

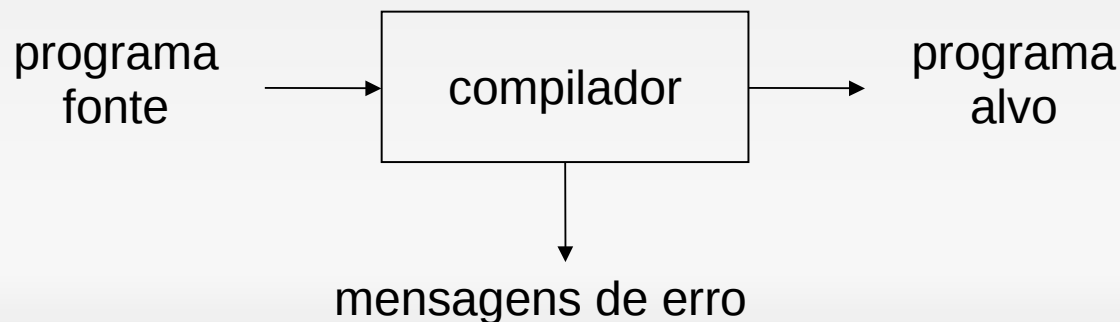
Construção de Compiladores

Introdução

Profa. Helena Caseli
helenacaseli@dc.ufscar.br

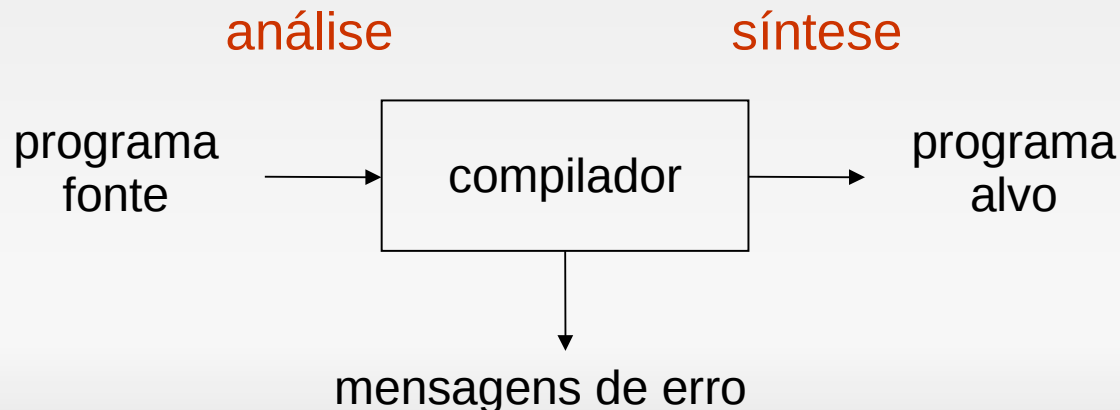
Introdução

- Compilador
 - Programa que lê um programa em uma linguagem-fonte e o traduz em um programa em uma linguagem-alvo (objeto)
 - Linguagem-fonte: Pascal, C etc.
 - Linguagem-alvo: linguagem de montagem (*assembly*), código de máquina
 - Durante o processo de tradução, relatam-se erros encontrados



Introdução

- Modelo de compilação em 2 fases
 - **Análise**
 - Divide o programa-fonte nas partes constituintes e cria uma representação intermediária do mesmo
 - **Síntese**
 - Constrói o programa-alvo desejado a partir da representação intermediária



Introdução

- Breve histórico
 - Evolução da programação
 - Linguagem de máquina C7 06 0000 0002
 - Linguagem de montagem MOV X, 2
 - Linguagem de alto nível x = 2
 - ➔ Hoje, dispomos de ambientes de desenvolvimento iterativo baseados em janelas (IDE) contendo editores, organizadores, depuradores e gerenciadores de projetos

Introdução

- Breve histórico
 - Início dos anos 50 – surgem os primeiros compiladores
 - Diversos experimentos e implementações independentes
 - Compiladores eram considerados programas muito difíceis de se construir
 - Desde então foram desenvolvidas
 - Técnicas sistemáticas para construção de compiladores
 - Reconhecimento de cadeias, gramáticas, geração de linguagem
 - Boas linguagens e ambientes de programação
 - C, C++, bibliotecas, linguagens visuais
 - Programas para produção automática de compiladores
 - Lex, Yacc, ANTLR, etc.
- ➔ Atualmente, um aluno de graduação pode construir um compilador simples em 1 semestre

Introdução

- Programas relacionados a compiladores
 - Interpretador
 - Montador
 - Organizador
 - Carregador
 - Depurador

Introdução

- Programas relacionados a compiladores
 - Interpretador
 - Traduz programas, assim como um compilador porém sem mapeá-los para linguagem de máquina
 - Interpretador X compilador
 - O interpretador executa o programa fonte imediatamente ao invés de gerar um código-objeto para ser executado após a tradução
 - Em princípio, qualquer linguagem de programação pode ser interpretada ou compilada

Compilador

- Compila uma única vez, executando quantas vezes quiser
- Tempo de execução menor
- Ex: C, Pascal

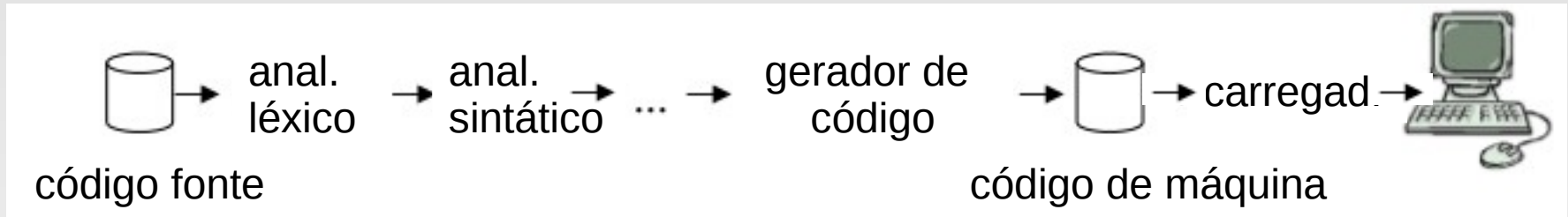
Interpretador

- Mais simples que o compilador
- Mais adaptável a ambientes computacionais diversos
- Inicia a execução mais rapidamente
- Ex: Javascript, Python, Perl

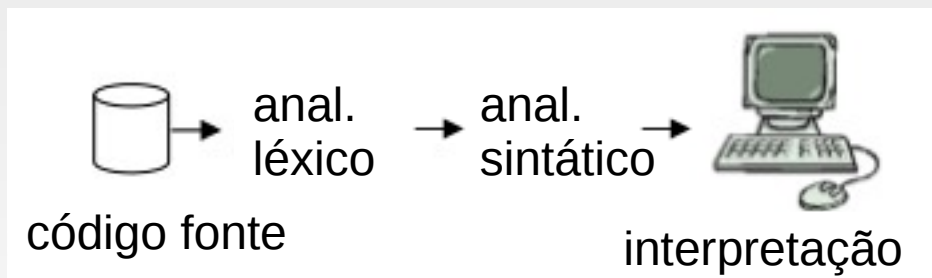
Introdução

- Programas relacionados a compiladores
 - Interpretador X Compilador (<http://ssw.jku.at/Misc/CC/>)

Compilador

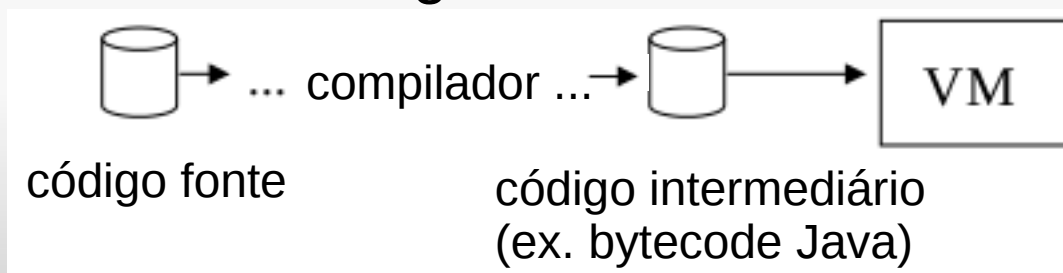


Interpretador



- comandos em um loop são analisados léxica e sintaticamente a cada iteração

Interpretador de código intermediário

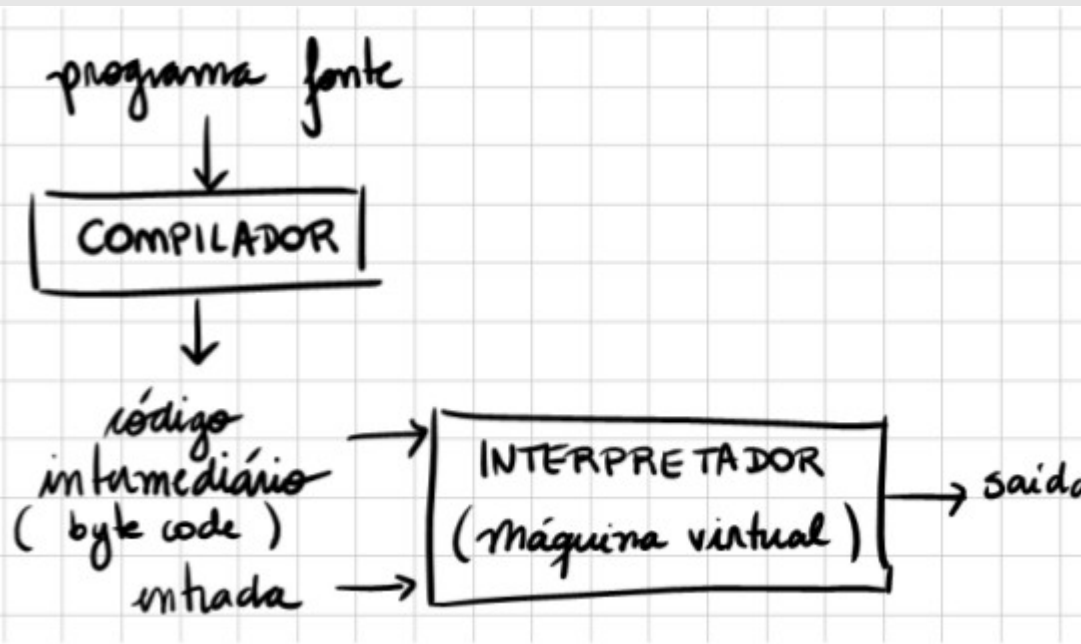


- o código fonte é traduzido em código de uma máquina virtual (VM)
- a VM interpreta o código simulando a máquina física

Introdução

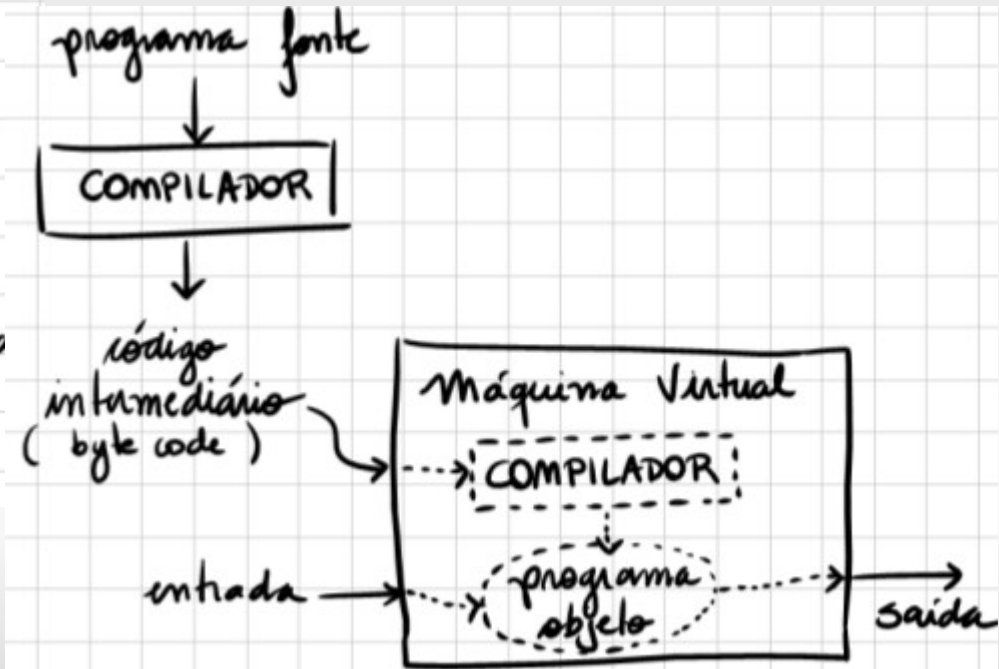
- Programas relacionados a compiladores
 - Interpretador X Compilador (notas aula Prof. Daniel Lucrédio)

Código intermediário (bytecode)



JIT – Just-in-time compilation

- Vantagem: acelera a execução na máquina alvo



Introdução

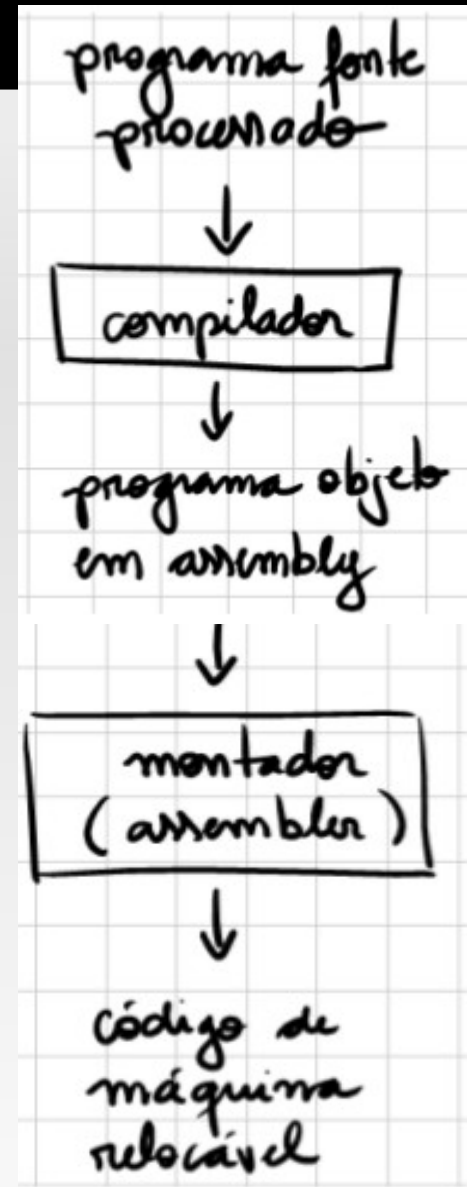
- Programas relacionados a compiladores
 - Interpretador X Compilador
 - Levantamento realizado em 1ºSem/2009

Linguagens	Interpretadas	Compiladas	Interpretadas e/ou Compiladas
Quantidade	18 (44%)	13 (32%)	10 (24%)

- Interpretadas
 - Python, Lua, PHP, Octave, Perl, R, Euphoria, Ruby, Basic, Awk, Scheme, Whenever, Chef, etc.
- Compiladas
 - C, C#, C++, D, B, Ada, Fortran, Algol, Cobol, Pascal, etc.
- Interpretadas e/ou Compiladas
 - Java, Smalltalk, Actionscript, Haskell, Eiffel, Clipper, Forth, etc.

Introdução

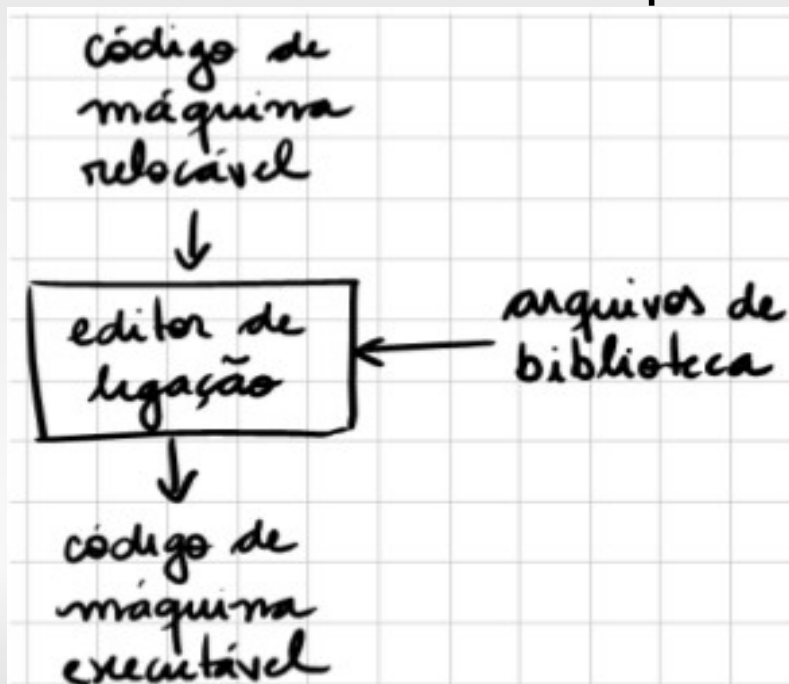
- Programas relacionados a compiladores
 - Interpretador
 - Montador (assembler)
 - Traduz a linguagem de montagem de um computador em particular
 - Mapeia instruções em linguagem simbólica (assembly) para instruções de máquina, geralmente uma relação um-para-um



Fonte: Notas de
aula do Prof.
Daniel Lucrédio

Introdução

- Programas relacionados a compiladores
 - Interpretador
 - Montador
 - Organizador (ou editor de ligação)
 - Coleta o código compilado ou montado separadamente e coloca tudo em um arquivo executável



Fonte: Notas de aula do Prof. Daniel Lucrédio

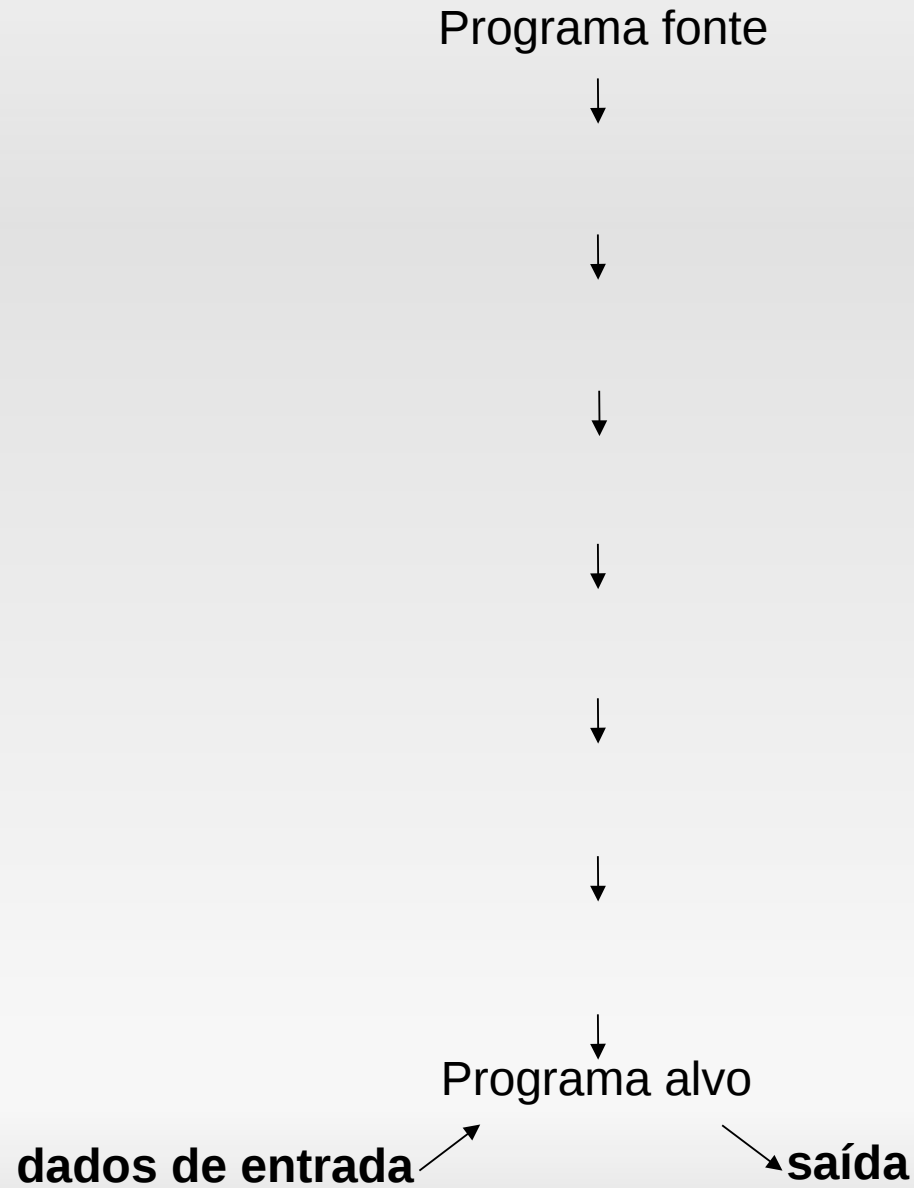
Introdução

- Programas relacionados a compiladores
 - Interpretador
 - Montador
 - Organizador
 - Carregador
 - Carrega um código realocável resolvendo os endereços relativos a um dado endereço de base ou inicial

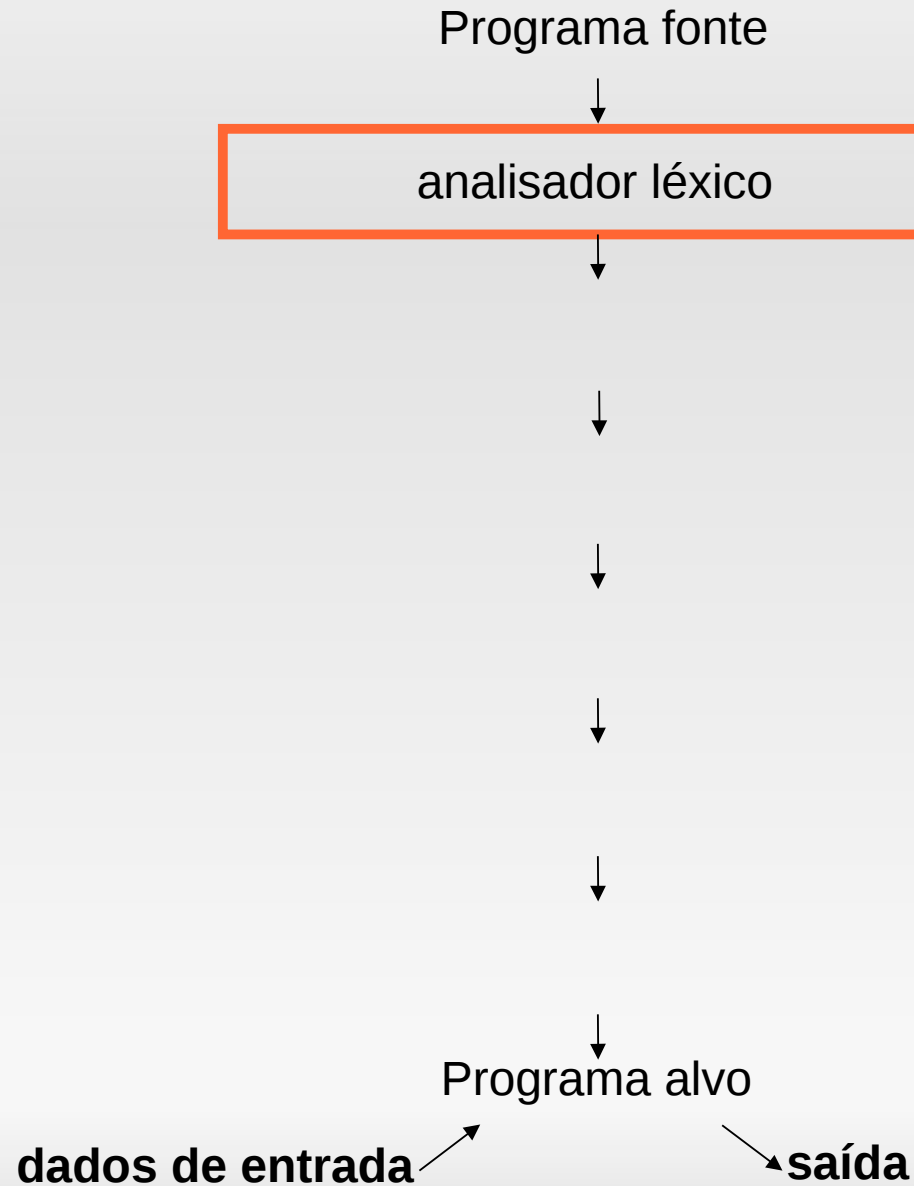
Introdução

- Programas relacionados a compiladores
 - Interpretador
 - Montador
 - Organizador
 - Carregador
 - Depurador
 - Determina erros de execução em um programa compilado

Processo de Tradução



Processo de Tradução



Processo de Tradução

- Etapas de análise
 - Análise léxica
 - O fluxo de caracteres que constitui o programa é lido da esquerda para a direita e esses caracteres são agrupados em *tokens*

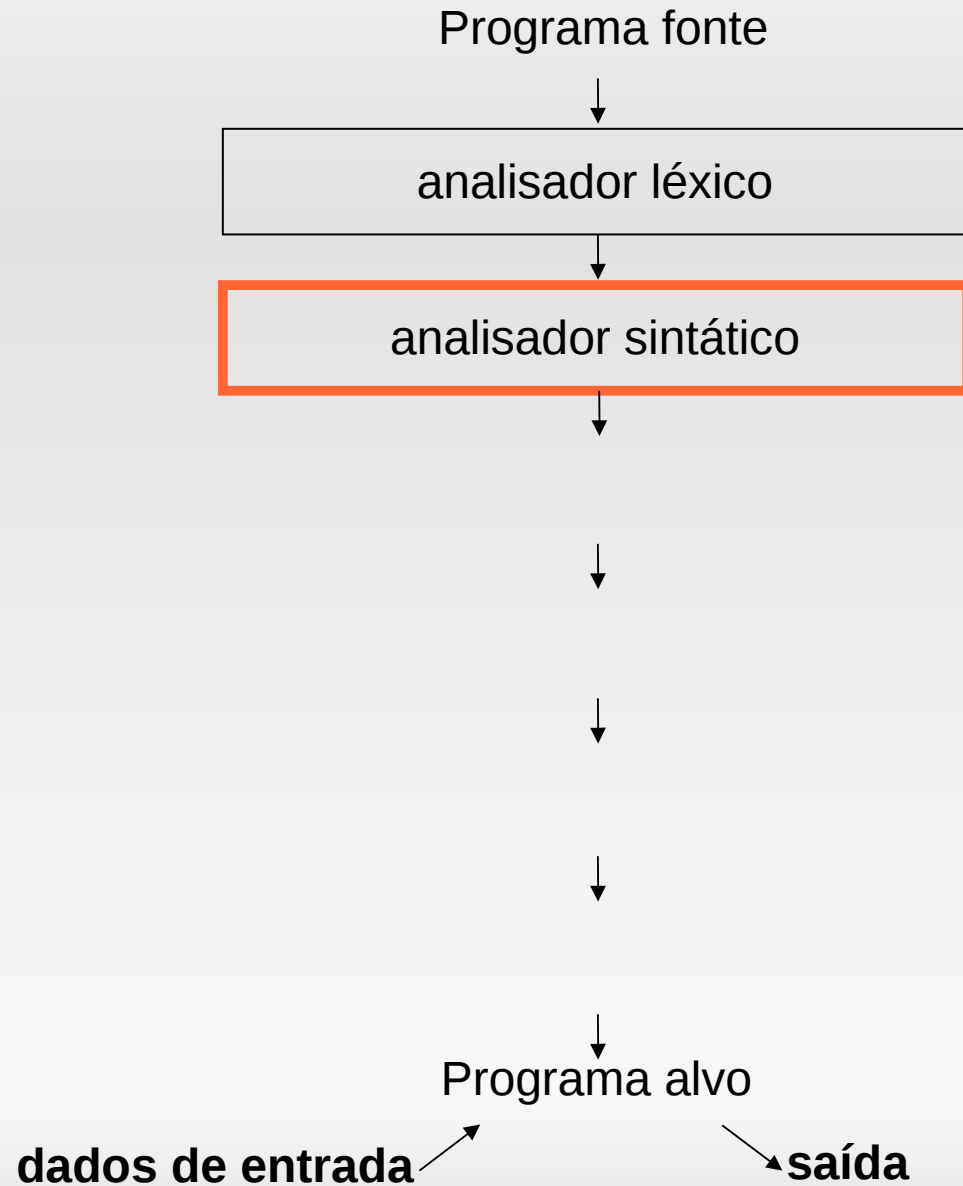
x:=y+z*4



<id, x> <:=, :=> <id, y> <op, +> <id, z> <op, *> <num, 4>

→ Expressões regulares e autômatos

Processo de Tradução



Processo de Tradução

- Etapas de análise
 - Análise sintática
 - Os caracteres ou *tokens* são agrupados hierarquicamente em coleções aninhadas com significado coletivo

$\langle id, x \rangle \langle :=, := \rangle \langle id, y \rangle \langle op, + \rangle \langle id, z \rangle \langle op, * \rangle \langle num, 4 \rangle$

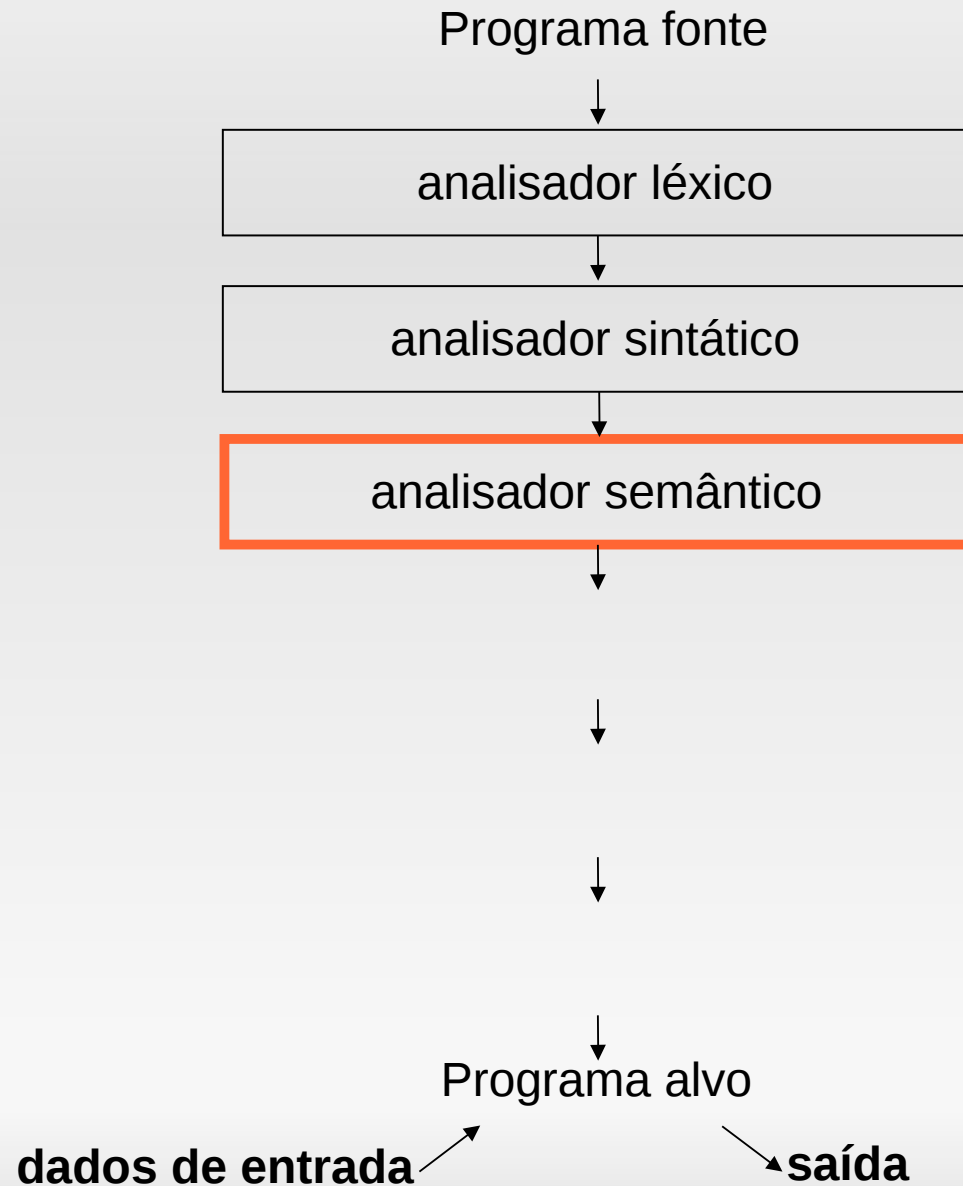


comando_atribuição $\rightarrow id_1 := id_2 op_1 id_3 op_2 num$

→ Gramáticas Livres de Contexto



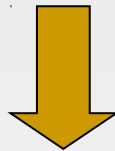
Processo de Tradução



Processo de Tradução

- Etapas de análise
 - Análise semântica
 - Certas verificações são realizadas a fim de se assegurar que os componentes de um programa se combinam de forma significativa

$id_1 := id_2 \ op_1 \ id_3 \ op_2 \ num$



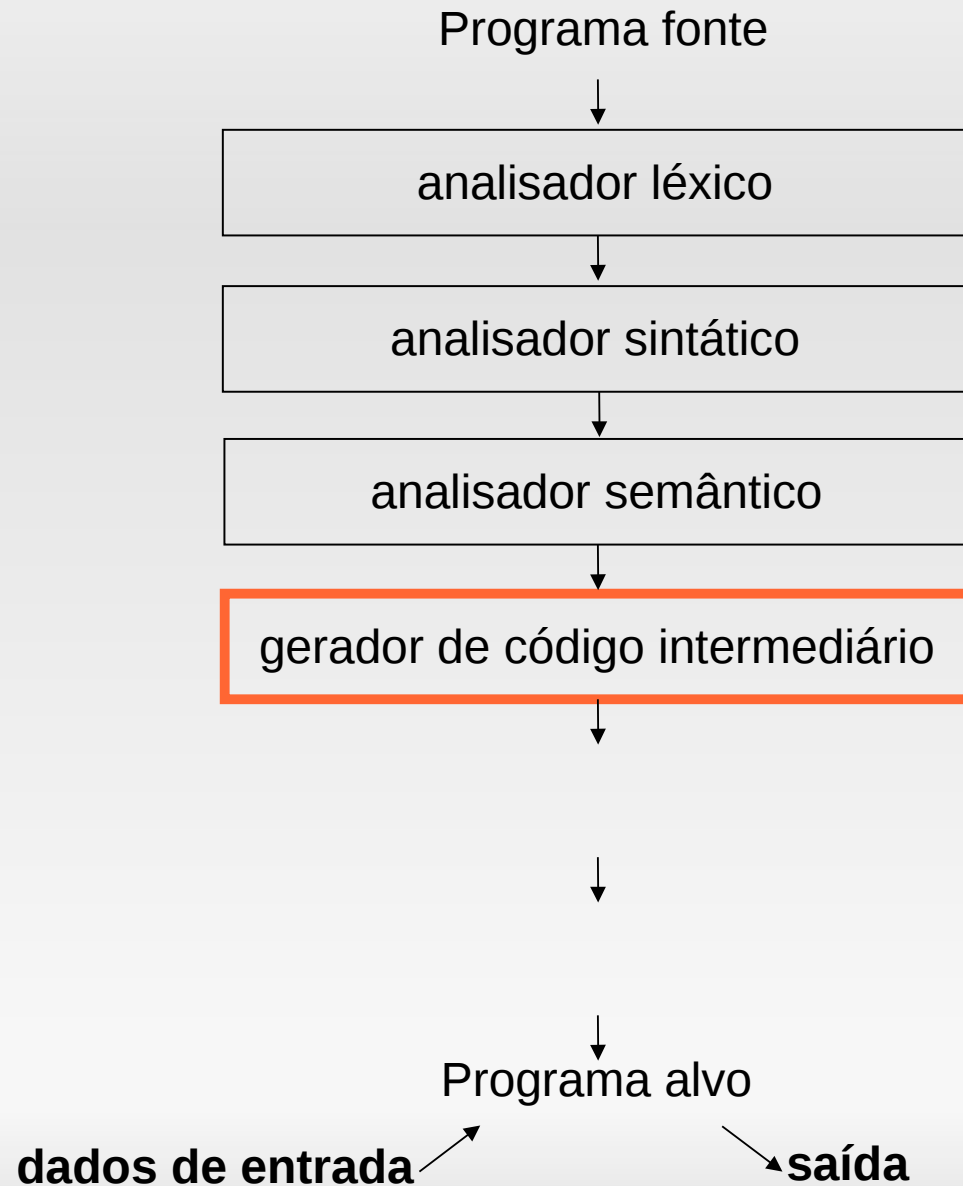
$busca_tabela_símbolos(id_1)=TRUE$

$busca_tabela_símbolos(id_2)=TRUE$

$busca_tabela_símbolos(id_3)=TRUE$

$(id_1)_{int} := (id_2 \ op_1 \ id_3 \ op_2 \ num)_{int}$

Processo de Tradução



Processo de Tradução

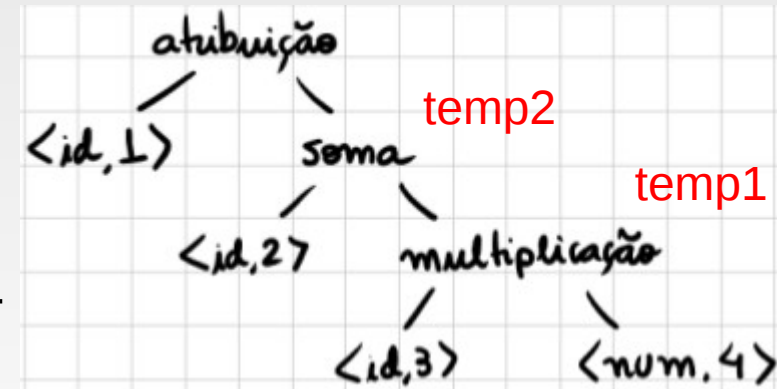
- Etapas de síntese
 - Geração de código intermediário
 - Alguns compiladores geram uma interpretação intermediária explícita para o programa fonte (ex: código de 3 endereços)

$id_1 := id_2 \text{ op}_1 id_3 \text{ op}_2 \text{ num}$

$(x := y + z * 4)$

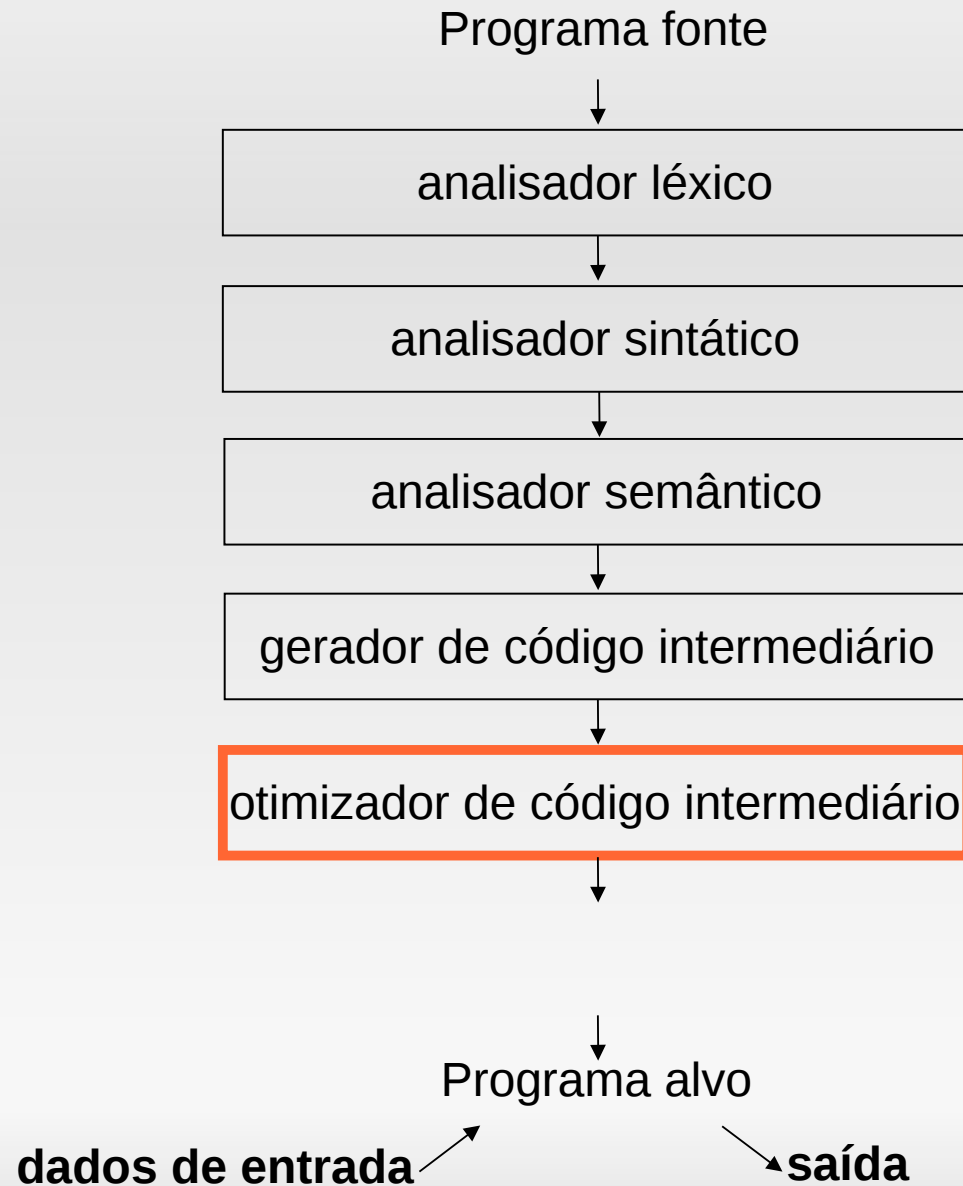


$\text{temp1} := id_3 * 4$
 $\text{temp2} := id_2 + \text{temp1}$
 $id_1 := \text{temp2}$



Fonte: Adaptado de notas de aula do Prof. Lucrédio

Processo de Tradução



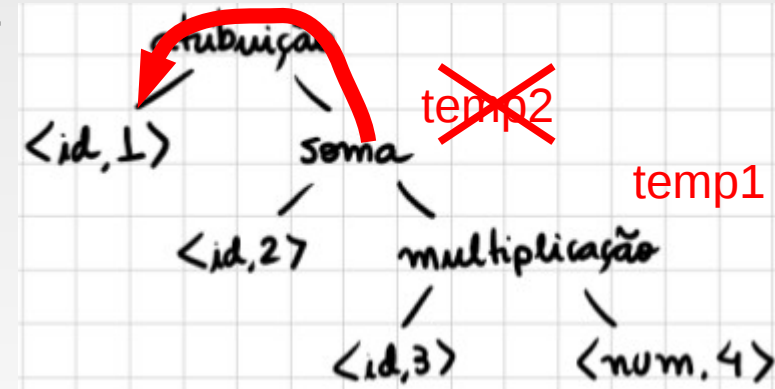
Processo de Tradução

- Etapas de síntese
 - Otimização de código intermediário
 - Tentativa de melhorar o código intermediário de tal forma que resulte em um código de máquina mais rápido em tempo de execução

temp1 := id₃ * 4
temp2 := id₂ + temp1
id₁ := temp2

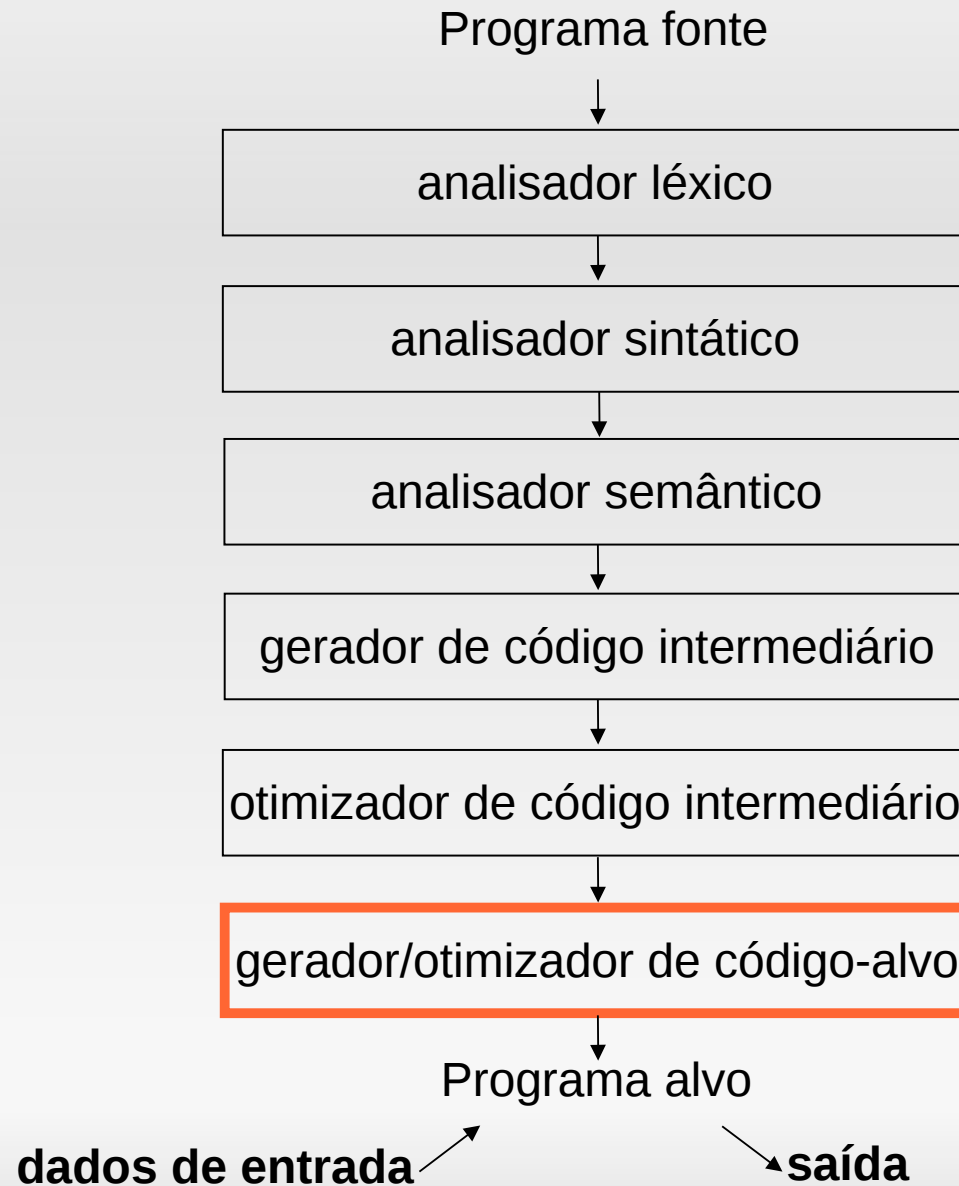


temp1 := id₃ * 4
id₁ := id₂ + temp1



Fonte: Adaptado de notas de aula do Prof. Lucrédio

Processo de Tradução



Processo de Tradução

- Etapas de síntese
 - Geração/otimização de código-alvo
 - As localizações de memória são selecionadas para cada uma das variáveis e as instruções intermediárias são traduzidas numa sequência de instruções de máquina

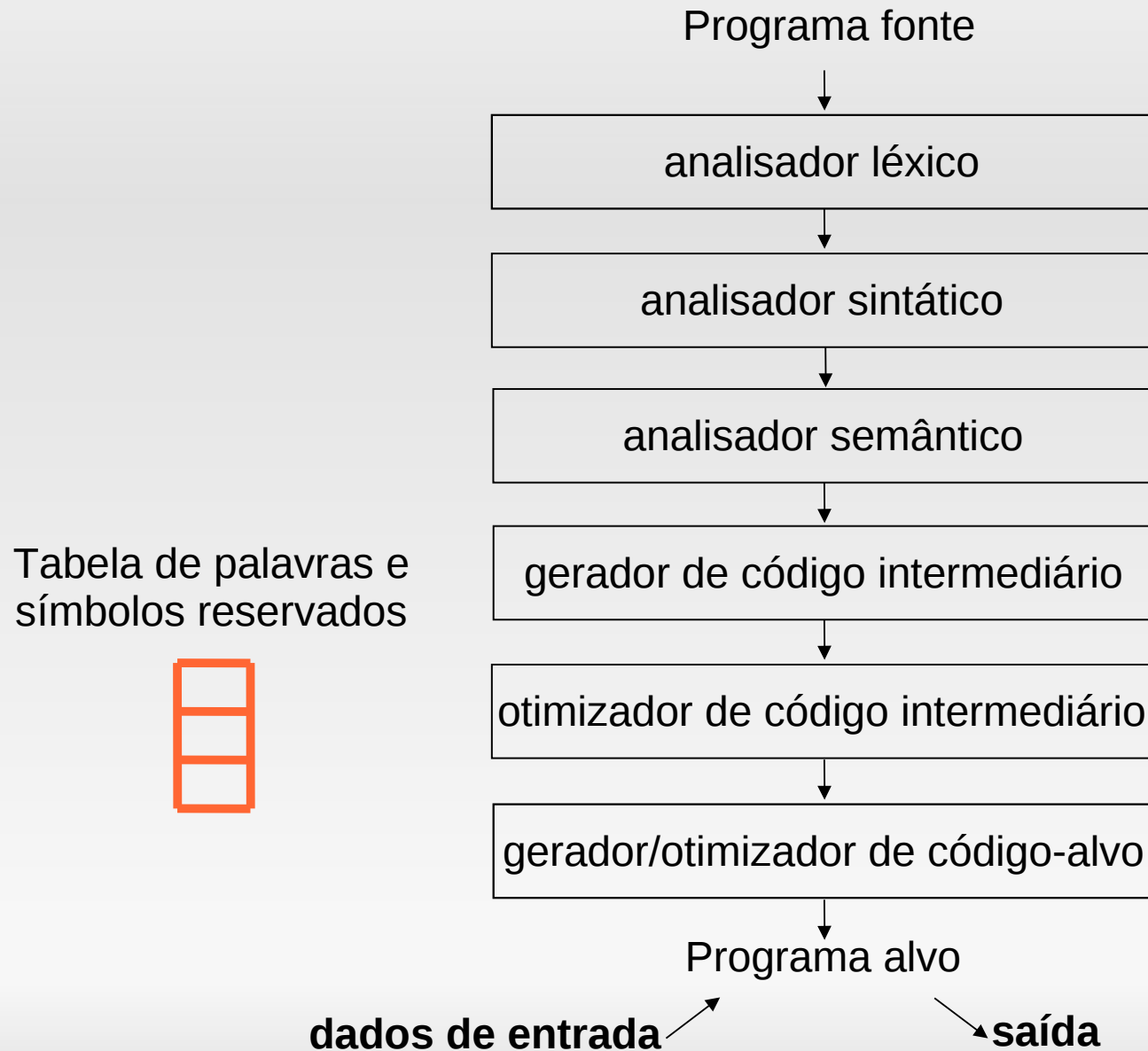
$\text{temp1} := \text{id}_3 * 4$
 $\text{id}_1 := \text{id}_2 + \text{temp1}$



MOV id_3 R1 MULT 4 R1
MOV id_2 R2 ADD R1 R2
MOV R2 id_1

- O compilador tenta melhorar o código-alvo

Processo de Tradução

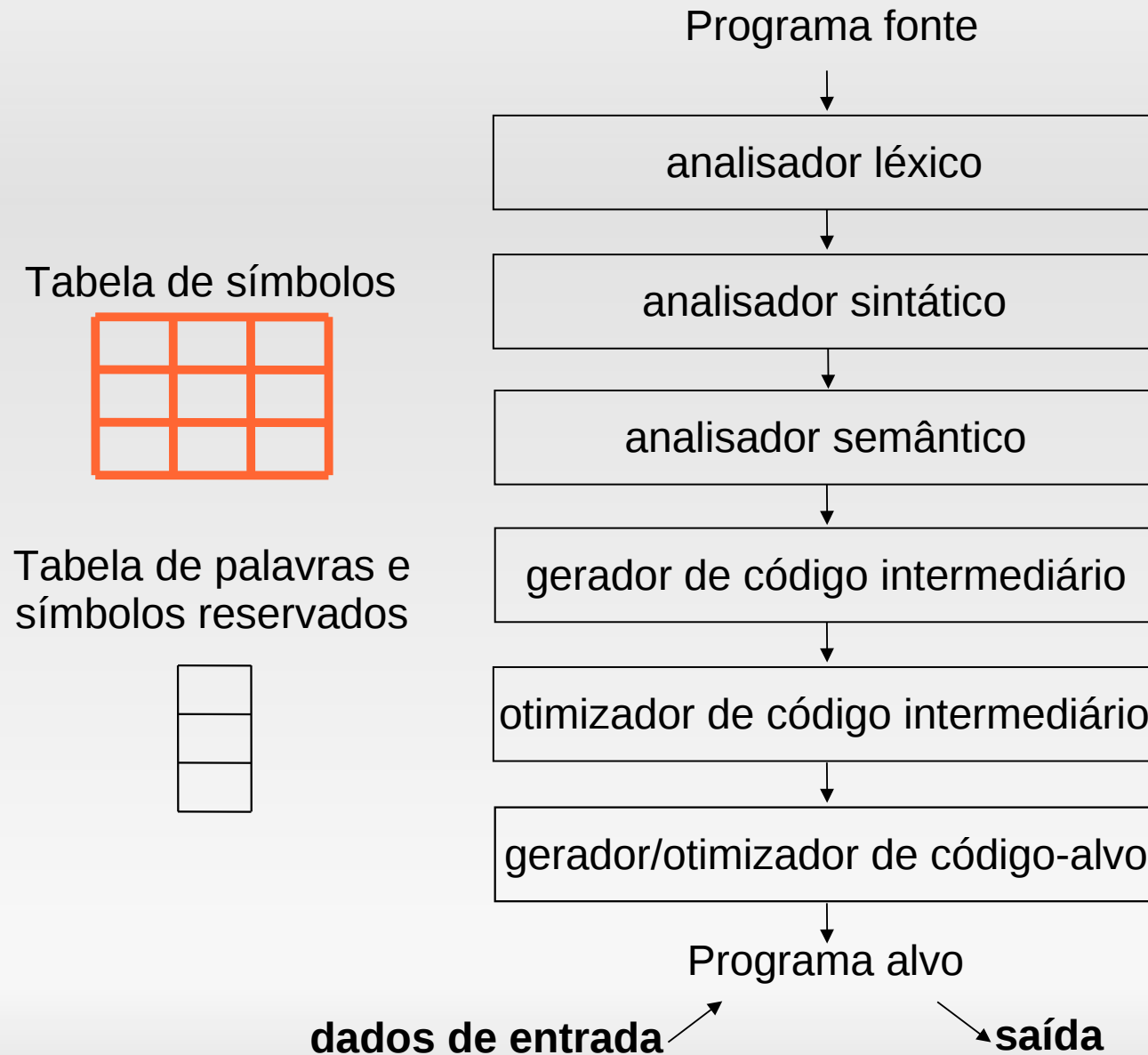


Processo de Tradução

- Principais estruturas de dados
 - Tabela de palavras e símbolos reservados
 - Armazena constantes e cadeias de caracteres usados em um programa
 - Diferencia palavras e símbolos reservados (while, int, :=) de identificadores definidos pelo usuário

int
while
:=
...

Processo de Tradução

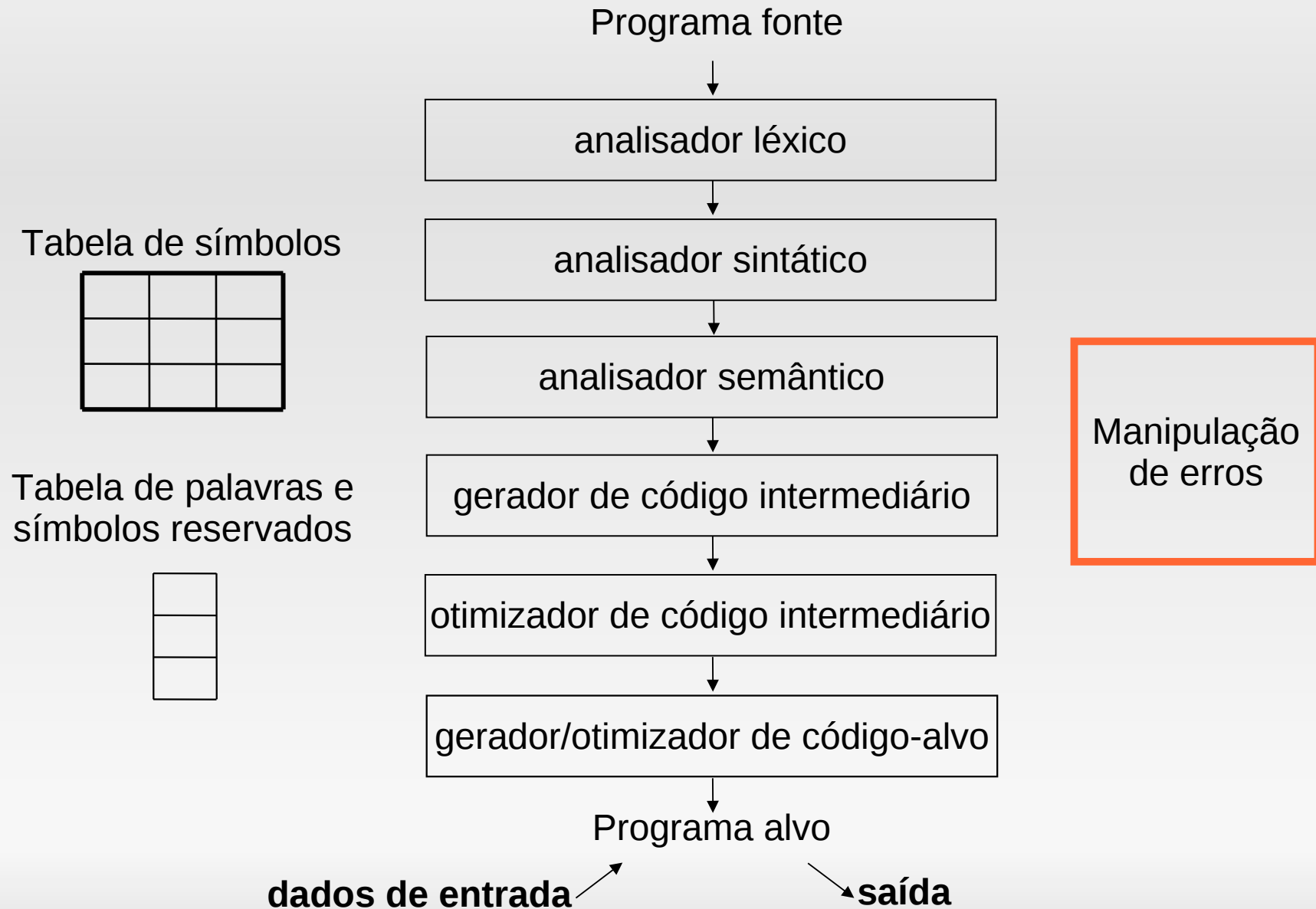


Processo de Tradução

- Principais estruturas de dados
 - Tabela de símbolos
 - Guarda informações de identificadores: funções, variáveis, constantes e tipos de dados
 - Indica, durante a compilação de um programa, o tipo e o valor dos identificadores, escopo das variáveis, número e tipo dos parâmetros de um procedimento, etc.
 - Interage com todas as etapas da compilação

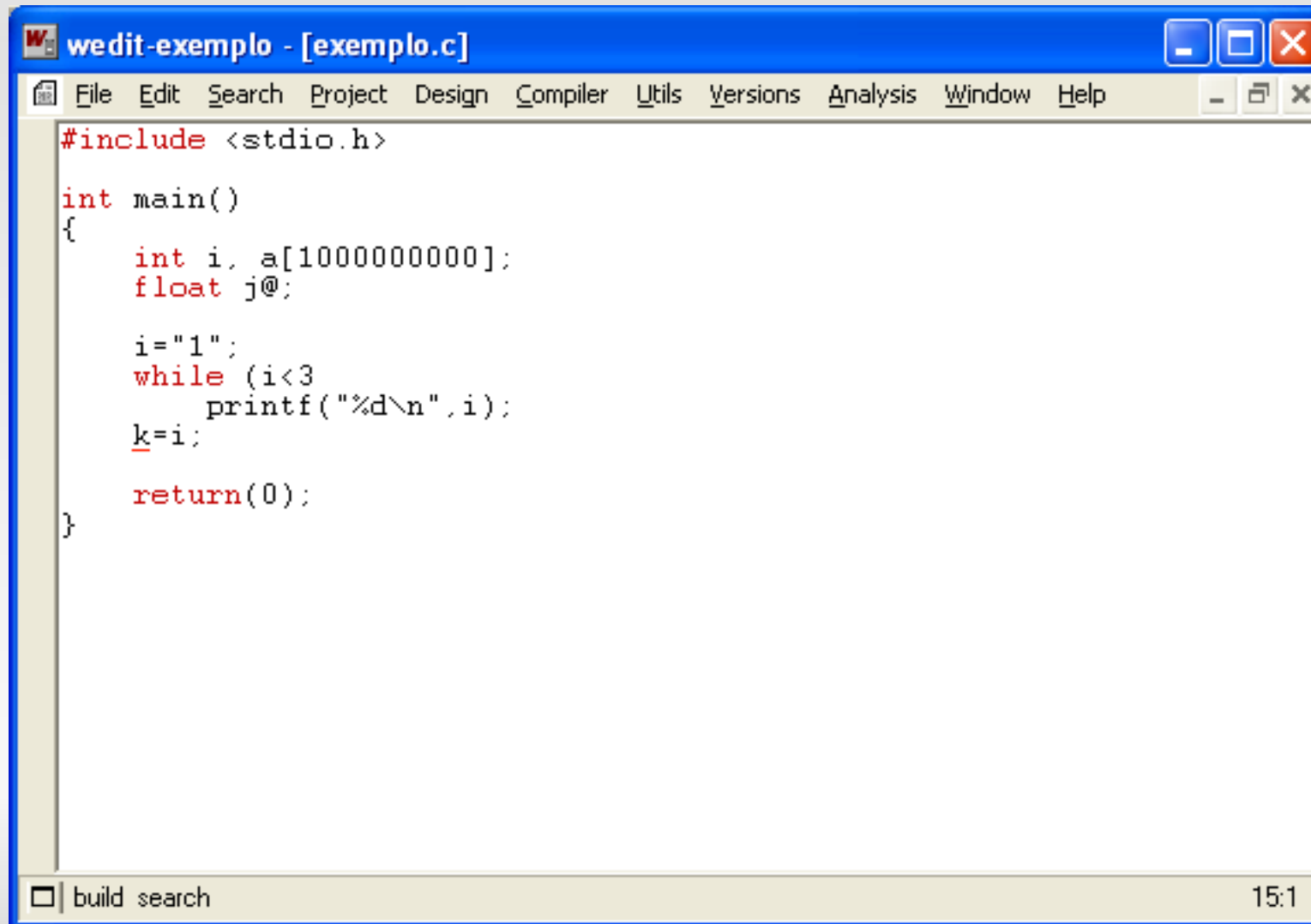
Cadeia	Classe	Tipo	Valor	...
i	var	integer	1	...
fat	proc	-	-	...
...				

Processo de Tradução



Processo de Tradução

- Manipulação de erros



```
#include <stdio.h>

int main()
{
    int i, a[1000000000];
    float j@;

    i='1';
    while (i<3
        printf("%d\\n",i);
    k=i;
    return(0);
}
```

Quais são os erros?

Processo de Tradução

- Manipulação de erros

```
#include <stdio.h>

int main()
{
    int i, a[1000000000];
    float j@;

    i="1";
    while (i<3
        printf("%d\n", i);
    k=i;

    return(0);
}
```

Error c:\temp\exemplo.c: 5 size of 'array of int' exceeds 2147483647 bytes
Error c:\temp\exemplo.c: 6 illegal character '@'
Error c:\temp\exemplo.c: 8 operands of = have illegal types 'int' and 'pointer to char'
Error c:\temp\exemplo.c: 10 syntax error; found 'printf' expecting ')'
Error c:\temp\exemplo.c: 11 undeclared identifier 'k'
Compilation + link time: 0.0 sec, Return code: 1

build search 15:1

Quais são os tipos de erros?

Processo de Tradução

- Manipulação de erros

The screenshot shows a C program in a text editor with several errors highlighted by callouts:

- Viola tamanho de memória: erro de geração de código de máquina** (Violates memory size: machine code generation error) - points to the array declaration `a[1000000000]`.
- Viola formação de identificador: erro léxico** (Violates identifier formation: lexical error) - points to the illegal character `@` in the variable `j@`.
- Viola o significado de i: erro semântico** (Violates the meaning of i: semantic error) - points to the assignment `i="1";`.
- Viola identificadores conhecidos: erro contextual (semântico)** (Violates known identifiers: contextual (semantic) error) - points to the undeclared variable `k` in the assignment `k=i;`.
- Viola formação de comando: erro sintático** (Violates command formation: syntactic error) - points to the missing closing parenthesis in the `while` loop.

The compiler error log at the bottom shows the following messages:

```
Error c:\temp\exemplo.c: 5 size of 'array of int' exceeds 2147483647 bytes
Error c:\temp\exemplo.c: 6 illegal character '@'
Error c:\temp\exemplo.c: 8 operands of = have illegal types 'int' and 'pointer to char'
Error c:\temp\exemplo.c: 10 syntax error; found 'printf' expecting ')'
Error c:\temp\exemplo.c: 11 undeclared identifier 'k'
Compilation + link time:0.0 sec, Return code: 1
```

O que diferencia os tipos de erros?

Processo de Tradução

- Erros e etapas da compilação
 - Lexical: palavras (*tokens*) do programa
 - i, while, =, [, (, <, int
 - Erro: **j@**
 - Sintático: combinação de *tokens* que formam o programa
 - comando_while → while (expressão) comandos
 - Erro: **while (expressão comandos**
 - Semântico e contextual: adequação do uso
 - Tipos semelhantes em comandos (atribuição, por exemplo), uso de identificadores declarados
 - Erros: **i = "1", k = i**
 - Geração de código: especificidades da máquina-alvo e sua linguagem
 - Alocação de memória, uso de registradores
 - Erro: **a[1000000000]**

Processo de Tradução

- Passadas
 - Cada uma das vezes que o programa-fonte é processado antes de gerar o código final
 - Geralmente uma passada engloba diversas etapas
 - Compilador de uma passada (ex: Pascal, C)
 - Todas as etapas ocorrem uma única vez
 - Compilação eficiente, mas código-alvo menos eficiente
 - Analisador sintático gerencia o processo de tradução
 - Compilador de mais de uma passada
 - A maioria dos compiladores com otimização:
 - Uma passada para análise léxica + análise sintática
 - Uma passada para análise semântica + otimização intermediária
 - Uma passada para geração de código + otimização alvo

Introdução

- Perguntas respondidas ou levantadas nesta aula
 - Como ocorre o processo de compilação?
 - Quais são as etapas desse processo?
 - Quais recursos e técnicas são usados em cada etapa?
 - Como os erros são detectados?
 - O que define um programa correto em uma linguagem de programação?
 - Todos os erros são do mesmo tipo ou há diferenças?
 - Um erro influencia outro?
 - Como detectar todos os erros?