Introdução à Software Básico: Compilação vs. Interpretação

Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade de Brasília

Sumário

Compilação vs. Interpretação

1. Linguagem de Programação.
2. Evolução das Linguagens de Programação.
3. Processadores de Linguagens.
4. Definição de Tradução.
5. Tradução vs Interpretação.
6. Compilação vs. Interpretação.

Linguagem de Programação

Linguagem

Sistema de comunicação humano que utiliza diferentes sinais, tais como sons, gestos ou símbolos escritos.

Linguagem de Programação

Linguagem artificial desenhada para comunicar instruções a um sistema computacional.

Função da Linguagem de Programação

- A linguagem de Programação é utilizada para criar um conjunto de instruções e diretivas (programa) para expressar um determinado algoritmo a ser realizado pelo sistema computacional.
- A linguagem de Programação é o meio de ligação entre o pensamento humano e o processamento de um computador/máquina. À diferença do ser humano que possui um alto nível de inferência sendo capaz de compreender e/ou realizar tarefas a partir de uma comunicação imprecisa e pouco estruturada, o computador/máquina requer uma alta formalidade na sintaxe das instruções as quais executa de forma precisa.

Desenvolvimento de um Programa

- Elaborar o algoritmo de solução para um determinado problema.
- Formalizar o algoritmo na linguagem de programação especifica.

- O processo de formalização do algoritmo se torna mais fácil se a linguagem de programação utilizada é próxima a linguagem do ser humano.
- Por outro lado, dependendo do problema a ser resolvido uma determinada linguagem de programação ou tipo de linguagem de programação pode facilitar também o trabalho.

Tipos de Linguagem de Programação

Como visto anteriormente, as linguagens de programação podem ser agrupadas em dois grandes grupos:

- Linguagens de baixo nível e
- Linagens de alto nível.

Linguagens de baixo nível vs. Linguagens de alto nível

Linguagens de baixo nível

Basicamente, existem 2 formas de linguagem de baixo nível:

- Linguagem de Máquina.
- Linguagem *Assembly*.

Linguagens de baixo nível

Linguagem de Máquina

- Cada CPU entende somente programas escritos no seu próprio linguagem de máquina.
- É proporcinado pelo fabricante da CPU.
- É necessário traduzir toda outra linguagem de programação à linguagem de máquina.
- As instruções são escritas como números. A CPU interpreta os números na sua representação binária. (e.g. 10111010).

Linguagens de baixo nível

Linguagem de Máquina

Desvantagens:

- Muito difícil de escrever. A linguagem binária não é natural para o ser humano.
- O programador precisa saber os endereços de armazenamento de dados e instruções.

Linguagens de baixo nível

Linguagem *Assembly*

- Também é proporcionado pelo fabricante da CPU.
- Cada instrução é representada por um código mnemônica (conjunto de letras).
- O código precisa ser montado à código máquina.
- Desvantagens:
  - O programa é ainda dependente da arquitetura da CPU/computador.

Linguagens de baixo nível

Linguagem de alto nível

- Programação é muito mais fácil. As instruções são mais próximas à linguagem do ser humano.
- Permite diferentes tipos de abstração. Gerando diferentes paradigmas de programação: estruturado, orientado à objetos, etc.

Baixo nível vs. Alto nível

Alto nível

- Permite ao programador concentrar-se mais na solução ao invés de como o hardware vai tratar o problema.
- Facilidade de codificação e manutenção do código.
- Permite fácil portabilidade.
- Porém, podem ser mais lentas e necessitam de tradutores eficientes
- Dificuldade de implementação dos tradutores e otimizadores de código

Baixo nível

- É indicada para funções que precisam implementar instruções de máquina específicas, já que muitas funções de baixo nível não são suportadas por linguagens de alto nível.
- Também é indicada para funções que requerem grande eficiência e um tamanho reduzido do programa (e.g. Drivers)
- Permite compreensão e controle total de como a máquina vai executar o algoritmo

Evolução

As linguagens de programação são classificadas historicamente em cinco gerações. Existem algumas divergências quanto as 4ta e 5ta gerações, pois ainda são muito recentes.

- Linguagens de Primeira Geração (1GL): $1940 - 50$.
  - Também chamadas Linguagens de Máquina
  - Tais linguagens permitem a comunicação direta com o computador em termos de bits, registradores e operações de máquina bastante primitivas.
  - Não existiam compiladores ou tradutores, e os valores eram fornecidos via chaves no painel
  - Os seus programas são seqüências de 0's e 1's: muito complexos, cansativos e sujeitos a muitos erros.
- É possivél afirmar que a primeira geração de linguagens de programação começou muito antes de 1940, com Ada Lovelace e Charles Babbage no primeiro programa de computador o qual realizava a sequência de Bernoulli para a maquina analítica.

Evolução

- Linguagens de Segunda geração (2GL): $1950 - 58$
  - Também chamadas Linguagens Simbólicas ou *Assembly*.
  - Projetadas para minimizar as dificuldades da programação binária.
  - Códigos de operação e endereços binários foram substituídos por mnemônicos. O seu processamento requer a tradução para a linguagem de máquina, antes de sua execução.
    - eg. ADD = B805 = 1011100000000101
    - Alguns autores classificam a primeira e segunda geração como linguagem de máquina.
  - Programas de tradução dos mnemônicos para linguagem de máquina são chamados de Assemblers ou Montadores

Evolução

- Linguagens de Terceira Geração (3GL): $1958 - 85$
  - Também chamadas de linguagens de programação modernas (estruturadas/orientadas à objetos),
  - Principais características:
    - Tipagem de dados
    - Grande capacidade procedural e estrutural de seus dados
    - Possibilidade de criar sistemas distribuídos (programas mais complexos)
  - Podem ser divididas em duas grandes categorias:
    - linguagens de propósito geral, e
    - linguagens especializadas.
  - As linguagens de propósito gerais foram desenvolvidas baseadas principalmente na linguagem Algol e servem para uma infinidade de aplicações envolvendo desde a área científica, até a área comercial.
  - As linguagens C, Fortran, Pascal, PL/1 e Modula-2 são as principais linguagens desta categoria, sendo que as duas primeiras continuam bastante usadas atualmente (Pascal ainda é usado didaticamente).

Evolução

- Linguagens de Terceira Geração (3GL)
  - Centenas de linguagens especializadas estão em uso atualmente. Caracterizadas pela forma sintática não usual com que foram desenvolvidas para uma aplicação distinta.
  - Exemplos:
    - Lisp: manipula símbolos e listas
    - Prolog: trata e representa conhecimento
    - Smalltalk: representa dados em forma de objetos, sendo a primeira a ser puramente orientada a objetos
    - APL: manipula vetores.

Evolução

- Linguagens de Quinta Geração (4GL): A partir aprox. de 1985
  - Consistem de comandos similares aos da linguagem natural.
  - São normalmente empregadas em banco de dados e scripts.
  - Os principais objetivos dessas linguagens são:
    - facilitar a programação de computadores de tal forma que os usuários finais possam resolver seus problemas;
    - acelerar o processo de desenvolvimento de aplicações;
    - facilitar e reduzir o custo de manutenção de aplicações;
    - minimizar problemas de depuração; e
    - gerar código sem erros a partir de requisitos de expressões de alto nível.
  - Nesta categoria, temos:
    - SQL, Access, etc.
    - QuatroPro, LOTUS 1-2-3, Excel, etc.

Evolução

- Linguagens de Quinta Geração (5GL): A partir aprox. de 1989
  - Nesta categoria estão as linguagens que utilizam recursos visuais para o desenvolvimento de programas.
  - Um exemplo de linguagem de quinta geração é o VB,Delphi,etc.
  - Outra definição de gerações, inclui as linguagens visuais dentro da 4ta geração, a 5ta geração é reservada para linguagens de programação desenvolvidas mediante inteligência artificial.
    - Algumas características destas linguagens são a utilização de IDEs (Integrated Development Environment) e RADs (rapid application development).

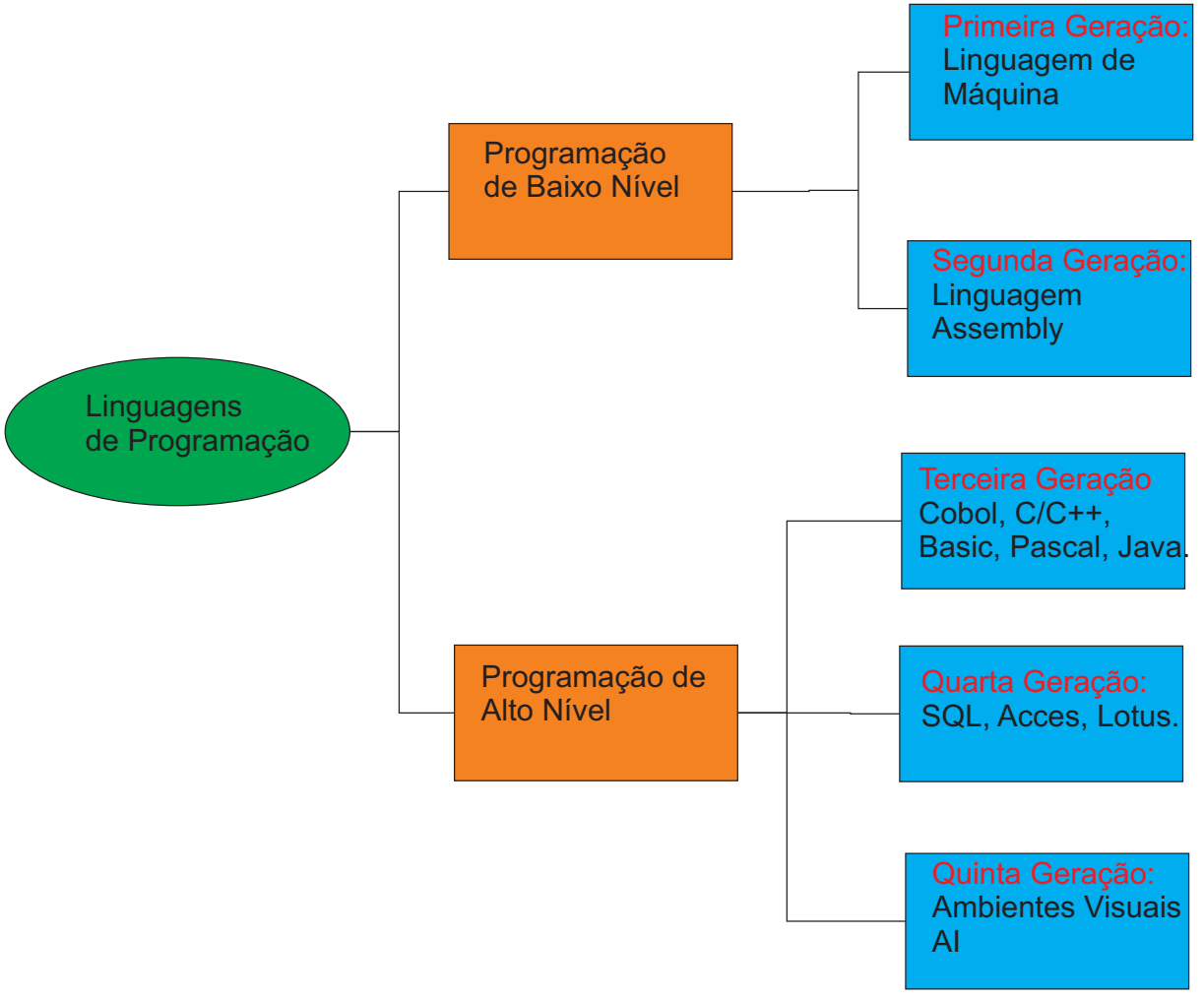

Figura: Gerações das linguagens de Programação.

Processadores de Linguagens

- As linguagens de alto nível são as linguagens que possuem uma certa independência da máquina
- Elas não são desenvolvidas utilizando instruções específicas do processador (linguagem de máquina), mas um conjunto de comandos que são transformados em linguagens de baixo nível.
- As linguagens de programação são implementadas por compilação de programas em linguagem de baixo nível, por interpretação das mesmas, ou por alguma combinação de compilação e interpretação.

Processadores de Linguagens

- Qualquer sistema para processamento de programas – executando-os ou preparando-os para a execução – é chamado **processador de linguagem**.
- Processadores de linguagem incluem **compiladores, interpretadores**, e ferramentas auxiliares como editores dirigidos à sintaxe.

Tradução

- Tradução significa mudar uma linguagem para outra **sem mudar seu significado**. Exemplo: traduzir de português para inglês.
- Em computação, um compilador é um tipo de tradutor.

Tradução

- Programas são normalmente traduzidos para versões equivalentes em linguagem de máquina, antes de serem executados.
- Esta tradução é feita em vários passos.
- Exemplo:
  - Subprogramas $\rightarrow$ tradução $\rightarrow$ código Assembly,
  - Assembly $\rightarrow$ tradução $\rightarrow$ linguagem de máquina;
  - Ling. máquina $\rightarrow$ ligação $\rightarrow$ módulo de carga (ling.máquina);
  - Módulo de carga $\rightarrow$ carga $\rightarrow$ módulo de carga em memória principal.
- Os tradutores usados em cada um destes passos tem nomes especiais: compilador, montador, ligador (linker) e carregador (loader), respectivamente.

Tradução

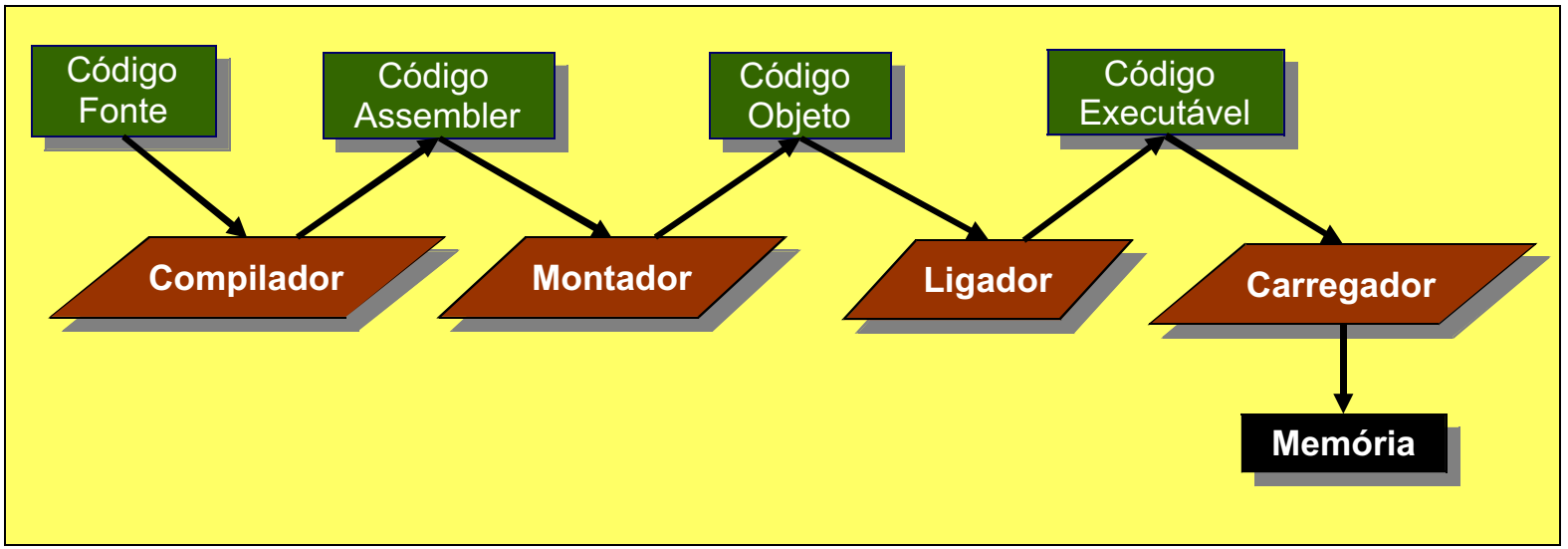

Figura: Hierarquia de Tradução.

Nota

Em alguns casos, a máquina onde a tradução é feita (a máquina hospedeira) é diferente daquela onde o código gerado é executado (a máquina objetivo). Neste caso o processo é chamado tradução cruzada. Tradutores cruzados são a única opção de tradução quando a máquina objetivo é muito pequena para conter o tradutor

Classificação dos Tradutores

Montadores (*assemblers*)

- A linguagem fonte é essencialmente uma representação simbólica para a linguagem de máquina.
- A linguagem fonte, nesses casos, é denominada de linguagem de montagem.
- O processo de montagem recebe como entrada um arquivo texto com o código fonte do programa em *assembly* e gera como saída um arquivo binário contendo o código de máquina e outras informações relevantes para a execução do código gerado.

Macro-Assemblers

- Em geral, montadores oferecem facilidades além da simples tradução de código *assembly* para código de máquina.
- Além das instruções do processador, um programa fonte para o montador pode conter **diretivas** (ou pseudo-instruções) definidas para o montador (e não para o processador), assim como **macro-instruções**, uma seqüência de instruções que será inserida no código ao ser referenciada pelo nome.
- Um montador que suporte a definição e utilização de macro-instruções é denominado **macro-assembler** (ou macro-montador).
- Um **montador multiplataforma** (cross-assembler ou tradutor-cruzado) é um montador que permite gerar código para um processador-alvo diferente daquele no qual o montador está sendo executado.

Pré-compiladores, Pré-processadores ou Filtros

- Mapeiam instruções escritas em linguagem de alto nível estendida para instruções da linguagem de programação original, ou seja, efetuam tradução entre duas linguagens de alto nível.
- gcc –help
  - -E Preprocess only; do not compile, assemble or link
  - -S Compile only; do not assemble or link
  - -c Compile and assemble, but do not link

```
#define A 10
#define B 20
int main(void){
  int i,j;
  for(j=0,i=A;i<B;i++)
   j+=i
}
```

```
$ gcc -E PreProc.c
int main(void){
 int i,j;
 for(j=0,i=10;i<20;i++)
  j+=i
```

Figura: Exemplo de pré-compilação

Classificação dos Tradutores

Interpretador

- As ações indicadas pelos comandos da linguagem são diretamente executadas. Sem gerar arquivo objeto.
- Um programa totalmente interpretado a principio executaria as instruções diretamente. Essas interpretadores **não fazem nenhum tipo de tradução**. Logo, a principio os Interpretadores não podem ser visto como um tipo de tradutor.
- Porém, normalmente um interpretador traduz a uma linguagem intermediaria onde é interpretado por um ambiente de software, por exemplo: máquina virtual Java, antes da execução.
  - A interpretação consiste em executar, para cada ação possível, um subprograma (escrito na linguagem de máquina do computador hospedeiro).
  - A interpretação de um programa é feita pela chamada dos subprogramas, em uma seqüência apropriada.
- Um interpretador é um programa que executa repetidamente a seguinte seqüência:
  - Obter o próximo comando do programa.
  - Determinar que ações devem ser executadas.
  - Executar estas ações.

Interpretação

- Os passos utilizados na interpretação são bastante semelhantes àqueles executadas por um processador:
  1. Obter a próxima instrução (aquela cujo endereço é especificado no indicador de instruções da máquina).
  2. Deslocar o indicador de instruções (obtendo o endereço da próxima instrução a ser executada).
  3. Decodificar a instrução. A instrução é codificada para uma representação intermediária eficiente.
  4. Executar a instrução. Para isso é necessário ter subprogramas pre-compilados que fazem parte do interpretador ou interpretar as instruções dentro de um ambiente independente do OS (ex: máquina virtual).

Tradução vs. Interpretação

- A interpretação pura e a tradução pura são dois extremos.
- Na prática, muitas linguagens são implementadas por uma combinação destas técnicas. De fato a maioria das linguagem interpretadas possuem um processo de tradução.

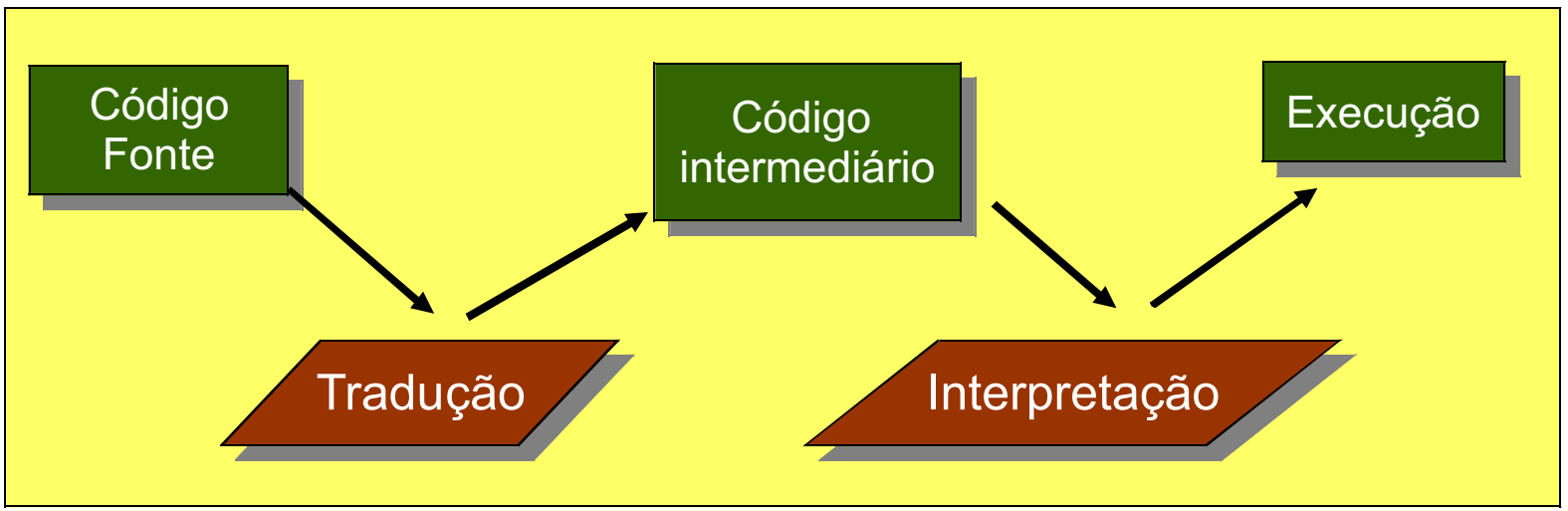

Figura: Interpretadores com Tradução

Classificação dos Tradutores

Compiladores

- Programa utilizado para executar a tradução de uma linguagem para outra. Logo, montadores é um tipo específico de compilador.
- Normalmente quando se fala de compiladores espera-se que a **linguagem alvo** é de mais baixo nível semântico que a **linguagem fonte**.
- Na disciplina vamos assumir que os compiladores usados sempre tem como objetivo traduzir de uma linguagem próxima a linguagem humana (linguagem de alto nível, como o C e Pascal) e chegar ao ponto de uma linguagem que o computador pode entender (Linguagem de baixo nível).

IMPORTANTE!

Um compilador pode a partir de uma linguagem de alto nível gerar linguagem numérica de máquina diretamente ou gerar linguagem simbólica (assembly) e depois chamar um montador.

Compilador vs Interpretador

Compilador

- Os compiladores podem tomar certas decisões a nível de compilação e não a nível de execução:
  - Verificação de sintaxe.
  - Alocação estática.
  - Ligação estática.
  - Otimização de código.
- Os programas compilados normalmente possuem um melhor desempenho.

Interpretador

- Os programas interpretados facilitam a interação entre teste/debug.
- A partir do momento em que um código intermediário é produzido, ele independe de máquina, pois para cada máquina já existe um interpretador específico. Logo, os programas interpretados facilitam a portabilidade.

Próxima Aula

Estrutura e Gramática dos Tradutores