FACULDADE DOM BOSCO de Porto Alegre

Paradigmas de Linguagem de Programação

Prof. Filipo Mór

2016/II - www.filipomor.com

Aula 1 - Introdução

Apresentação do Professor

- B.Sc. SI – FDBPOA – 2012
- M.Sc. em CC – PUCRS/DALHOUSIE – 2015
- Atuação na área de TI desde 1995.
- Passagem por:

Apresentação da Turma

- Vocês!
  - Nome.
  - Experiência profissional.
  - Expectativa da disciplina.

Roteiro

- Apresentação da disciplina.
- Introdução.
- Revisão de conceitos importantes.

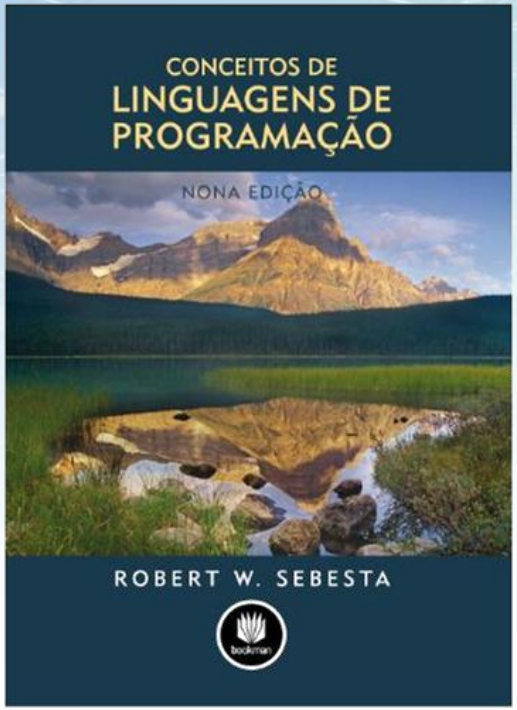
- *Baseado no capítulo 1 do livro "Conceitos de Linguagens de Programação", Robert W. Sebesta.*

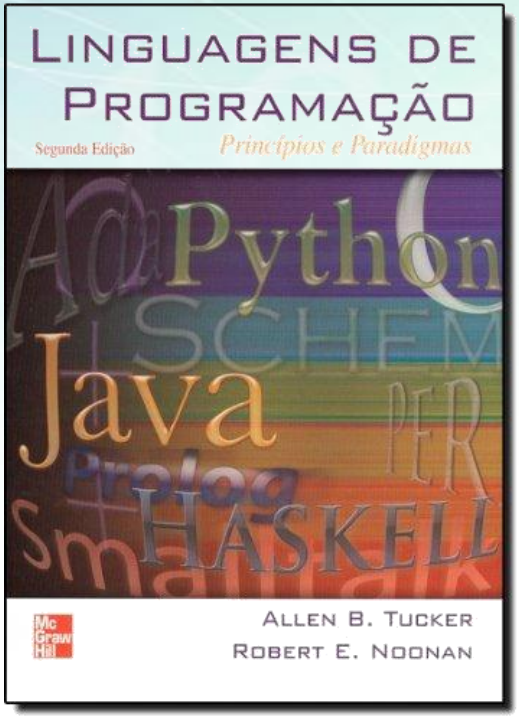
Apresentação da Disciplina

- www.filipomor.com / teaching/ Paradigmas 2016
- Quartas-feiras a partir das 19h10
- Chamadas no início e no final da aula.
- Entregas via moodle (link será disponibilizado no site).


CAUTION
THIS IS SPARTA

Livros Texto





Conceitos Preliminares

- Motivos para estudar os Conceitos de Linguagens de Programação.
- Domínios de Programação.
- Critérios de Avaliação da Linguagem.
- Categorias de Linguagem.
- Trade-offs no Projeto da Linguagem.
- Métodos de Implementação.
- Ambientes de Programação.

Motivos para estudar os Conceitos de Linguagens de Programação

FACULDADE DOM BOSCO de Porto Alegre

- Aumentar a capacidade de expressão de idéias.
- Maior conhecimento para a escolha de linguagens apropriadas.
- Capacidade aumentada para aprender novas linguagens.
- Entender melhor a importância da implementação.
- Aumento da capacidade de projetar novas linguagens.
- Avanço global da computação.

Domínios de Programação

- Aplicações científicas.
- Aplicações comerciais.
- Inteligência artificial.
- Programação de sistemas.
- Linguagens de *scripting*.
- Linguagens de Propósitos Especiais.
- Programação web e plataformas especiais.

Critérios de Avaliação de Linguagem

- Legibilidade.
  - legibilidade x writability x confiabilidade.
  - Facilidade com que os programas podem ser lidos e entendidos.
  - Facilidade com que uma linguagem pode ser utilizada para a escrita de programas.
  - Aderência a especificações.
- Simplicidade Global.
  - `count = count + 1`
  - `count += 1`
  - `count++`
  - `++count`

Critérios de Avaliação de Linguagem
Legibilidade

- Simplicidade Global.
  - `count = count + 1`
  - `count += 1`
  - `count++`
  - `++count`

Critérios de Avaliação de Linguagem
Legibilidade

- Ortogonalidade.
  - Conjunto pequenos de construções primitivas que pode ser combinado em um número pequeno de maneiras.
  - Pouca multiplicidade de caracteristicas (jeitos de fazer a mesma coisa).
  - Minima sobrecarga de operações.
- Instruções de controle
  - Presença de estruturas conhecidas de controle.
  - Devem preceder seus alvos, exceto em laços.
  - Alvos sempre próximos.
  - Número limitado.

Critérios de Avaliação de Linguagem
Legibilidade

- Tipos de Dados e Estruturas
  - Presença de ferramentas adequadas para a definição das estruturas necessárias.
- Considerações sobre a Sintaxe.
  - Formas identificadoras.
  - Nomes muito pequenos x muito grandes.
  - Palavras especiais/reservadas.
  - Forma e significado.

Critérios de Avaliação de Linguagem
Writability

- Simplicidade e Ortogonalidade
  - Poucas construções, número pequeno de primitivas e e uma pequena quantidade de regras sobre como combiná-los.
- Suporte a abstração.
  - Habilidade de definir e utilizar estruturas complexas em operações de forma que os detalhes sejam ignorados.
- Expressividade
  - Um conjunto de formas convenientes de especificar operações.
  - Exemplo: declaração ‘for’ existe em quase todas as linguagens.

Critérios de Avaliação de Linguagem
Confiabilidade

- Verificação de Tipos
  - Erros de tipos de dados. (‘somar’ strings?).
- Manipulação de Exceções
  - Interceptar erros de execução e tratá-los de forma adequada.
- Aliasing
  - Presença de dois ou mais métodos se referenciado a mesma posição de memória.
- Legibilidade e Capacidade de Escrita
  - Suporte a formas ‘naturais’ de expressão de um algoritmo.

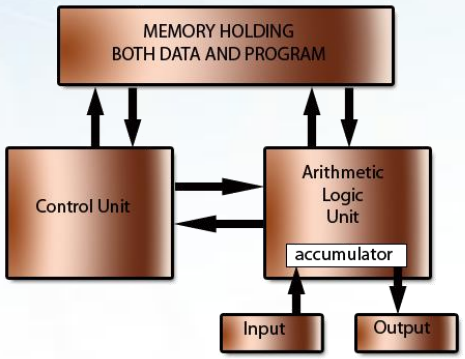
Critérios de Avaliação de Linguagem
Custo

- Treinamento na linguagem.
- Custo de se escrever os programas na linguagem.
  - Linguagens de alto nível foram motivadas por isso.
- Custo da compilação.
- Custo da execução.
- Custo da implantação.
- Custo de manutenção.
- Custo da falta de legibilidade.

Influências no projeto da linguagem

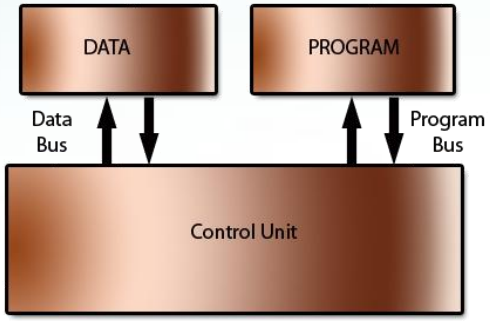
- Arquitetura-Alvo
  - Arquitetura que prevalece: Von Neuman
  - Outras arquiteturas:
    - Hardvard.
    - Aceleradores

The Von Neumann or Stored Program architecture

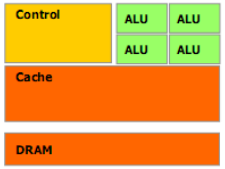


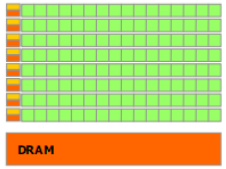
(c) www.teach-ict.com

The Harvard architecture



(c) www.teach-ict.com





Influências no projeto da linguagem

- Metodologias de Programação
  - Novas metodologias levam a novos paradigmas, que, por sua vez, levam a novas linguagens.

Influências da arquitetura-alvo.

- Paradigma dominante
- Imperativo-procedural, em função da arquitetura Von Neumann.
- Dados e instruções armazenados na memória.
- Memória é separada da CPU.
- Dados e instruções acessíveis através de pipeline.
- Conceitos básicos para o paradigma imperativo:
- Variáveis modelam células de memória.
- Declarações de atribuição modelam o uso do pipeline.
- Iterações são eficientes.

Influências mas metodologias de programação

- Décadas de 50 e 60
  - Preocupação com a eficiencia da máquina.
- Final da década de 60
  - Eficiência das pessoas passa a ser importante.
  - Legibilidade, melhores estruturas de controle.
  - Programação estruturada
  - Projeto top-down e refinamentos passo-a-passo.
- Década de 70
  - Orientaçao a processos orientados a dados
  - Abstração de dados.
- Década de 80:
  - Abstração de dados + herança + polimorfismo.

Categorias de Linguagens

- Imperativas
  - Centradas no uso de variáveis, declarações de atribuição e iterações. Ex.: Pascal, C
- Funcionais
  - Computação baseada na aplicação de parâmetros a funções. Ex.; LISP, Scheme.
- Lógicas
  - Baseada em regras (que são especificadas sem uma ordem particular). Ex.: Prolog, .QL
- Orientadas a Objeto
  - Abstração de dados, herança, late binding. Ex.: C++, C#

Categorias de Linguagens

- Marcação
  - Não são linguagens de programação propriamente. Utilizadas para a especificação de layouts. Ex.: XHTML, XML.

Trade-Offs no Projeto de Linguagens

- Legibilidade x Custo de Execução
  - Critérios conflitantes.
  - Ex.: Código gerenciado (Java, C#).
- Legibilidade x Writability
  - Critérios conflitantes.
  - Ex.: APL fornece vários operadores poderosos (e grande número de novos símbolos), permitindo computações complexas com códigos conscisos, mas difíceis de serem entendidos.

```
      ∇DET[⎕]∇
     ∇ Z←DET A;B;P;I
[1]    I←⎕IO
[2]    Z←1
[3]   L:P←(|A[;I])⍳⌈/|A[;I]
[4]    →(P=I)/LL
[5]    A[I,P;]←A[P,I;]
[6]    Z←-Z
[7]   LL:Z←Z×B←A[I;I]
[8]    →(0 1 ∨.=Z,1↑⍴A)/0
[9]    A←1 1 ↓A-(A[;I]÷B)∘.×A[I;]
[10]   →L
[11]  ⍝EVALUATES A DETERMINANT
      ∇
```

http://www.computerhistory.org/atchm/the-apl-programming-language-source-code/

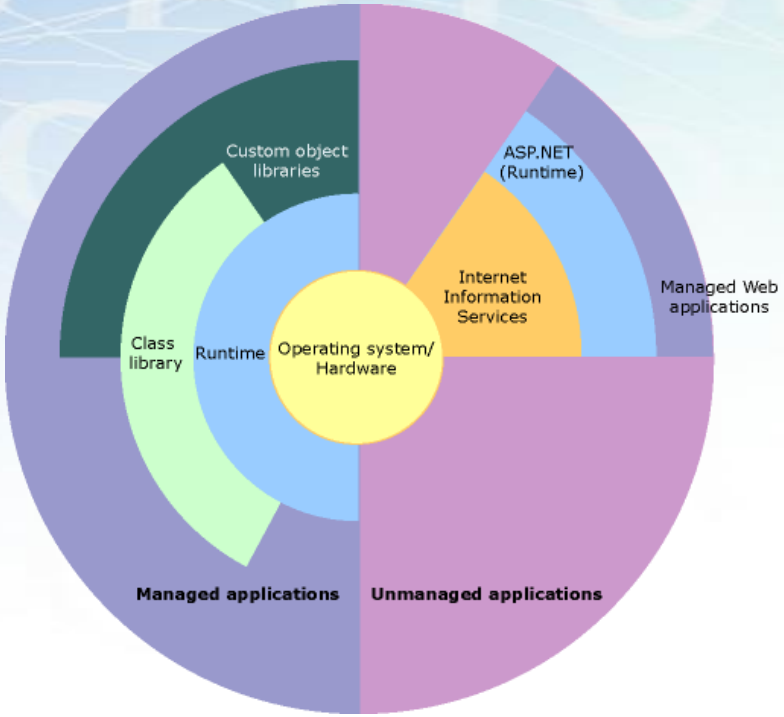
Trade-Offs no Projeto de Linguagens

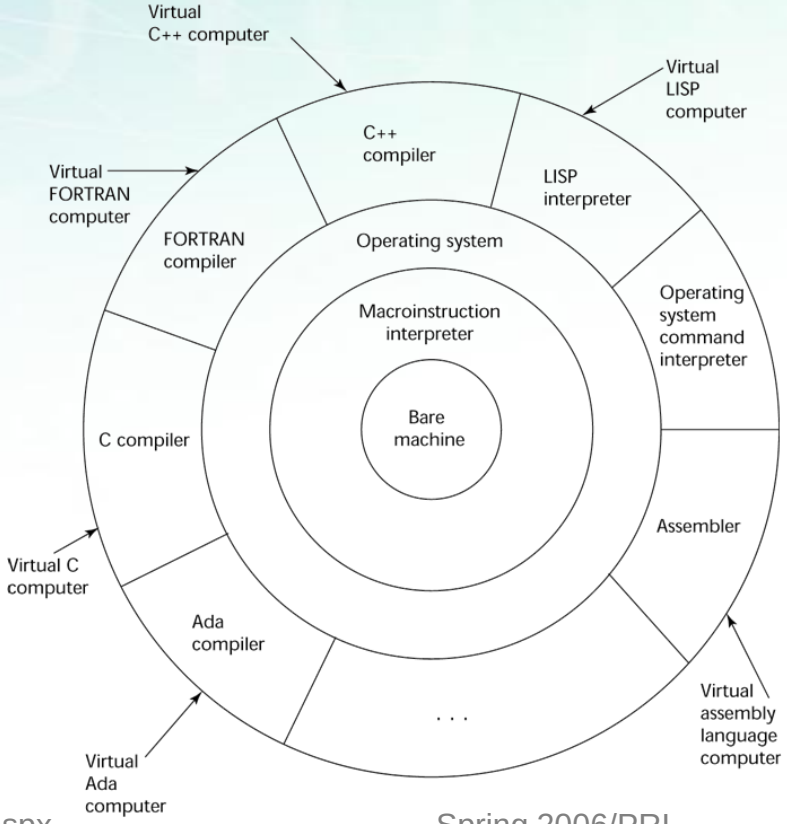
- Writability(flexibilidade) x Confiabilidade
  - Critérios conflitantes.
  - Ex.: ponteiros no C, são poderos e flexíveis, mas normalmente utilizados de forma não confiável. (memory leak, vigrant).

Métodos de Implementação

- Compilação
  - Programas são convertidos para linguagem de máquina.
- Interpretação (‘pura’).
  - Programas são interpretados, instrução por instrução, por outro programa chamado “*in*
- *Implementações Híbridas*
  - Um “meio termo” entre um compilador e um intepretador.

Métodos de Implementação

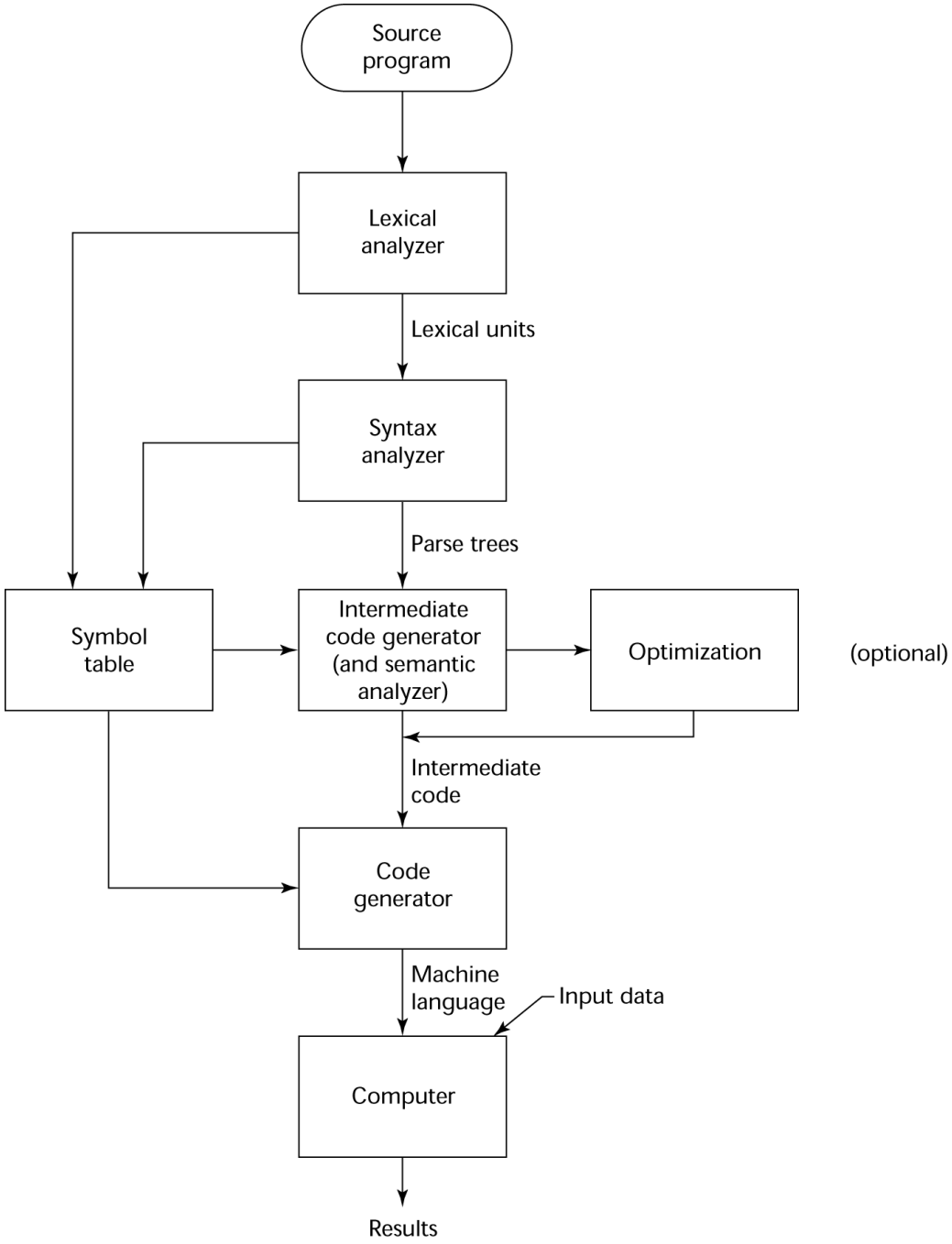




https://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.110).aspx

Compilação

- Transforma código de alto nível em código inteligível pela máquina.
- Processo de tradução lento; execução rápida.
  - **Análise Léxica**: converte os caracters do código fonte em objetos ou unidades léxicas.
  - **Análise Sintática**: transforma unidades léxicas em árvores sintáticas que representam a estrutura do programa.
  - **Análise Semântica**: gera um código intermediário.
  - **Geração de Código**: gera o código de máquina.


Source program
Lexical analyzer
Lexical units
Syntax analyzer
Parse trees
Symbol table
Intermediate code generator (and semantic analyzer)
Optimization
(optional)
Intermediate code
Code generator
Machine language
Input data
Computer
Results

Terminologia Adicional para Compilação

- Load Module (image do executável)
  - Código do usuário e do sistema juntos.
- Linking e Loading
  - Processo de coletar programas do sistema e ligá-los ao programa do usuário.
  - Ligação dinâmica ou estática.

Execução de Código de Máquina

- Ciclo *fetch-execute* (busca-executa)

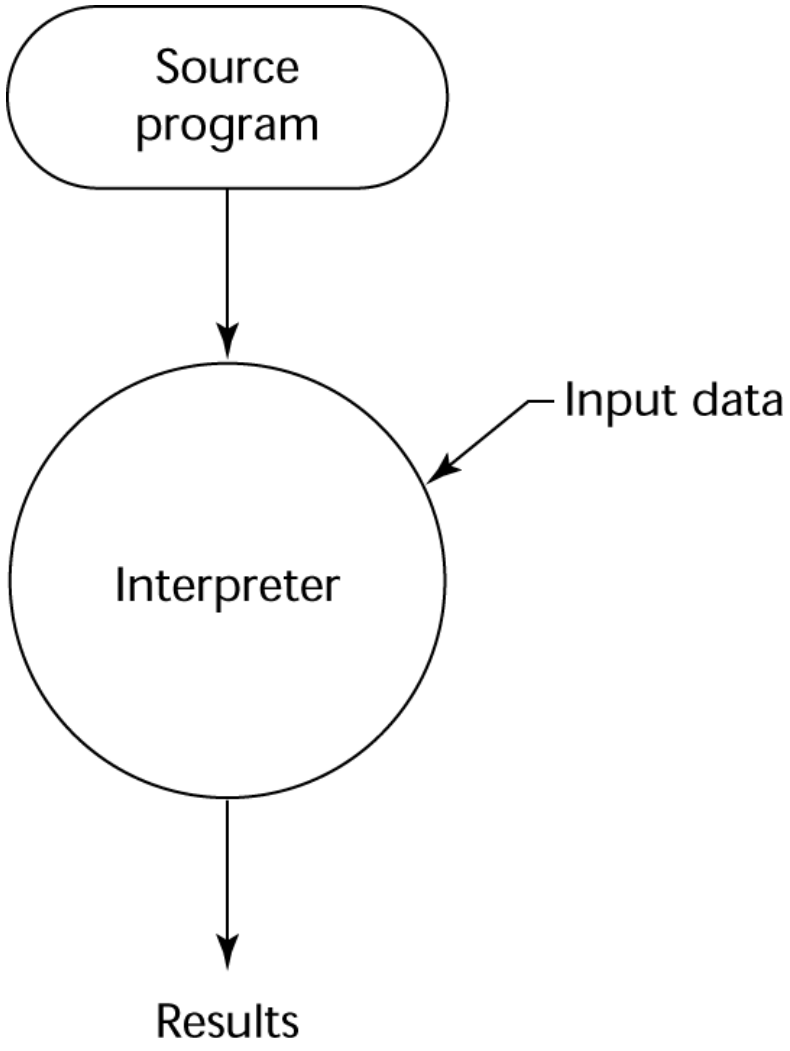
```
inicializa o contador do programa
repete enquanto não atinge condição de parada/para sempre
    Busca a instrução apontada pelo contador
    Incrementa o contador
    Decodifica a instrução
    Executa a instrução
fim repita
```

Gargalo da Arquitetura de Von Neumann

- Velocidade da conexão memória/processador define a velocidade do computador.
- Normalmente instruções podem ser executadas em velocidade maior do que a operação de busca dessas mesmas instruções na memória. A velocidade de transferência se torna o gargalo.
- Conhecido como ***Gargalo de Von Neumann***. É o principal fator limitador do desempenho do computador.

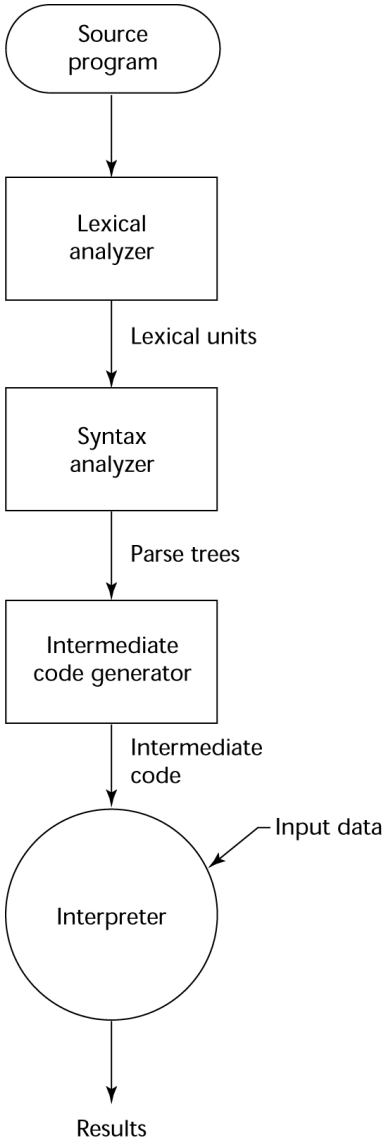
Interpretador ‘puro’

- Sem tradução.
- Mais fácil de se implementar programas. Erros são identificados e tratados imediatamente.
- Execução MUITO mais lenta do que em programas compilados.
- Normalmente exigem mais espaço.
- Raros em linguagens de alto nível (hoje em dia: será?)
- Volta as origens, com linguagens como Javascript ou Python.


Source
program
Input data
Interpreter
Results

Sistemas ‘híbridos’

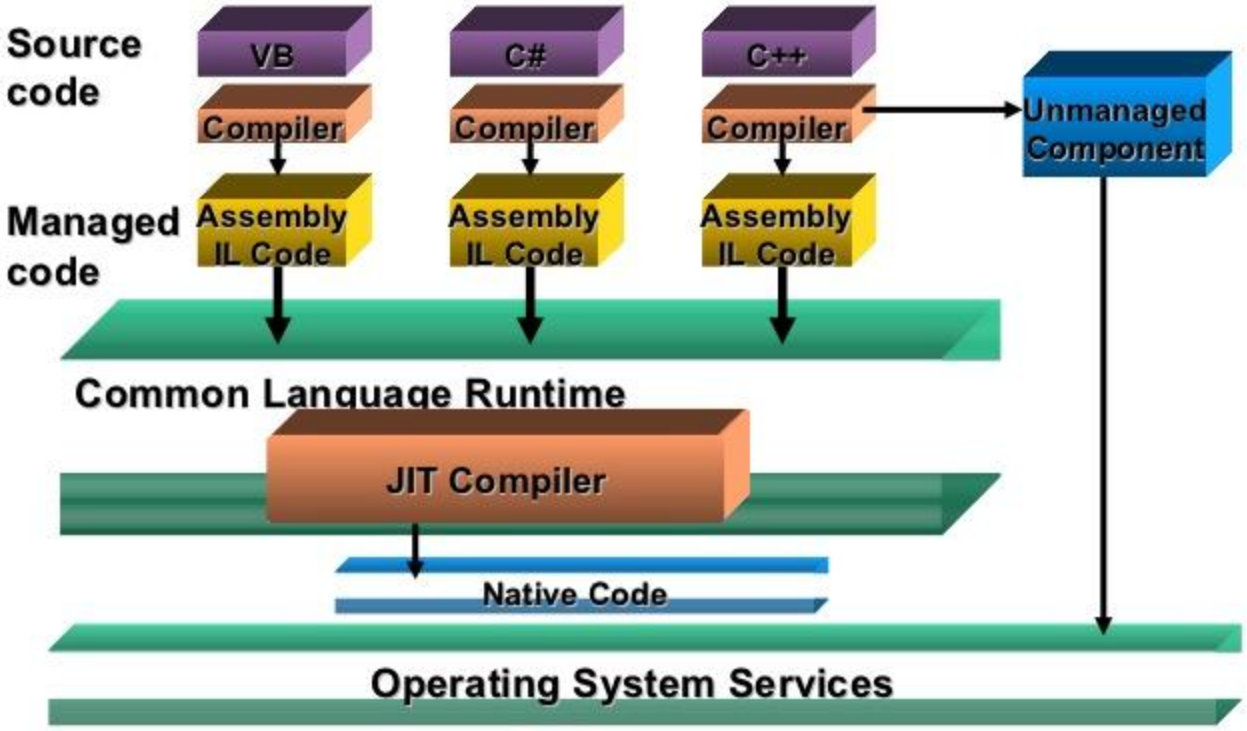
- Meio-termo entre compiladores e interpretadores.
- Uma linguagem de alto nível é traduzida para uma de linguagem intermediária que possua maior facilidade de interpretação.
- Mais rápido do que interpretação pura.
- Exemplos:
  - Programas Pearl são parcialmente compilados para detecção aprimorada de erros.
  - Implementações iniciais de Java eram híbridas. O formato intermediáril, byte code, permite a portabilidade para outras plataformas, desde que estas possuam um interpretador (run-time) chamado Java Virtual Machine.


Source program
Lexical analyzer
Lexical units
Syntax analyzer
Parse trees
Intermediate code generator
Intermediate code
Input data
Interpreter
Results

Sistemas ‘just-in-time’

- Inicialmente convertem o código-fonte para uma linguagem intermediária.
- Então, compilam este código para linguagem de máquina.
- Código de máquina é mantido para atender a novas chamadas.
- Método utilizado por Java e .Net.

CLR: Execution Model



http://image.slidesharecdn.com/netframework-111214212155-phpapp01/95

Pré-processadores

- Macros para pré-processamento são normalmente utilizadas para definir como código deve ser inserido a partir de um arquivo-fonte em outro.
- O pré-processamento é executado imediatamente antes da compilação propriamente dita, de forma que todas as macros são expandidas, gerando um novo código-fonte ‘completo’.
  - Exemplo: pré-processador C

```
#include
#define
```

Ambientes de Desenvolvimento

- É a coleção de programas utilizados no desenvolvimento de outros programas.
- UNIX (e sistemas UNIX-like, como LINUX)
- Atualmente, muito utilizado através de uma GUI (como CDE, KDE, Gnome, Unit, etc).
- Borland Jbuilder, Eclipse, NetBeans.
- Microsoft Visual Studio.

Questões?

FACULDADE DOM BOSCO de Porto Alegre

www.filipomor.com