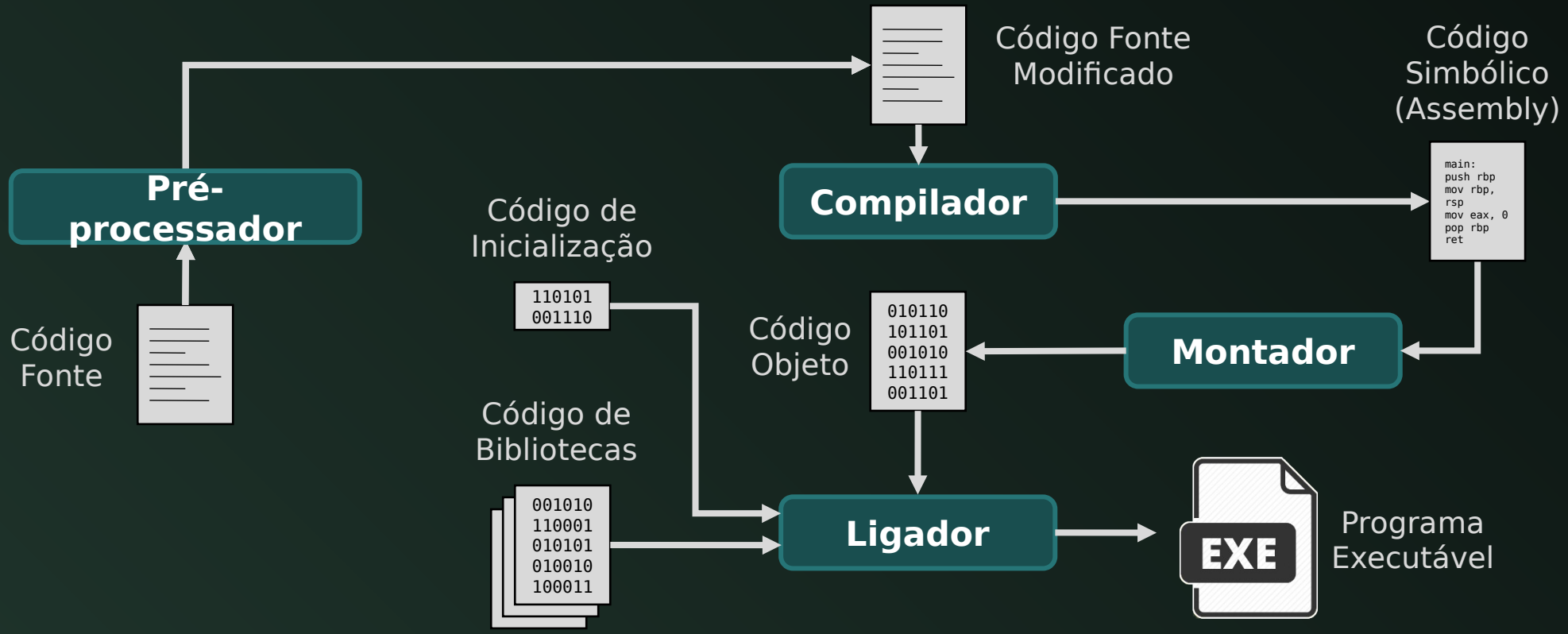
- As linguagens Java e Python utilizam processadores híbridos que combinam compilação e interpretação



Elas podem usar também um compilador *just-in-time* no lugar do interpretador

Processadores de Linguagem

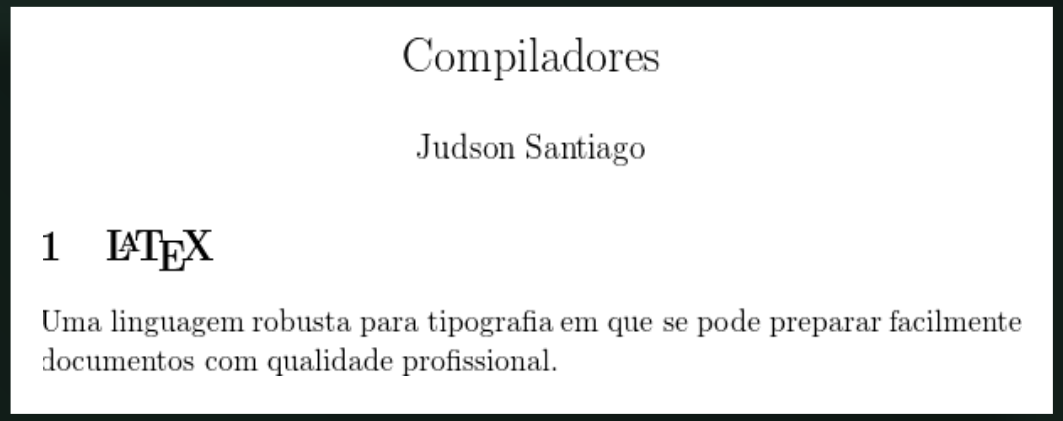
- Vários programas são usados na geração do executável



Processadores de Linguagem

- Alguns processadores de linguagem não geram executáveis
  - LaTeX é um sistema de preparação de documentos que recebe uma especificação de um documento e gera um arquivo DVI, PS ou PDF
  - Um visualizador lê uma especificação do documento escrito em uma linguagem e gera pixels na tela

```
\documentclass{article}
\title{Compiladores}
\author{Judson Santiago}
\maketitle
\begin{document}
\section{\LaTeX}
Uma linguagem robusta para
tipografia em que se pode
preparar facilmente documentos
com qualidade profissional.
\end{document}
```



Compiladores

Judson Santiago

1 LaTeX

Uma linguagem robusta para tipografia em que se pode preparar facilmente documentos com qualidade profissional.

Compiladores

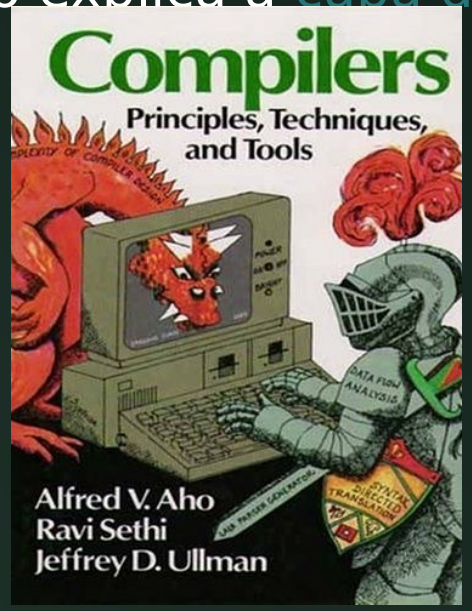
- Os compiladores utilizam muitos algoritmos e estruturas de dados
  - Buscas heurísticas gulosas
    Alocação de registradores
  - Autômatos finitos determinísticos
    Reconhecer palavras na entrada
  - Algoritmos de ponto fixo
    Análise de fluxo de dados
  - Provadores de teorema e simplificadores algébricos
    Para prever valores de expressões
  - Reconhecimento de padrão
    Mapear computações abstratas em operações em nível de máquina

Compiladores

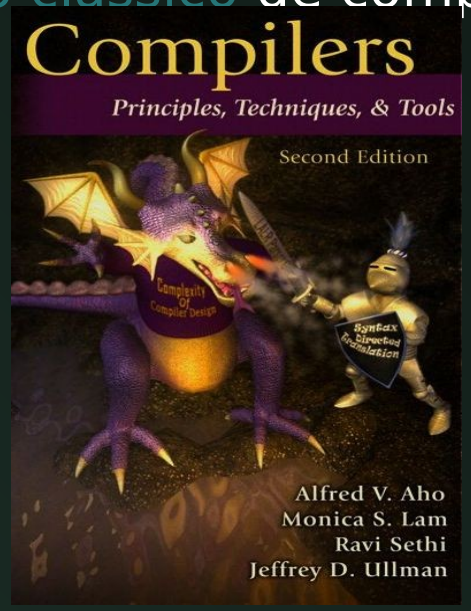
- Os compiladores utilizam muitas algoritmos e estruturas de dados
  - Equações diofantinas lineares e aritmética de Pressburger
    Analisar subscritos de vetores
  - Algoritmos em grafos
    Eliminação de código morto
  - Pilhas
    Analise sintática shift-reduce
  - Tabelas de dispersão (hash table)
    Tabela de símbolos
  - Algoritmos clássicos
    Buffers e estruturas de dados diversas

Compiladores

- Poucos sistemas reúnem tantos componentes complexos e diversos
  - Isso explica a capa do livro clássico de compiladores



1ª Edição



2ª Edição

Disciplina

Impacto dos Compiladores

- Dependem dos compiladores:
  - Avanço das linguagens de programação
    Algoritmos e representações para suportar os novos recursos
  - Avanço das máquinas
    Eles são os responsáveis pelo efetivo uso das arquiteturas de hardware
- O desempenho de um sistema computacional é totalmente dependente da tecnologia de compilação
  - Os compiladores são usados como uma ferramenta de avaliação de novas arquiteturas, antes da sua efetiva construção

Aplicações dos Compiladores

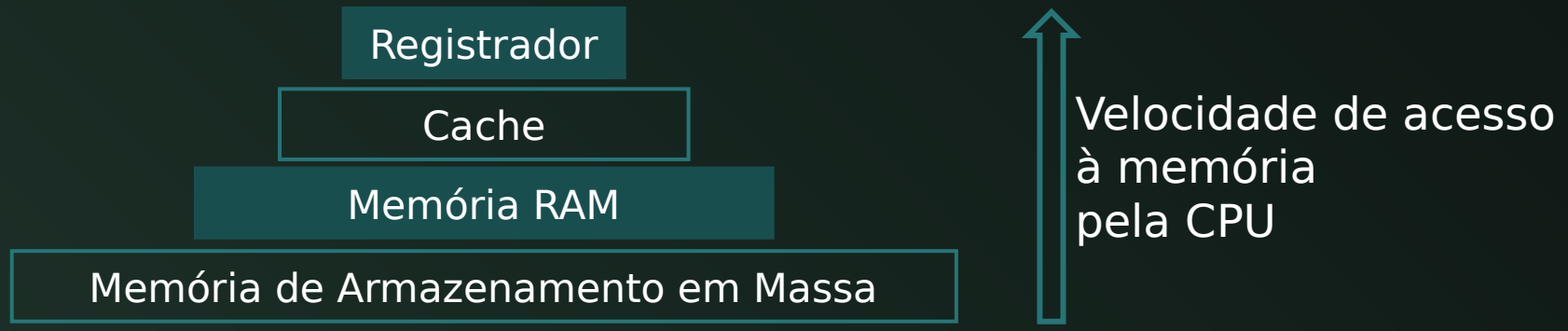
- Projeto de novas arquiteturas de computador
  - No início as máquinas eram criadas primeiro
    - O conjunto de instruções era complexo (CISC)
    - Objetivo era facilitar a programação Assembly
  - Hoje os compiladores são criados junto com as máquinas
    - O conjunto de instruções tende a ser reduzido (RISC)
      - ARM, PowerPC, SPARC, MIPS, Alpha
      - O compilador é capaz de otimizar mais e melhor
      - x86 apresenta melhor desempenho usando um subconjunto mais simples de instruções

Aplicações dos Compiladores

- Otimização para as arquiteturas de computadores atuais
  - Todos os processadores modernos exploram o paralelismo de instruções
    - Os programas são escritos como se as instruções fossem sequenciais
    - Se não há dependência entre elas, o hardware pode executá-las em paralelo
    - O compilador pode rearranjar instruções para melhorar o paralelismo
  - O paralelismo também pode aparecer no conjunto de instruções
    - Extensões SIMD – Single Instruction Multiple Data
      - Intel MMX, SSE, AVX
      - AMD 3DNow

Aplicações dos Compiladores

- Hierarquias de Memória
  - O computador possui vários níveis de armazenamento
    - Com velocidades e tamanhos diferentes
    - Quanto mais próximo do processador, mais rápido, e menor



Aplicações dos Compiladores

- Hierarquias de Memória
  - Comparação dos tempos de acesso

Atividade	Tempo de acesso	Comparação
Registrador	0.3 ns	1 s
Cache L1	0.9 ns	3 s
Cache L2	2.8 ns	9 s
Cache L3	12.9 ns	43 s
Memória Principal	120 ns	6 min
SSD	150 µs	6 dias
HDD	10 ms	12 meses

Aplicações dos Compiladores

- Hierarquias de Memória
  - O uso eficiente dos registradores é provavelmente o problema mais importante na otimização de um programa
    - Os registradores são gerenciados explicitamente pelos compiladores
  - O cache é gerenciado pelo hardware
    - Não é visível no conjunto de instruções da máquina
    - As políticas de gerenciamento de cache são genéricas
    - O compilador pode melhorar sua eficácia alterando
      - O layout dos dados
      - A ordem das instruções que acessam os dados

Aplicações dos Compiladores

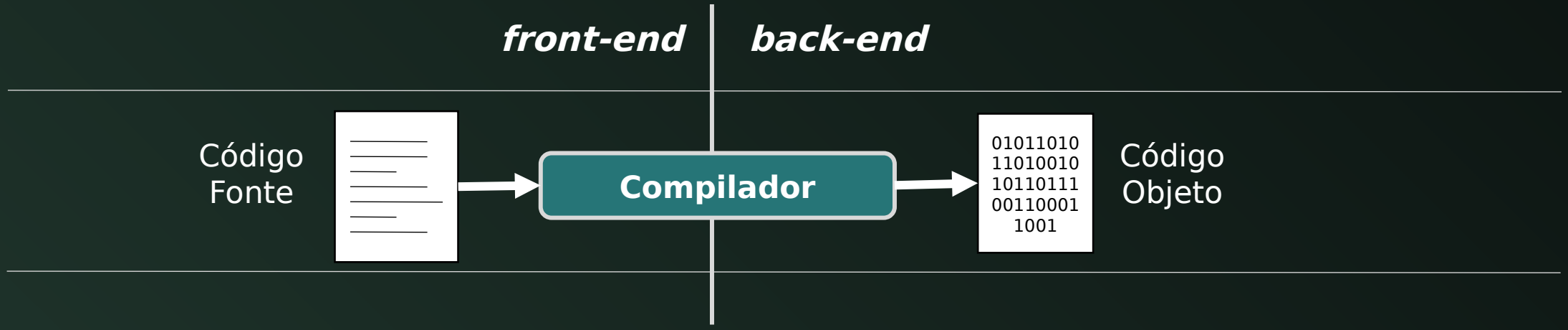
- Traduções de programas
  - A tecnologia de compiladores é usada também para fazer traduções:
    - Tradução binária
      - O código binário de uma máquina pode ser traduzido para outra
      - Permite execução de código antigo em uma arquitetura nova
        - Motorola MC 68040 para PowerPC no Apple Machintosh
        - Backward Compatibility do Xbox 360 para o Xbox One
    - Síntese de Hardware
      - Verilog e VHDL são traduzidos para RTL (Register Transfer Level)
      - RTL é traduzido para portas, transistores e um layout físico

Aplicações dos Compiladores

- Implementação das linguagens de programação de alto nível
  - É a mais importante das aplicações
    - Todo software construído passa por um processador de linguagem
    - Permite otimizações difíceis de fazer manualmente:
      - Alocação de registradores (`register`)
      - Expansão em linha (`inline`)
      - Otimizações de fluxo de dados
  - O compilador viabiliza o uso de abstrações de alto nível
    - Reduz a complexidade da programação

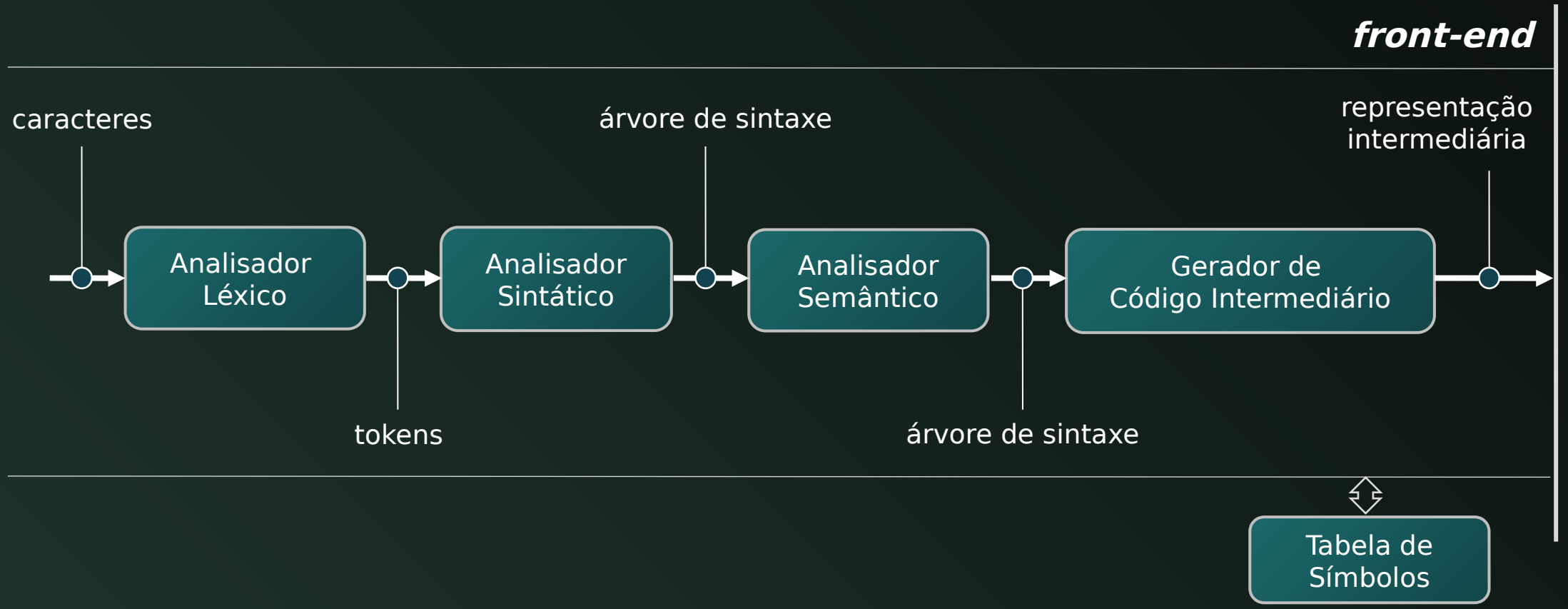
A Estrutura de um Compilador

- O processo de compilação é composto por uma sequência de fases que se dividem em duas partes:
  - Análise (*front-end*)
  - Síntese (*back-end*)



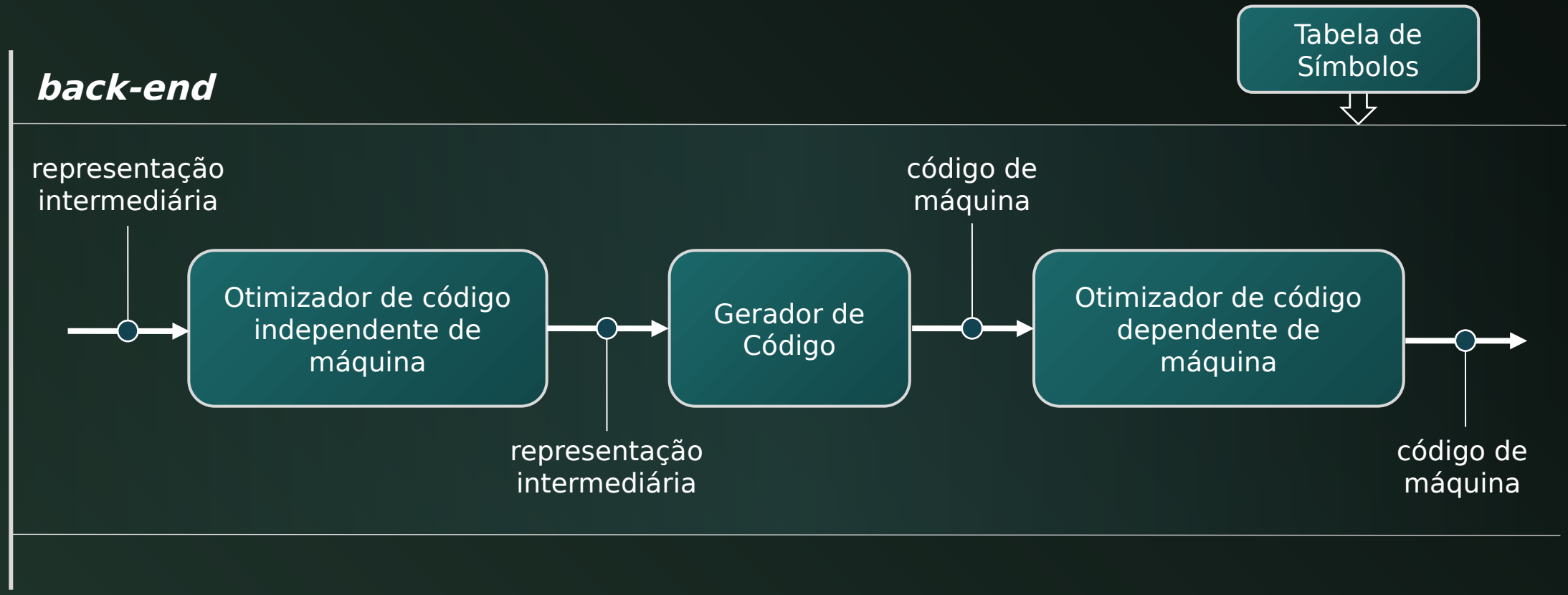
A Estrutura de um Compilador

- As fases de análise de um compilador



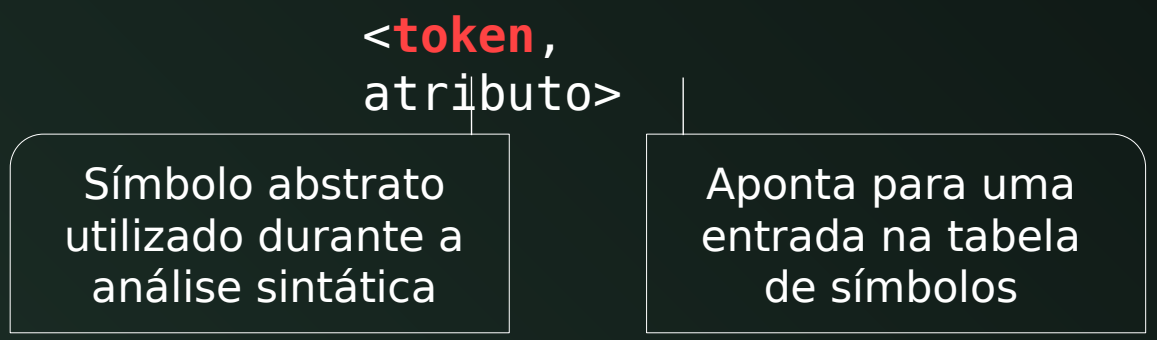
A Estrutura de um Compilador

- As fases de síntese de um compilador



Análise Léxica

- O analisador léxico lê uma sequência de caracteres e gera tokens no formato:




```
posição = inicial + taxa * 60

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

Tabela de Símbolos		
1	posição	id
2	inicial	id
3	taxa	id

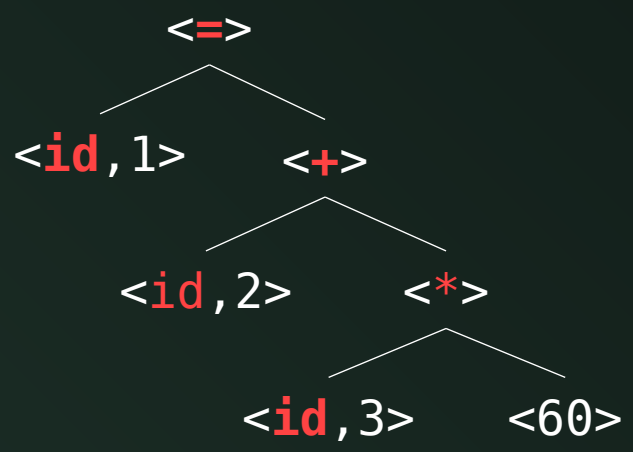
Análise Sintática

- O analisar sintático cria uma representação tipo árvore que mostra a estrutura gramatical da sequência de tokens

```
posição = inicial + taxa * 60
```

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

Tabela de Símbolos		
1	posição	id
2	inicial	id
3	taxa	id



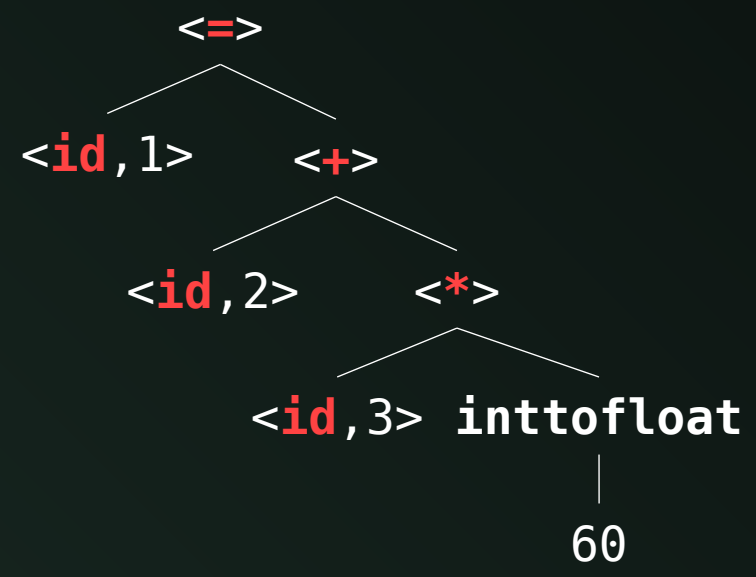
Árvore de Sintaxe gerada a partir dos tokens

Análise Semântica

- O analisador semântico utiliza a árvore de sintaxe e a tabela de símbolos para:
  - Verificar a consistência semântica do programa
  - Fazer a verificação e a conversão de tipos

Tabela de Símbolos		
1	posição	id
2	inicial	id
3	taxa	id

Se todas as variáveis forem float, é preciso converter 60 para float



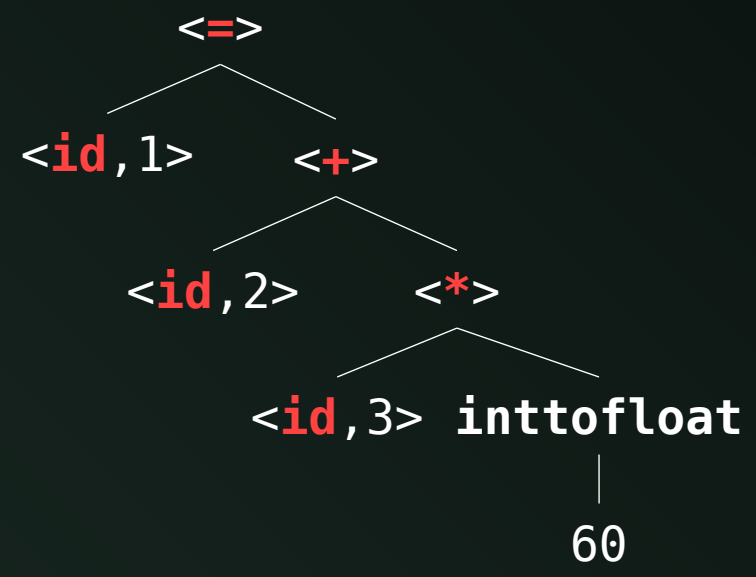
Geração de Código Intermediário

- Após as fases de análise sintática e semântica, os compiladores geram uma representação intermediária de baixo nível para uma máquina abstrata
  - Deve ser facilmente produzida
  - E facilmente traduzida para a máquina alvo

Código de 3 endereços:

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

Tabela de Símbolos		
1	posição	id
2	inicial	id
3	taxa	id



Otimização de Código

- A otimização de código faz algumas transformações com o objetivo de produzir um código objeto melhor
  - Mais rápido (e com menor consumo de energia)
  - Menor consumo de memória

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

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

A conversão do valor 60 pode ser feita durante a compilação e atribuições podem ser eliminadas

Geração de Código

- O gerador de código mapeia a representação intermediária em código de máquina de alguma arquitetura
  - Gera uma sequência de instruções de máquina
  - Um aspecto crítico é a atribuição de registradores à variáveis

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

```
LDF  R2, id3
MULF R2, 60.0
LDF  R1, id2
ADDF R1, R2
STF  id1, R1
```

Usando registradores **R1** e **R2**

O primeiro operando é o destino.

O F nas instruções indicam que elas operam sobre ponto-flutuantes

Otimização de Código

- O gerador de código pode reordenar instruções para melhorar o desempenho em certas arquiteturas de máquinas
  - O escalonamento de instruções é possível em alguns processadores que podem iniciar novas operações enquanto outras estão em andamento

Início	Fim	Instrução
1	3	LDF R2, id3
4	5	MULF R2, 60.0
6	8	LDF R1, id2
9	9	ADDF R1, R2
10	12	STF id1, R1

Ordenação original

Início	Fim	Instrução
1	3	LDF R2, id3
2	4	LDF R1, id2
4	5	MULF R2, 60.0
6	6	ADDF R1, R2
7	9	STF id1, R1

Instruções reordenadas

Supondo:

LDF - 3 ciclos
STF - 3 ciclos
MULF - 2 ciclos
ADDF - 1 ciclo

Tabela de Símbolos

- Registra os nomes de variáveis e funções e guarda informações sobre os diversos atributos de cada nome:
  - Variáveis
    - Espaço de memória alocado
    - Tipo
    - Escopo
  - Funções
    - Quantidade e tipos dos argumentos
    - Tipo de retorno
    - Passagem por valor ou referência

Tabela de Símbolos		
1	posição	id
2	inicial	id
3	taxa	id

Famílias de Compiladores

- Uma representação intermediária cuidadosamente projetada permite criar famílias de compiladores:
  - O ***front-end*** de várias linguagens podem utilizar um único ***back-end***, o que permite criar compiladores para várias linguagens
    - GNU Compiler Collection (C, C++, Objective-C, Fortran, Ada e Go)
  - O ***front-end*** de uma linguagem pode ser combinado com vários ***back-ends*** para gerar código para várias arquiteturas de máquina
    - LLVM Compiler Infrastructure (x86, PowerPC, GPU Nvidia, GPU AMD)

Ferramentas Auxiliares

- Existem várias ferramentas criadas para auxiliar o processo de criação de um compilador:
  - Geradores de analisadores léxicos
  - Geradores de analisadores sintáticos
  - Mecanismos de tradução dirigida por sintaxe
    Auxiliam na geração de uma representação intermediária
  - Geradores de geradores de código
  - Mecanismos de análise de fluxo de dados
    Essenciais para a otimização de código

A Criação do Compilador

- Ele precisa estar correto
- Ser eficiente na melhoria de muitos tipos de programas
  - Tamanho do código para aplicações embarcadas
  - Consumo de energia para dispositivos móveis
- Ter boa usabilidade
  - Depuração e geração de relatórios de erros
  - Tempo de compilação pequeno para rápido ciclo de desenvolvimento
  - Baixo custo de engenharia e manutenção

Conclusão

“Os compiladores desempenham papel fundamental na atividade central da ciência da computação: preparar problemas para serem solucionados pelo computador.”

“A maior parte do software é compilada, e a exatidão desse processo e a eficiência do código resultante têm impacto direto sobre nossa capacidade de construir sistemas de grande porte.”

Cooper & Torczon
Construindo Compiladores

Resumo

- Existem dois tipos de processadores de linguagem:
  - Compiladores: traduzem programas para linguagem de máquina
  - Interpretadores: executam programas em linguagem de alto-nível
- A compilação se divide em duas partes:
  - Análise: leitura e análise do código fonte
  - Síntese: geração e otimização do código objeto
- O desenvolvimento de um compilador é uma tarefa complexa
  - Ele deve estar correto, ser eficiente e ter uma boa usabilidade