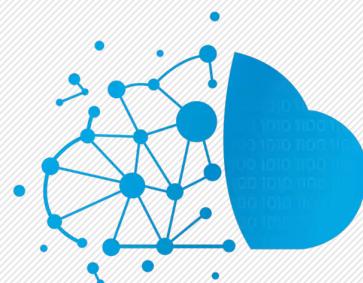


Compiladores

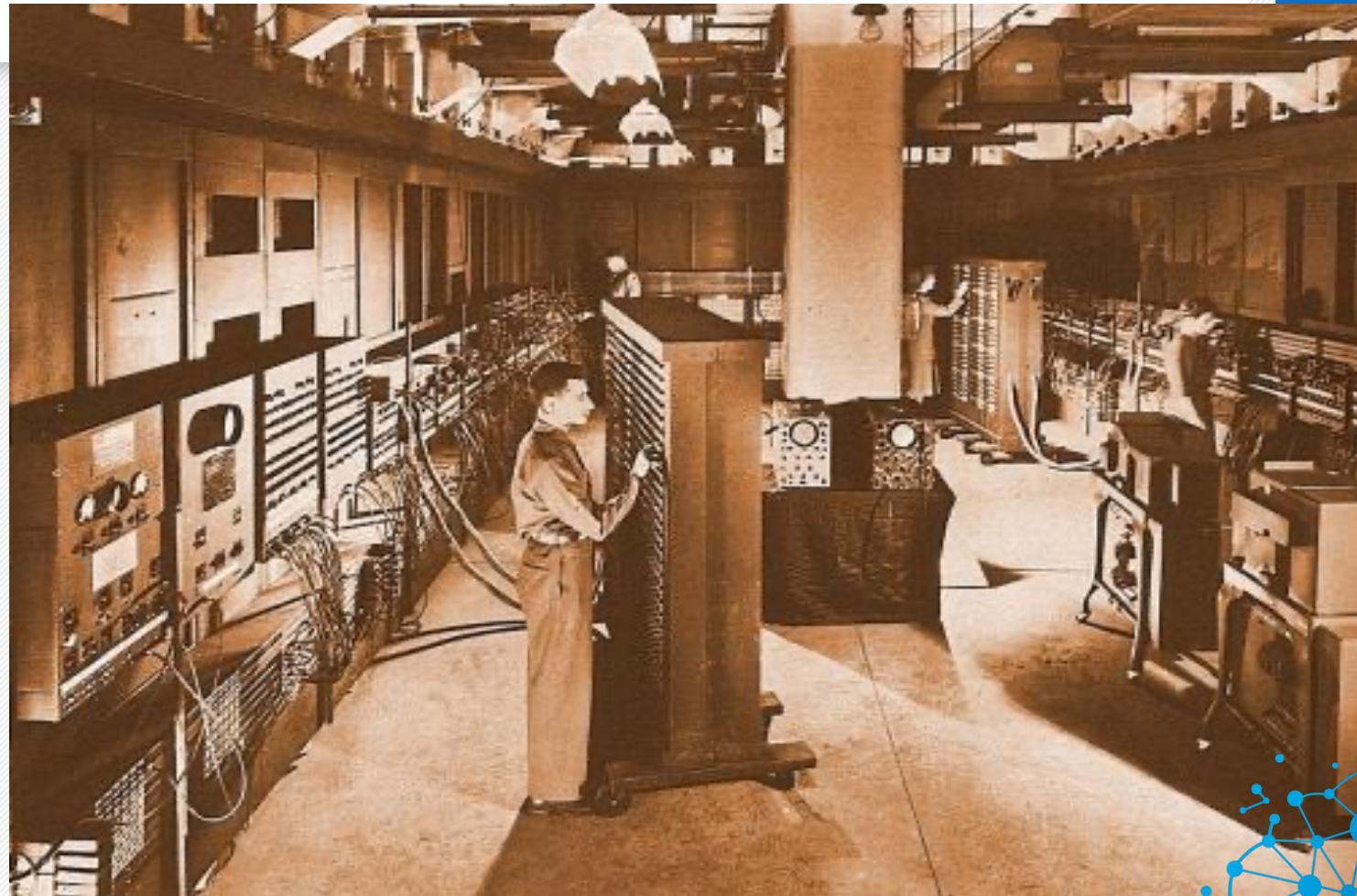


De onde vem o Compilador?

Quem são?
Onde vivem?
Para que servem?
Como funcionam?

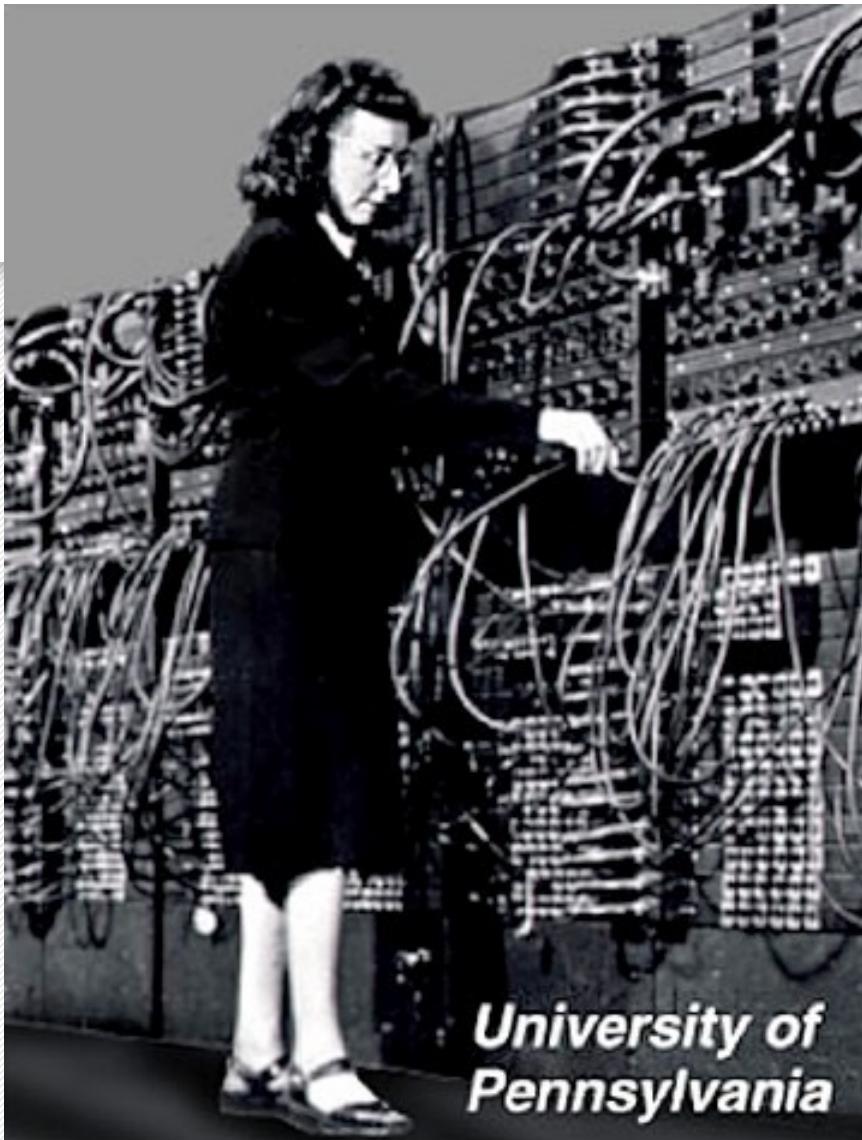


De onde vem o Compilador?



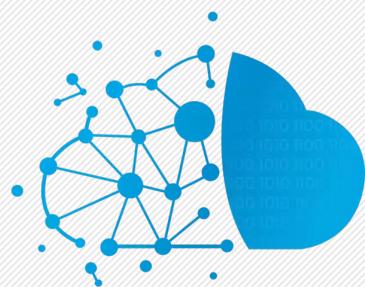
ENIAC - 1946





*University of
Pennsylvania*

ENIAC - 1946



Histórico

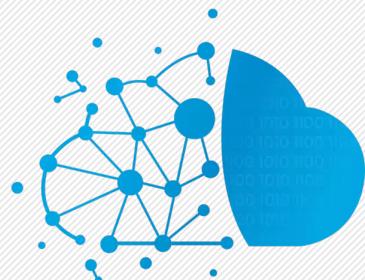
A construção de compiladores tem sido alvo de estudos desde o início dos anos 50 com a criação das primeiras linguagens de programação

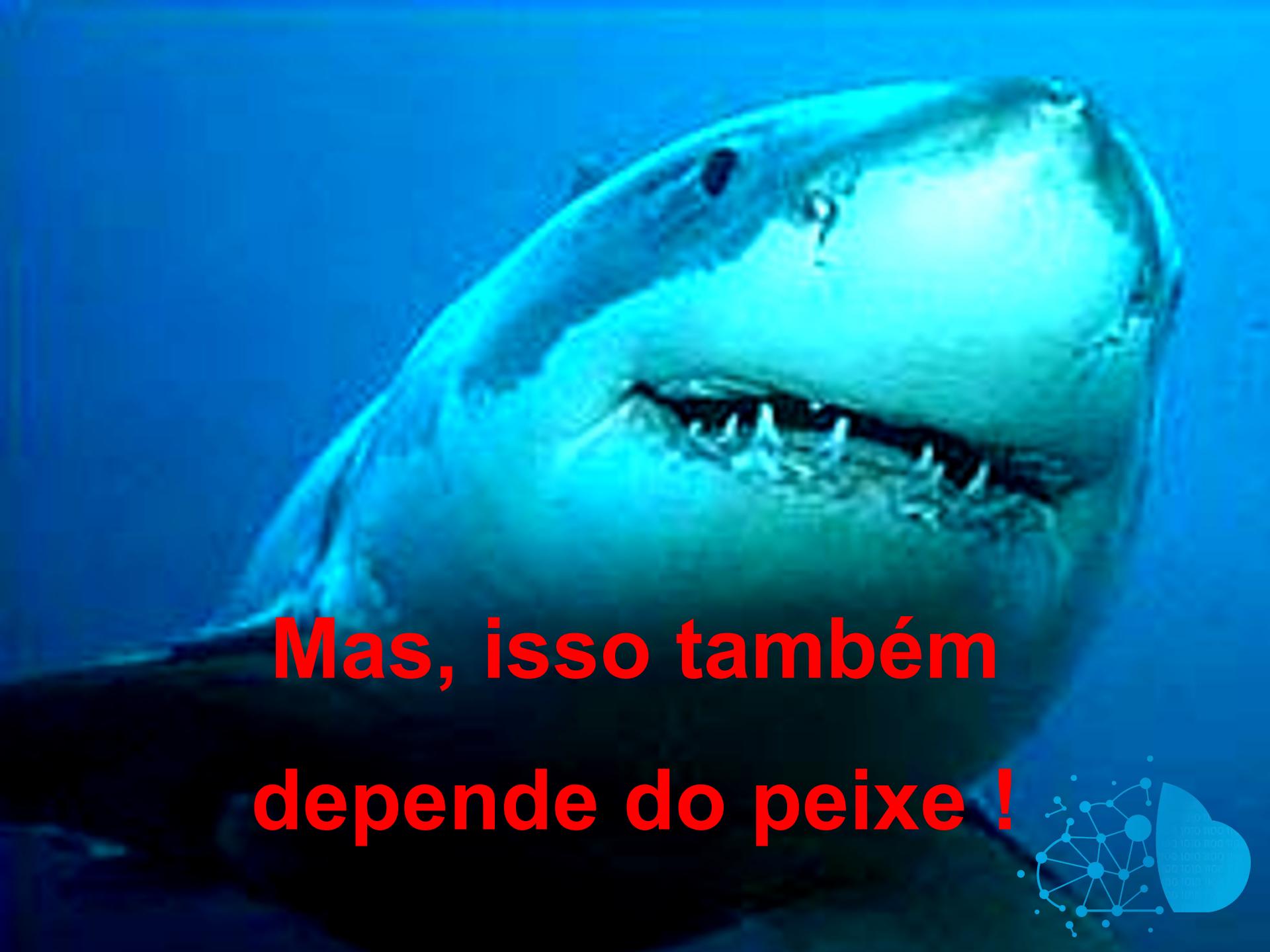


!!! Desafiador !!!

Tradicionalmente a construção de compiladores é considerada uma tarefa de alta complexidade

Mas na verdade é fácil como pescar ...





**Mas, isso também
depende do peixe !**

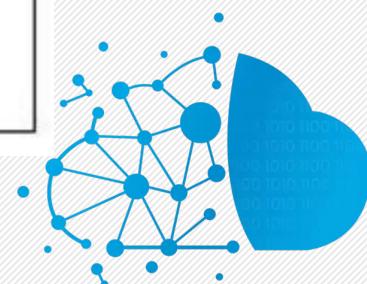
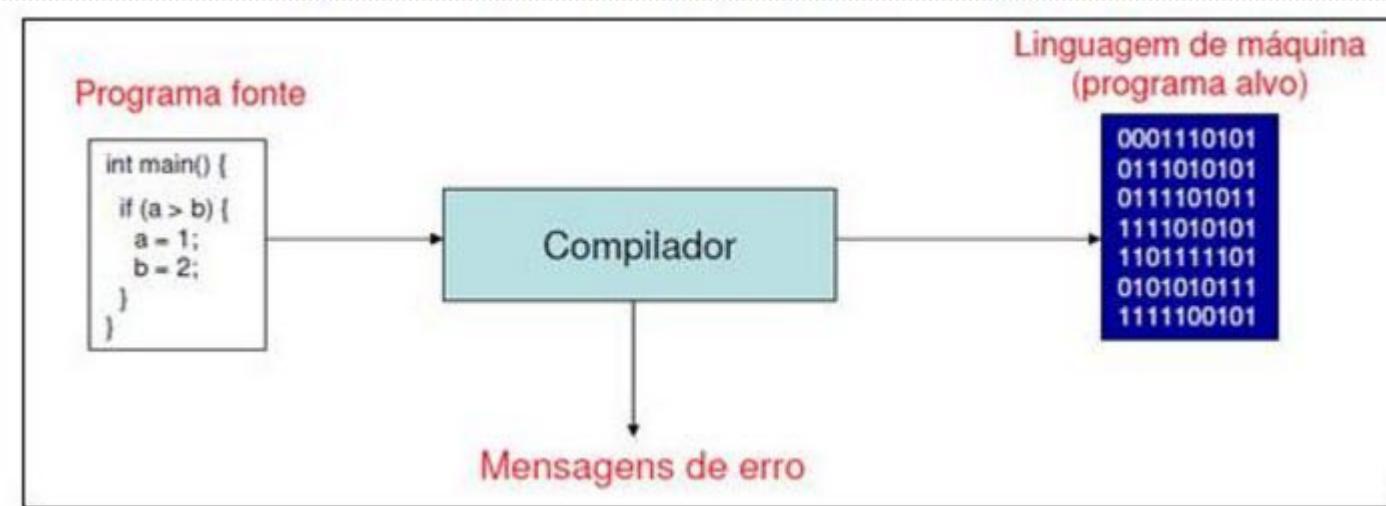


Projeto de Compiladores



Tradução de linguagens

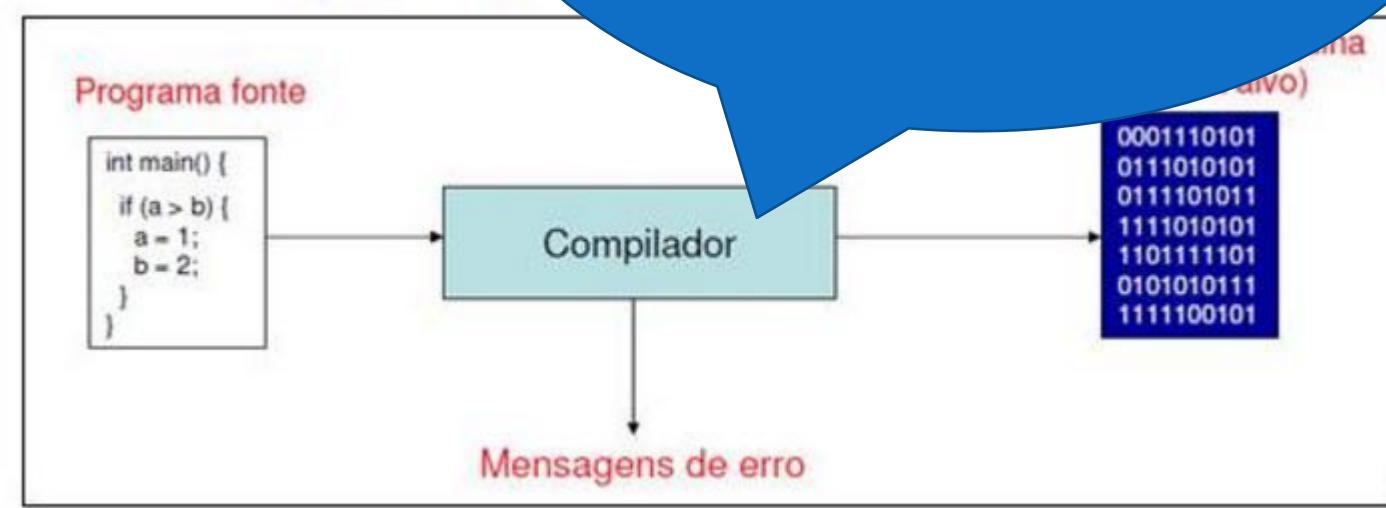
- Os comandos escritos em linguagens de alto nível precisam ser convertidos para linguagens de máquina (códigos binários)



Tradução de linguagens

- Os comandos em linguagem de programação precisam ser convertidos para máquina (códigos binários)

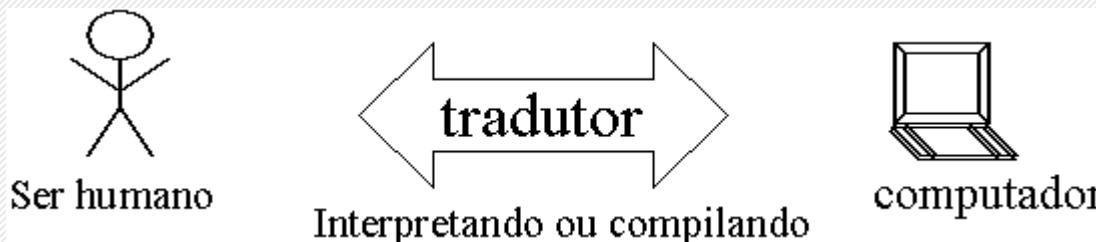
O que há
nesta caixa???



Tradução de linguagens

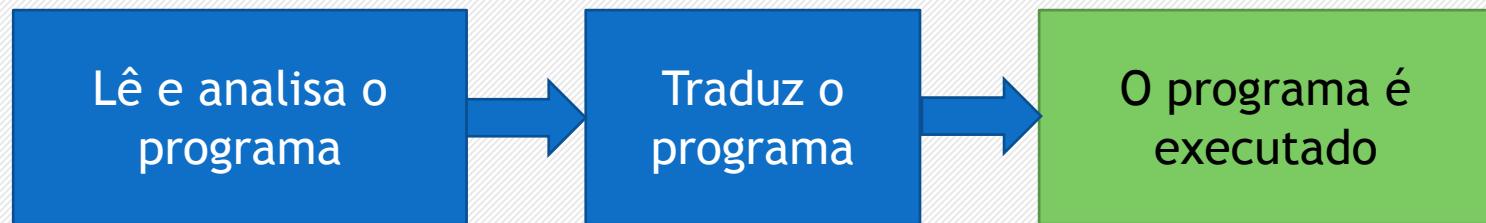
Existem duas abordagens básicas para a tradução de linguagens de alto nível para linguagens de máquina:

- Compilação
- Interpretação



Compilação x Interpretação

- Compilador (*Compile-time*)



- Interpretador (*Run-time*)



Compilação x Interpretação

	Vantagens	Desvantagens
Compiladores	Execução mais rápida	Várias etapas de tradução
	Permite estruturas de programação mais completas para a sua execução	Programação final é maior necessitando mais memória
	Permite a otimização do código fonte	Processo de correção de erros e depuração é mais demorado
Interpretadores	Depuração do programa é mais simples	Execução do programa é mais lenta
	Consome menos memória	Estruturas de dados demasiado simples
	Resultado imediato do programa ou rotina desenvolvida	Necessário fornecer o programa fonte ao utilizador



Compilação x Interpretação

C++



Definição de compilador

- Um programa que lê um programa escrito numa linguagem (**a linguagem fonte**) e o traduz num programa equivalente numa outra linguagem (**a linguagem alvo**).



arquivo.java



arquivo.class



Definição de compilador

- Um programa que lê um programa escrito numa linguagem (**a linguagem fonte**) e o traduz num programa equivalente numa outra linguagem (**a linguagem alvo**).



arquivo.cpp



arquivo.o

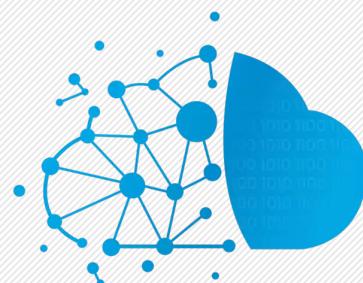
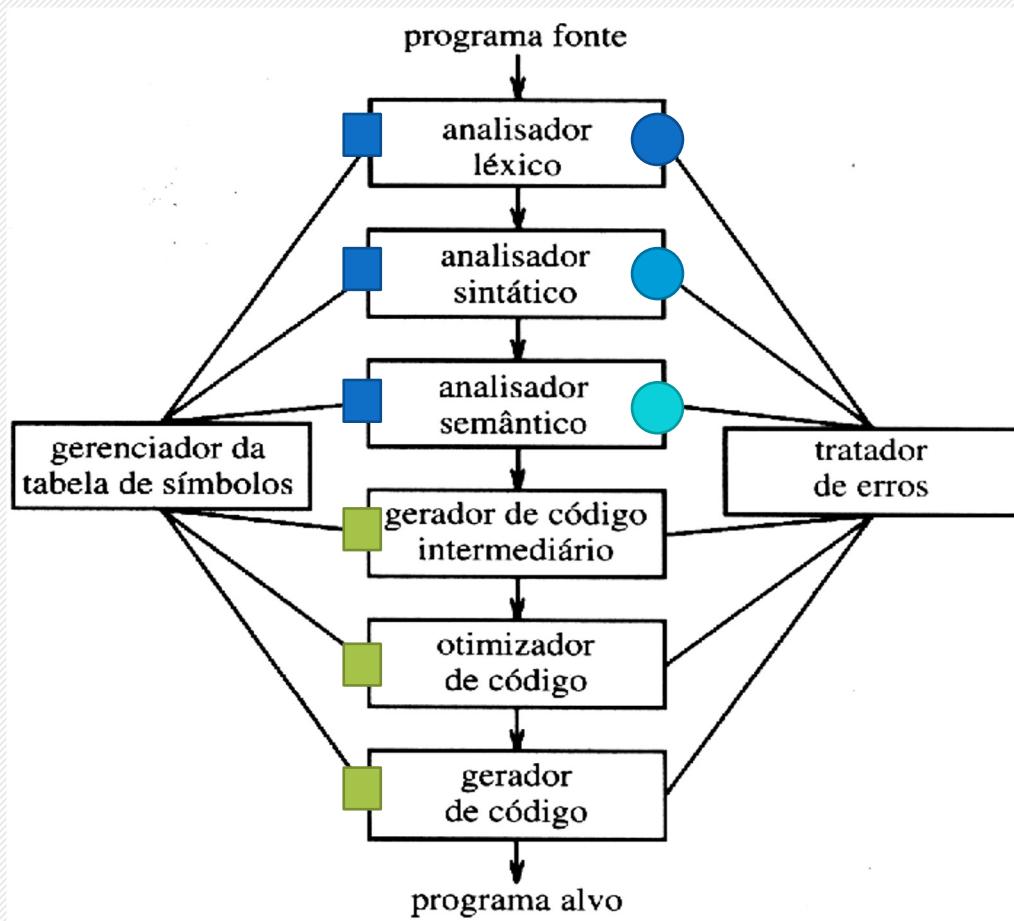
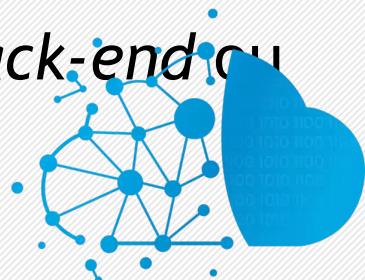


Diagrama da compilação



- Análise linear
- Análise hierárquica
- Análise semântica
- Fase de *front-end* ou análise
- Fase de *back-end* ou síntese



Análise léxica

O fluxo de caracteres que compõem o programa fonte é lido e agrupado em sequências.

Principais tarefas:

1. Abrir o programa feito na linguagem-fonte
2. Ler caractere a caractere, formando sequências
3. Retornar as sequências identificadas
4. Fechar o arquivo



Análise léxica

C Code

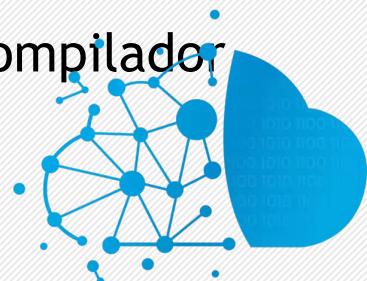
```
while (count <= 100) { /* some loop */  
    count++;  
    // Body of while continues  
    ...
```

tokenizing

Tokens

```
while  
(  
count  
<=  
100  
)  
{  
count  
++  
;  
...
```

Um *token* é a menor informação representativa de um compilador.

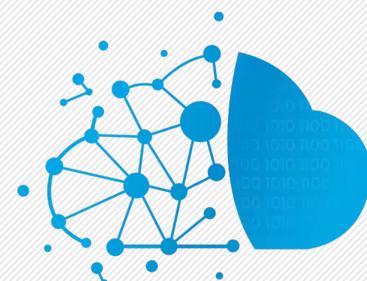


Análise sintática

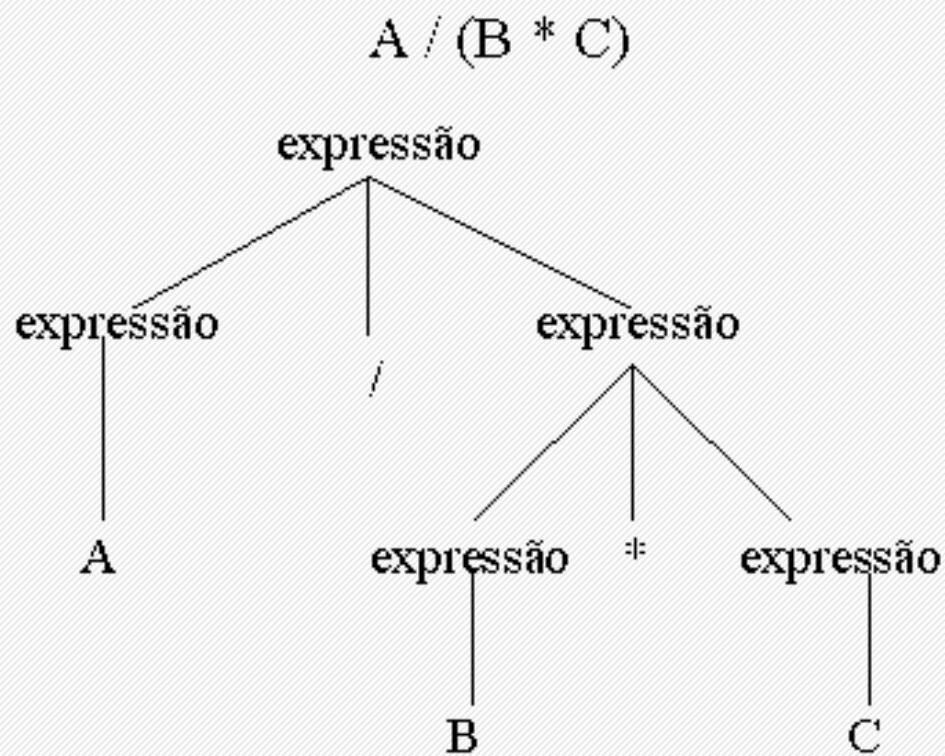
Na qual *tokens* são agrupados hierarquicamente em coleções aninhadas com significado coletivo.

Principais tarefas:

1. Pedir ao analisador léxico um *token* de cada vez, tentando “casá-lo” com a regra de produção da gramática.
2. Retornar a sequência de reconhecimentos válidos pela gramática.



Análise sintática



Análise semântica

Na qual certas verificações são realizadas a fim de assegurar que os componentes de um programa se combinam de forma significativa.

Principais tarefas:

1. Identificar variáveis não declaradas, violações de escopo, operandos ou tipos incompatíveis, etc..
2. Produzir uma saída semelhante à entrada, mas já validada.



Gerador de código intermediário

Na tradução de um programa fonte para um código objeto, o compilador pode produzir uma ou mais representações intermediárias.

Exemplo: *Three Address Code*, com uma operação, dois operandos de entrada e um operando (temporário) de saída.

Operação	Operando resultado	Operando A	Operando B
----------	--------------------	------------	------------



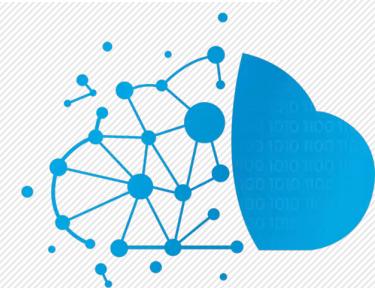
Gerador de código intermediário

Exemplo:

$a = b * c + b * d$

→ *Gerador* →

*	t_1	b	c
*	t_2	b	d
+	t_3	t_1	t_2
=	a	t_3	



Otimizador de código

São realizadas transformações no código intermediário com o objetivo de produzir um código melhor.

Principais técnicas:

- Inibição de segmentos de código
- Inibição do uso de variáveis declaradas e não utilizadas
- Otimização de laços
- Verificação de dependências em pipelines



Otimizador de código

Exemplo:

$$a = (b*c) + 0 + (b*c)$$

*	t_1	b	c
+	t_2	t_1	0
*	t_3	b	c
+	t_4	t_2	t_3
=	a	t_4	

→ *Otimizador* →

*	t_1	b	c
+	t_2	t_1	t_1
=	a	t_2	



Gerador de código

Nesta etapa é gerada a saída do programa na linguagem alvo.

O código alvo é responsável por:

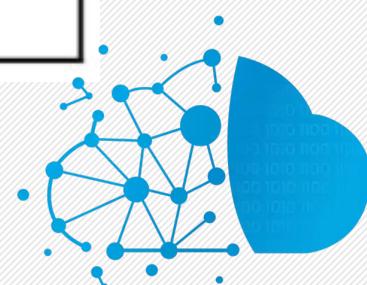
- Alocar registradores da máquina para cada variável;
- Gerar instruções (em código alvo) para cada operação;
- Traduzir a sequência de instruções da representação intermediária em uma sequência de instruções da linguagem alvo de forma a desempenhar as mesmas tarefas.



Gerenciador da tabela de símbolos

Estrutura de dados responsável por armazenar informações de todos os identificadores (variáveis, palavras-chave) durante a compilação.

Cadeia	Token	Categoria	Tipo	Valor	...
i	id	var	integer	1	...
fat	id	proc	-	-	...
2	num	-	integer	2	...
...					

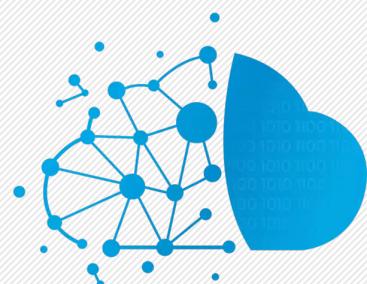


Tratador de erros

Parte responsável do compilador por tratar erros em qualquer fase da compilação.

Estratégias básicas:

- **Abortar**, ou seja, parar a compilação em qualquer fase caso seja encontrado algum erro;
- **Recuperar**, tentar continuar a compilação, desconsiderando algum fato.



Pré-processador

Um pré-processador é um programa separado que é ativado pelo compilador antes do início da tradução.

Faz a eliminação de construções de nível mais alto através da inserção de textos equivalentes na linguagem fonte.

```
#define soma(x, y) (x + y)
int main()
{
    cout << soma (3,5);
}
```

→ Pré-processador →

```
int main()
{
    cout << (3+5);
}
```



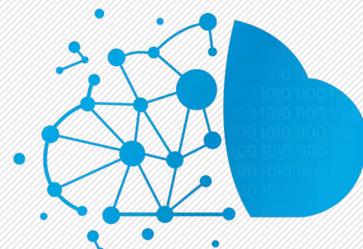
Bibliotecas

Coleções de códigos utilizados no desenvolvimento de programas.

Estáticas - são inseridas antes do processo de compilação.

Dinâmicas - são inseridas em alguma fase da compilação/execução.

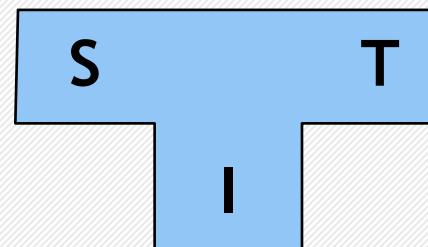
As bibliotecas dinâmicas são mais dependentes do sistema operacional que as estáticas, pois já estão "pré-compiladas", enquanto que as estáticas podem ser compiladas juntamente com o código do programa.



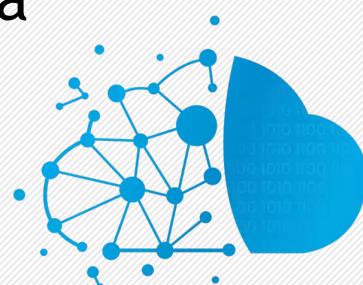
Abordagens para a construção de compiladores

Um compilador é caracterizado por três linguagens que podem ser todas completamente diferentes:

- A linguagem fonte **S** que ele compila;
- A linguagem alvo **T** que ele gera;
- A linguagem de implementação **I**.

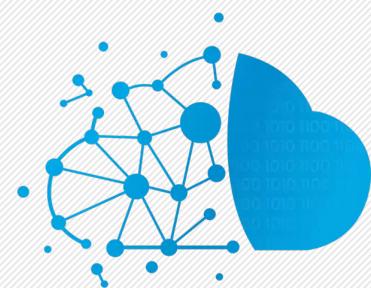
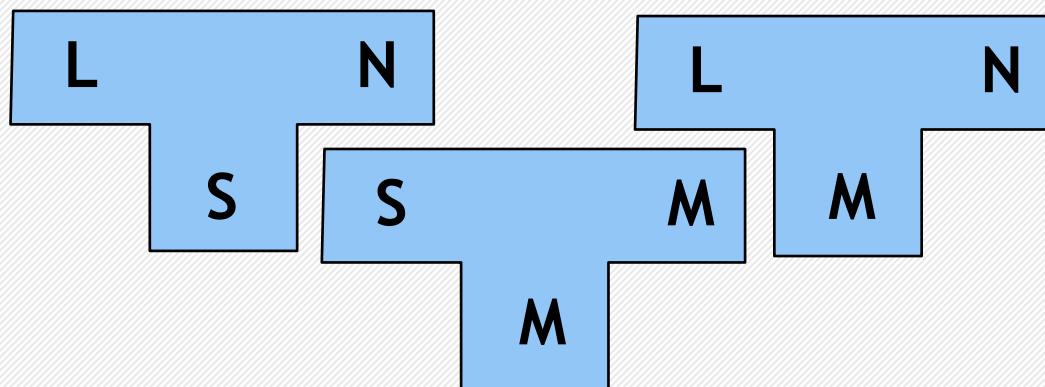


Um **compilador cruzado** é aquele que roda numa máquina e gera código para outra máquina.



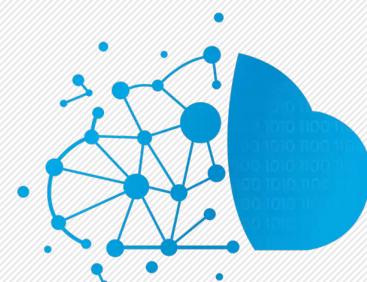
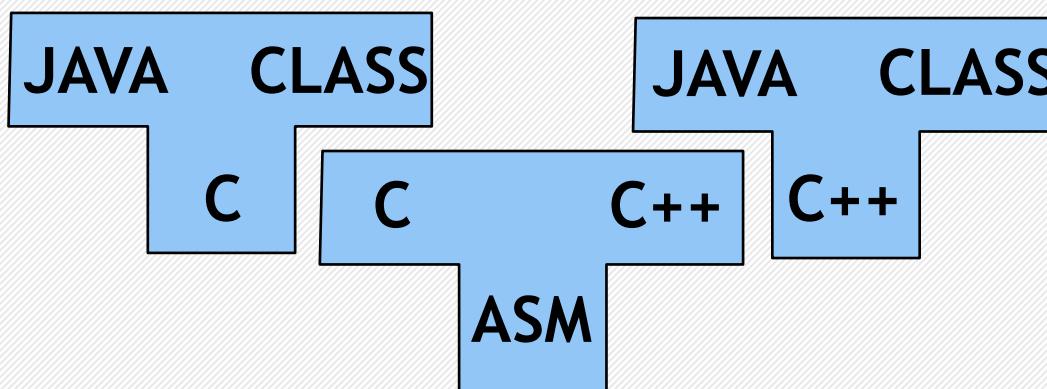
Compilação de compiladores

No exemplo abaixo, um compilador na linguagem M que traduz L para N é gerado a partir de um compilador que realiza a mesma tradução utilizando a linguagem S.



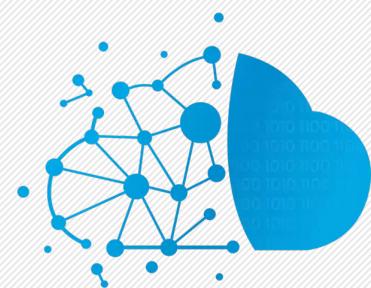
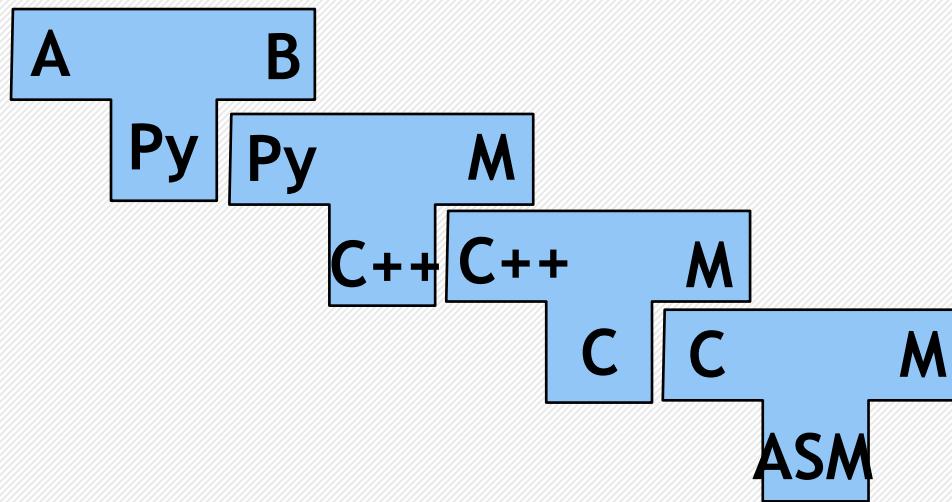
Compilação de compiladores

No exemplo abaixo, um compilador na linguagem C++ que traduz JAVA para CLASS é gerado a partir de um compilador que realiza a mesma tradução utilizando a linguagem C.



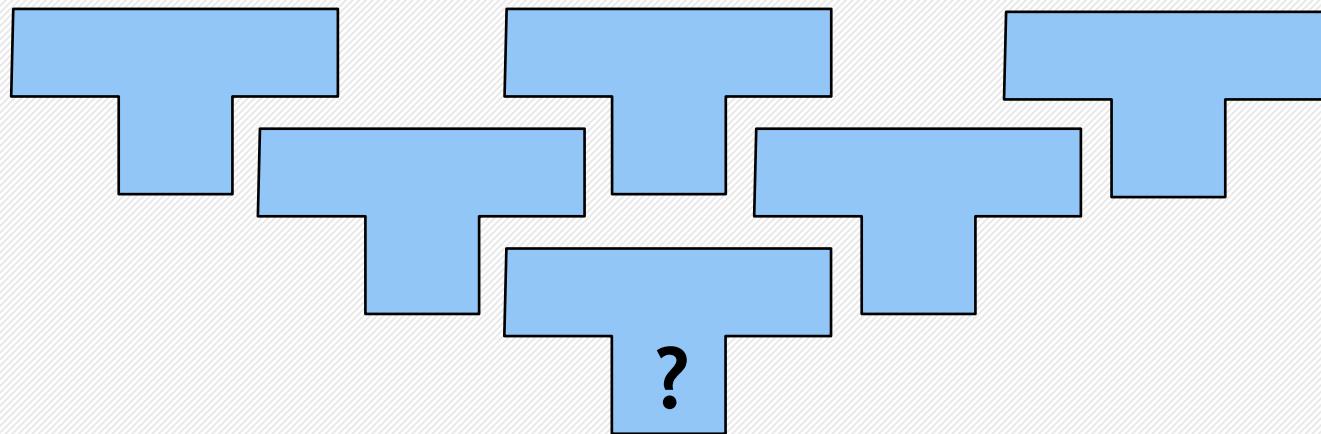
Compilação de compiladores

Cascata de tradução de linguagens de alto nível



Bootstrapping

Como foi compilado o primeiro compilador?



→ Pesquisa sobre a história dos compiladores.



Motivação

Compilador: uma das **principais ferramentas** do cientista do cientista/engenheiro da computação

Técnicas de compilação se aplicam a projetos gerais de programas

- Editores de texto, sistemas de recuperação de informação, reconhecimento de padrões...



Motivação

Usa conceitos de diversas disciplinas

- Algoritmos
- **Linguagens de programação**
- Grafos
- Engenharia de SW
- Arquitetura de Computadores
- **Linguagens formais e autômatos**



Onde estamos?

Novas arquiteturas dominando o mercado

ARM, GPGPU: CUDA x GCN,
NUMA (acesso não-uniforme a memória)

Novas plataformas dominando o mercado

Android, IoT



Exercício

Cada um escolhe uma linguagem de programação e responde:

- Como são as regras para nomes de variáveis?
- Como são os comentários?
- Como são os operadores?
- Que tipos de variáveis possui?
- Quantos tipos de desvio condicional?
- Quantos tipos de laços? Quais?
- Formas de atribuição?
- Compatibilidade entre tipos?



Referências

- LOUDEN, Kenneth C. Compiladores: princípios e práticas. São Paulo, SP: Pioneira Thomson Learning, 2004. 569 p. ISBN 8522104220
- AHO, Alfred V.; VIEIRA, Daniel. Compiladores: princípios, técnicas e ferramentas. 2. ed. São Paulo, SP: Pearson/Addison Wesley, c2007, c2008. x, 634 p. ISBN 9788588639249.
- SEBESTA, Robert W. Conceitos de linguagens de programação. 4. ed. Porto Alegre: Bookman Companhia, 2000. xii, 624p ISBN 8573076089 (broch.)
- VAREJAO, Flavio. Linguagens de programação: Java, C e C++ e outras : conceitos e técnicas. Rio de Janeiro, RJ: Campus, 2004. xvi, 334 p. : (Serie Campus SBC) ISBN 8535213171

