Instituto Superior de Engenharia do Porto

1 - Introdução aos Compiladores

Linguagens e Programação

Ana Madureira
Engenharia Informática
Ano Letivo: 2021/2022

1

2

Compilers Theory

- Compilers are well understood – in theory as well as in practice
- Tools are available to build them based on high level abstractions and formal specification methods
  - Flex - fast lexical analyzer generator. It is a tool to generate lexical analyzers.
  - Bison: is a general-purpose parser generator that converts an annotated context-free grammar into an LALR(1)
    - **lex vs. flex, yacc vs. Bison** ⇒ lex and yacc versus flex and Bison
  - YACC means: Yet Another Compiler Compiler
  - ANTLR: ANother Tool for Language Recognition
- Building a compiler for e.g. Modula or Pascal would take approx. 2 person month of work. The first Fortran compiler was proposed in April 1957.

3

Programming Language Popularity

Programming language rankings

- what the most popular programming languages are
- Sometimes an obsession in the developer community
- a source of constant debate
- many conflicting indices that rank them, each based on its own methodology.
- The issue is not which ranking is the best but, rather, which ranking is the best for a particular question or situation.

https://techbeacon.com/programming-language-rankings-which-ones-matter

4

The History of Programming Languages

In 1940s, programs, including all instructions and memory locations, were written in machine-level programming languages in binary (0s and 1s), and memory had to be manually moved around

- binary code was tedious and error prone
- programs were not easy for humans to read, write, or debug
- a simple algorithm resulted in a lengthy code

In 1950s symbolic programming languages like assembly language were introduced, that used symbolic notation to represent machine language commands.

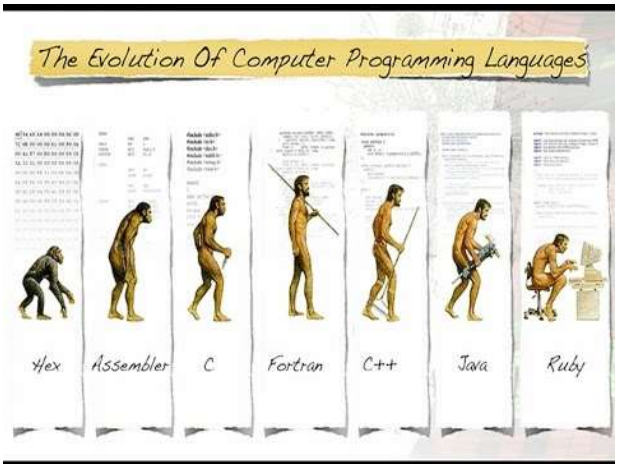
high-level languages (like COBOL, FORTRAN, BASIC, and C) because the program statements were not closely related to the internal characteristics of the computer, making it possible to write programs using familiar terms instead of difficult machine instructions - procedural paradigm

- these languages had compilers

object-oriented programming paradigm attempts to abstract modules of a program into reusable objects(Java and Smalltalk).

5

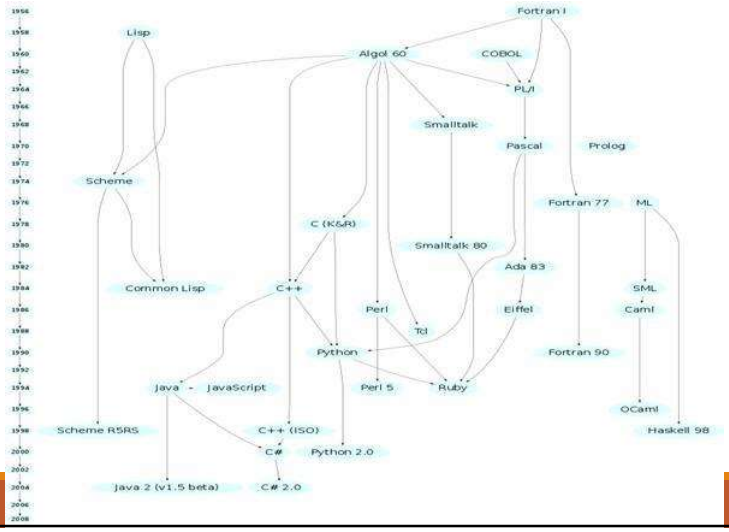
The History of Programming Languages



http://rigaux.org/language-study/diagram.tml

6

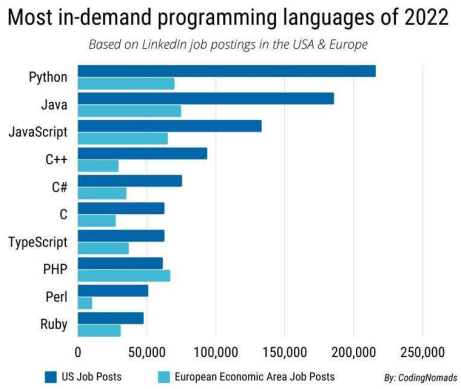
The History of Programming Languages



http://rigaux.org/language-study/diagram.html

7

The Ten Best Programming Languages to learn in 2022



Source: https://www.techrepublic.com/article/the-best-programming-languages-to-learn-in-2022/

8

Popularidade das Linguagens de Programação

Feb 2022	Feb 2021	Change	Programming Language	Ratings	Change
1	3	↑	Python	15.33%	+4.47%
2	1	↓	C	14.08%	-2.26%
3	2	↓	Java	12.13%	+0.84%
4	4		C++	8.01%	+1.13%
5	5		C#	5.37%	+0.93%
6	6		Visual Basic	5.23%	+0.90%
7	7		JavaScript	1.83%	-0.45%
8	8		PHP	1.79%	+0.04%
9	10	↑	Assembly language	1.60%	-0.06%
10	9	↓	SQL	1.55%	-0.18%
11	13	↑	Go	1.23%	-0.05%
12	15	↑	Swift	1.18%	+0.04%
13	11	↓	R	1.11%	-0.45%
14	16	↑	MATLAB	1.03%	-0.03%
15	17	↑	Delphi/Object Pascal	0.90%	-0.12%
16	14	↓	Ruby	0.89%	-0.35%
17	18	↑	Classic Visual Basic	0.83%	-0.18%
18	20	↑	Objective-C	0.81%	-0.08%
19	19		Perl	0.79%	-0.13%
20	12	↓↓	Groovy	0.74%	-0.76%

TIOBE Index for February 2022

Fonte: https://www.tiobe.com/tiobe-index/

Frequency: Monthly.

Methodology: Based on the number of queries in popular search engines such as Google, Bing, Yahoo, Wikipedia, Amazon, YouTube, and Baidu using the +"<language> programming" search term.

9

IEEE Spectrum

Rank	Language	Type	Score
1	Python	web, desktop, embedded	100.0
2	Java	web, mobile, desktop	95.4
3	C	mobile, desktop, embedded	94.7
4	C++	mobile, desktop, embedded	92.4
5	JavaScript	web	88.1
6	C#	web, mobile, desktop, embedded	82.4
7	R	desktop	81.7
8	Go	web, desktop	77.7
9	HTML	web	75.4
10	Swift	mobile, desktop	70.4
11	Arduino	embedded	68.4
12	Matlab	desktop	68.3

Frequency: Annually.

Methodology: Rankings are created by weighting and combining 11 metrics from eight sources: CareerBuilder, GitHub, Google, Hacker News, the IEEE, Reddit, Stack Overflow, and Twitter

Top Programming Languages 2021 - IEEE Spectrum:
https://spectrum.ieee.org/top-programming-languages-2021

Fonte: https://spectrum.ieee.org/top-programming-languages/

10

Topics

- Programming languages
- Formal Languages
- Source code
- Machine Code
- Source / target language
- Compiler
- Translator
- Transpiler

11

Topics

- Programming languages - Formal languages used in the description of abstract mechanisms. They aim to describe and communicate a computational process.
- Formal Languages - set of chains of symbols (words) of a given alphabet.
- Source Code - High level languages (algorithmic, imperative, procedural, etc.)
- Machine Code - Low level language
- Compiler - program, able to read a file (usually text) containing a program written in a given programming language and generate a program in target language equivalent to the initial program.
- Transpiler - (also called source-to-source compiler or a transcompiler) is a program that translates a source code from a programming language to another at the same level of abstraction.

12

Sistema de Computação

- **Hardware**
- **Software**
  - **Software de Sistema**
    - **Sistema Operativo**
    - **Software de Linguagens (Tradutores, compiladores,...)**
    - **Utilitários (editores, …)**
  - **Software de Desenvolvimento**
    - **Ambientes de programação**
    - **SGBD- Sistemas de Gestão de Bases de Dados**
    - **Folhas Cálculo,...**
  - **Aplicações**
    - **"Packages"**
    - **Aplicações por medida**

13

Tradutores

•**Assembler** (o primeiro tradutor) Linguagem de baixo nível (Assembly)
(1 instrução da linguagem -> 1 instrução máquina)

•**Compiladores e Interpretadores** - Linguagens de alto nível
(1 instrução da linguagem -> várias instruções máquina)

A principal diferença entre um **compilador** e um **interpretador**:

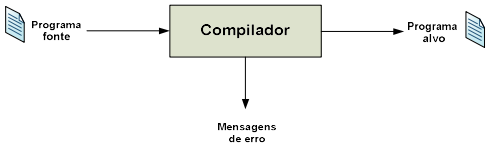
- o **compilador** traduz o programa fonte no programa em linguagem máquina que depois é executado como um todo.

- O **interpretador** traduz e executa instrução a instrução até concluir o programa.

14

Compiladores

Um COMPILADOR é um programa que lê um programa escrito numa linguagem (a linguagem fonte) e o traduz num programa equivalente numa outra linguagem a linguagem objecto), identificando a presença de erros no programa fonte.



Compilação is the process of translating a program written in high-level language to target language code The compiling process includes basic translation mechanisms and error detection. Compiler process goes through lexical, syntax, and semantic analysis at the front end, and code generation and optimization at a back-end. .

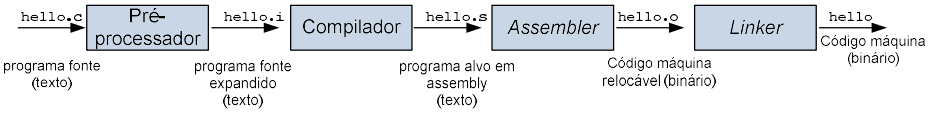
15

Funções do compilador

- **Função principal:** traduzir uma linguagem (fonte) noutra (alvo)
- **Programa fonte:** geralmente uma linguagem de alto nível
- **Programa alvo:** geralmente código alvo
- **Outras características:**
  - reconhecer os programas fonte legais (e ilegais)
  - gerar código alvo correcto
  - gerir a colocação em memória de dados e código
  - gerar o código alvo num formato reconhecido por outros componentes do sistema de desenvolvimento de *software* (*linkers* e *loaders*)

16

Contexto de um compilador



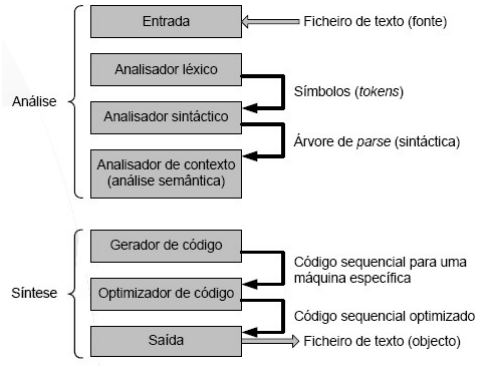
•Um compilador *traduz* um programa de uma linguagem *textual*, facilmente entendida por um ser humano, para *linguagem alvo*, específica para um processador e um sistema operativo.

•Em algumas linguagens de programação, o compilador tem o papel de converter o código fonte num código chamado Bytecode (código em bytes), um estado intermédio entre o código-fonte (escrito numa linguagem de programação específica) e a aplicação final.

•Como exemplo de linguagens que utilizam *bytecode* temos: Java, C# e Lua.

17

Estrutura Geral de um Compilador



18

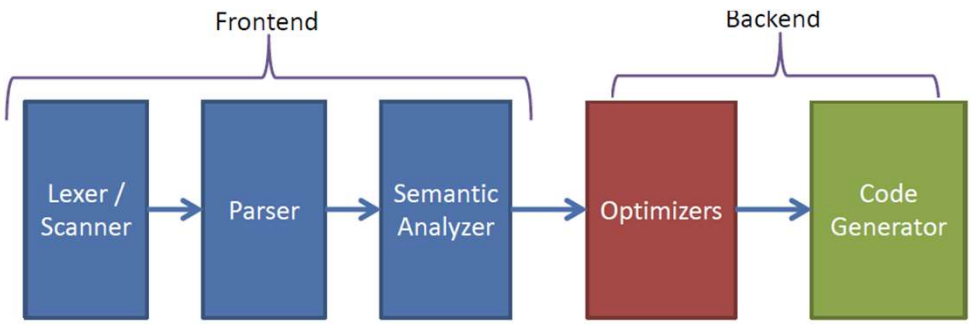
Processo de Compilação

O processo de compilação em duas fases:

- **Análise**
  - Divide o texto fonte em partes
  - Cria representação intermédia
  - Independente da máquina
- **Síntese**
  - Constrói programa-alvo a partir de código intermédio
  - Dependente da máquina

19

Estrutura Geral de um Compilador

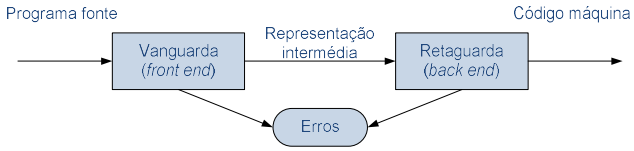


20

Estrutura Geral de um Compilador

(Estrutura funcional)

Divisão do compilador em vanguarda (*front end*) e retaguarda (*back end*)

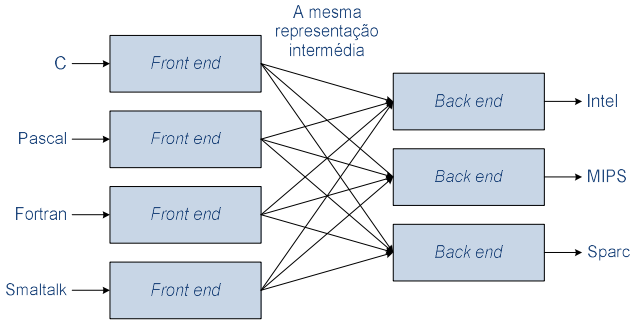


Características

- Existência de uma representação intermédia do programa fonte (máquina abstracta)
- A vanguarda mapeia o programa fonte numa representação intermédia
- A retaguarda produz o código alvo(máquina concreta) a partir da representação intermédia
- Simplifica a produção de compiladores para várias máquinas concretas
- Simplifica a produção de compiladores para várias linguagens fonte
- Duas passagens → código mais eficiente que numa única passagem

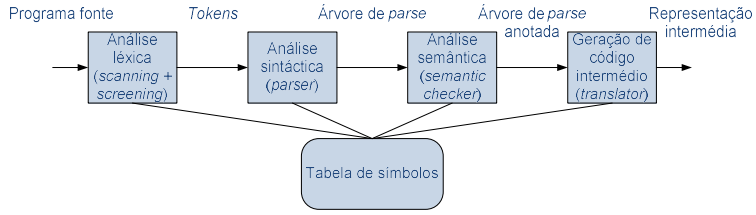
21

Vantagens da representação intermédia



22

Front end

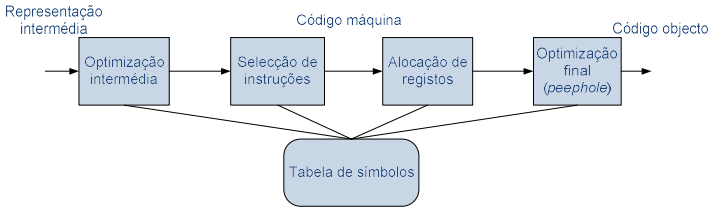


Funções:

- Reconhecer programas válidos
- Produzir mensagens de erro
- Produzir a representação intermédia
- Produzir um mapa de armazenamento preliminar

23

Back end



Funções:

- Traduzir a representação intermédia em código alvo
- Escolher as instruções correspondentes a cada operação definida na representação intermédia
- Decidir que informação manter nos registos do processador
- Assegurar a concordância com os formatos usados por outros componentes do sistema de desenvolvimento de software

24

Exemplo: Compilação de uma instrução de atribuição de uma dada linguagem de acordo com as regras gramaticais:

...

<atribuição> → id := <expressão>

<expressão> → <expressão> * <expressão>

| <expressão> + <expressão>

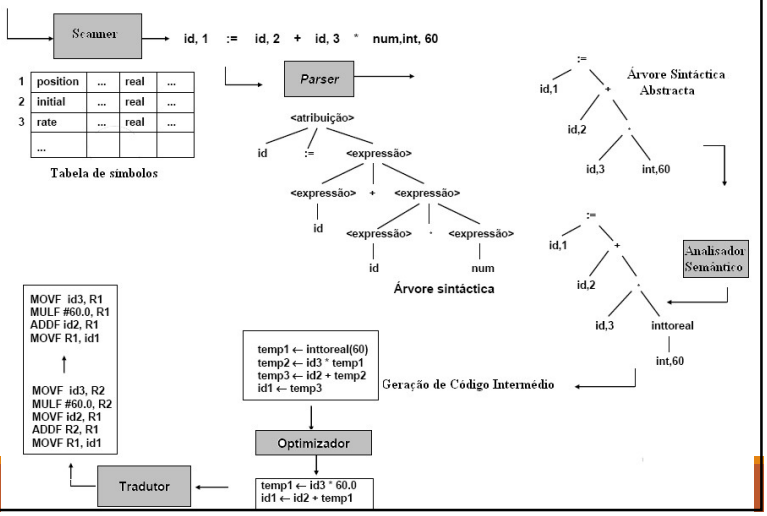
| id

| num

Atribuição a compilar:

...

position := initial + rate * 60



25

Análise léxica

- Agrupa sequências de caracteres em tokens - as unidades básicas da sintaxe.
- Nesta fase é realizada uma Análise linear do código fonte,o ficheiro de entrada é "varrido" carácter a carácter->varrimento.
  - Símbolos especiais (espaço em branco, símbolos de pontuação e new line) são utilizados para estabelecer os limites das palavras.
  - Eliminação de espaços em branco e comentários.
  - Durante a análise léxica, as palavras ou **lexemas** são guardados na tabela de símbolos e classificados de acordo com a linguagem, em palavras reservadas, comandos, variáveis e tipos básicos.
  - Identificação de erros léxicos ("overflow" de campo numérico, uso de símbolos não pertencentes ao alfabeto da linguagem).
  - Agrupamento de caracteres em sequências com um determinado significado – ***tokens***
  - Exemplo: **y = x + 'a';**
    - Identificador y
    - Símbolo de atribuição =
    - Identificador x
    - Sinal +
    - Carácter 'a'
    - Símbolo de terminação ;

Recorre-se de:

- **Expressões Regulares**
- **Autómatos Finitos**

26

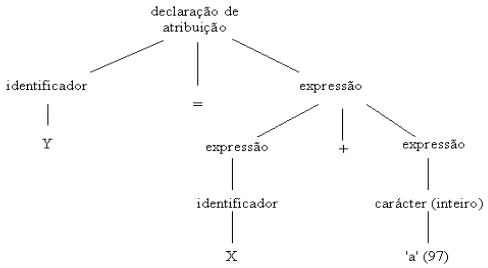
Análise sintáctica

É responsável pela verificação da correcta formação dos comandos da linguagem, de acordo com as regras especificadas pela gramática da linguagem. Agrupamento hierárquico dos *tokens*.

Regras recursivas

Definição recursiva duma expressão

- um identificador é uma expressão
- um número é uma exp
- se e1 e e2 são expres



27

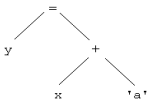
Análise Semântica

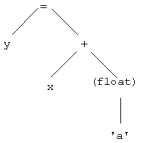
Trata da apreensão do significado associado à concretização das estruturas sintácticas identificadas na fase anterior

A apreensão do significado num compilador implementa-se pelo cálculo de uma série de atributos (valores que caracterizam) associados às classes sintácticas definidas na gramática da linguagem

Exemplos de atributos:

- Tipos dos identificadores (variáveis, procedimentos, classes, …)
- Tipos e valores das constantes (literais)
- endereços de armazenamento na memória (relativos)





Calculados os atributos são efectuadas algumas verificações, principalmente para evitar a existência de inconsistências, tais como:

- Os identificadores foram previamente declarados
- Numa expressão os tipos dos operandos e operadores são compatíveis

28

Geração da representação intermédia

- O programa fonte depois de verificado semanticamente é transformado numa representação intermédia:
  - deve ser fácil de produzir
  - representar bem todas as características da linguagem fonte
  - representar bem as operações disponíveis na máquina alvo
  - deve ser fácil de traduzir para instruções máquina (alvo)
- Algumas formas de representação intermédia:
  - **Gráficas**: Árvores sintácticas abstractas com anotações (os atributos da análise semântica)
  - **Lineares**: Sequência de instruções para uma máquina genérica e abstracta.Existem várias formas:
    - máquinas de 0 ou 1 endereços (máquinas de stack; p-code)
    - máquinas de 2 endereços (próximas de alguns processadores reais)
    - máquinas de 3 endereços (de mais alto nível)
  - **Híbridas**: Grafos de fluxo entre blocos de instruções

29

Optimização do código intermédio

Fase opcional, que pode exigir várias passagens sobre a representação intermédia

Tentativa de melhoramento da representação intermédia por forma a facilitar a produção de melhor código máquina:
- mais rápido
- e/ou mais compacto (que utiliza menos memória)

Algumas optimizações:
- Reconhecer e propagar valores constantes
- Mover cálculos para locais onde o número de execuções é menor
- Reconhecer cálculos redundantes e eliminá-los
- Remover código que é redundante ou inalcançável

As optimizações não podem alterar resultados parciais ou finais do programa fonte

30

Optimização final (peephole)

- Optimização efectuada localmente no código máquina através da análise de sequências curtas de instruções
- Certas sequências podem ser substituídas por outras equivalentes,
  - mas mais eficientes
  - Incremento em vez de soma
  - deslocamento (shift) em vez de multiplicação
  - Eliminação de instruções redundantes, inúteis ou inalcançáveis

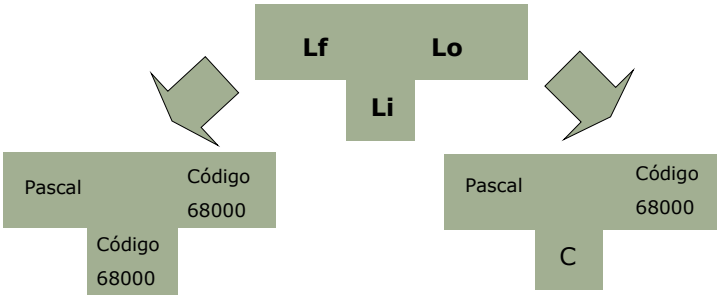
31

Detecção e recuperação de erros

- É possível encontrar erros em praticamente todas as fases do processo de compilação
- A maior parte dos erros é detectada nas fases de análise (léxica, sintáctica e semântica)
- Quando um erro é detectado deve ser emitida uma mensagem esclarecedora e dever-se-á prosseguir na compilação
- Sempre que um erro é detectado dever-se-á recuperar desse erro (descartando parte do texto fonte ou assumindo uma correcção do erro) por forma a que a tarefa de análise em curso possa prosseguir.

32

Representação por esquema T

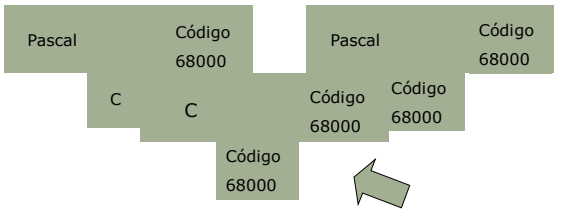


Lf – Linguagem fonte
Lo – Linguagem objecto
Li – Linguagem de implementação

33

Representação por esquema T

Esta descrição é também utilizada para simbolizar processos de tradução complexos que envolvem vários estádios, linguagens ou máquinas

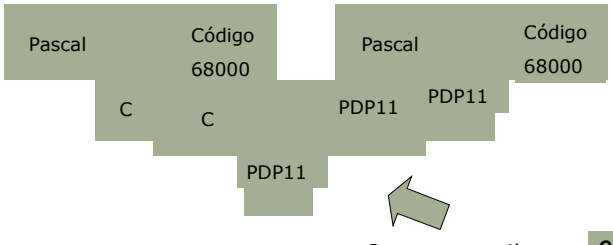


$Pascal_{C}\ 68000 + C_{68000}\ 68000 = Pascal_{68000}\ 68000$

Admitamos que é escrito um compilador para ling. Pascal através da linguagem de implementação C para gerar código 68000, ou seja $Pascal_{C}\ 68000$. Se existir um compilador para linguagem C que corra numa máquina 68000 e que gere código para essa mesma máquina, este é caracterizado por $C_{68000}\ 68000$.
Se $Pascal_{C}\ 68000$ é executado sobre $C_{68000}\ 68000$, obtém-se um compilador $Pascal_{68000}\ 68000$.

34

Cross-Compiler



Cross-compiler

Cross-compiler: compilador que corre sobre uma máquina e gera código objecto para outra máquina.

Um compilador pode ser escrito em qualquer linguagem, mas existem ferramentas próprias para o seu desenvolvimento (ex: FLEX, LEX, YACC, BISON) e linguagens capazes de expressar de forma clara uma determinada estrutura (ex: 'C', Pascal, Algol 68, etc).

35

Bootstrapping

A técnica de *bootstrapping* consiste na construção de um compilador, usando uma linguagem de implementação L, que compile a própria linguagem L.
É usada para:

- obter um novo compilador de uma linguagem L2, usando um compilador já existente de uma linguagem L1, em que L1 é um subconjunto de L2
- obter um novo compilador mais optimizado da mesma linguagem

36