Aspectos preliminares

Marco A L Barbosa
malbarbo.pro.br

Departamento de Informática
Universidade Estadual de Maringá

Conteúdo

Razões para estudar conceitos de linguagens de programação

Razões para estudar conceitos de linguagens de programação

- Aumentar a capacidade de expressar ideias
- Melhorar as condições de escolha da linguagem apropriada para cada problema
- Aumentar a capacidade de aprender novas linguagens
- Melhorar o uso das linguagens já conhecidas
- Entender a importância da implementação
- Avanço da área de computação

Domínios de programação

Domínios de programação

Linguagens de programação com objetivos diferentes

- Desde controle de usinas nucleares até jogos em dispositivos móveis

Domínios de programação

Aplicações científicas

- Estruturas simples (arranjos e matrizes)
- Muitas operações com pontos flutuantes
- Fortran, C/C++
- Recentemente
  - Hardware especializado (GPU, TPU)
  - Python (numpy), Octave (Mathlab), Julia

Domínios de programação

Aplicações comerciais

- Inicialmente
  - Integração com banco de dados
  - Produção de relatórios
  - Armazenamento e manipulação de números decimais e texto
  - Cobol
- Atualmente
  - Muitos requisitos
  - Muitas linguagens

Domínios de programação

Inteligência artificial

- Manipulação de símbolos (estruturas encadeadas ao invés de arranjos)
- Criação e execução de código
- Lisp, Prolog
- Atualmente
  - Métodos numéricos
  - Python, C/C++

Domínios de programação

Software de sistema

- Software básico e ferramentas de suporte a programação
  - Sistemas operacionais
    - Construções para interfaceamento com dispostos externos
    - Eficiência devido ao uso contínuo
  - Compiladores e interpretadores
- E os softwares que são executados continuamente?
  - Navegadores, Sistemas web
- C/C++, D, Go, Rust

Domínios de programação

Sistemas Web

- Apresentação de conteúdo dinâmico
  - Código junto com a tecnologia de apresentação
  - Inicialmente
    - PHP, JavaScript
- Segurança
- Escalabilidade
- Desempenho
- Integração com sistemas existentes

Classes de linguagens

Classes de linguagens

É comum a seguinte classificação hierárquica:

- Imperativas
  - Procedurais (Fortran, Pascal, C, ...)
  - Orientada a Objetos (Smalltalk, Java, C++, ...)
- Declarativas
  - Funcionais (Lisp/Scheme, Haskell, Ocaml, ...)
  - Lógicas (Prolog)

Classes de linguagens

A maioria das linguagens é multiparadigma, por isso, ao invés de classificação hierárquica é mais útil identificar as características de cada paradigma presente em uma linguagem.

Classes de linguagens

A seguir, mostramos como o problema de encontrar o valor máximo em uma lista não vazia de números inteiros pode ser resolvido utilizando os paradigmas imperativo, funcional e lógico.

Imperativo

- Inicialize a variável `max` com o primeiro elemento da lista
- Para cada elemento `x` da lista a partir do segundo elemento, faça:
  - Se `x` é maior do `max`, então atribua o valor `x` para `max`
- A variável `max` contém o maior valor da lista

Imperativo

Código em C

```c
int maximo(int lst[], size_t n)
{
    assert(n > 0);
    int max = lst[0];
    for (size_t i = 1; i < n; i++) {
        if (lst[i] > max) {
            max = lst[i];
        }
    }
    return max;
}
```

Funcional

- O valor `maximo(lst)` é definido como:
  - O primeiro elemento de `lst` se `lst` só tem um elemento
  - O primeiro elemento de `lst` se ele é maior do que o `maximo` do restante de `lst`
  - `maximo` do restante de `lst` se ele é maior ou igual ao primeiro elemento de `lst`
- Para computar o valor de `maximo` de uma dada lista, expanda e simplifique esta definição até que ela termine

Funcional

Código em Racket

```
(define (maximo lst)
  (cond
    [(empty? (rest lst))
     (first lst)]
    [(> (first lst) (maximo (rest lst)))
     (first lst)]
    [else (maximo (rest lst))]))
```

Lógico

- A proposição `maximo(lst, x)` é verdadeira se:
  - `x` é o único elemento de `lst`; ou
  - `x` é primeiro elemento de `lst` e existe um valor `m` tal que `m` é o máximo do restante de `lst` e `x` é maior do que `m`; ou
  - `x` é o máximo do restante de `lst` e existe um valor `p` que é o primeiro elemento de `lst` e `x` é maior ou igual a `p`
- Para computar o valor máximo de uma dada lista, busque por um valor `x` que permita provar que a proposição `maximo(lst, x)` é verdadeira.

Código em Prolog

```
maximo([X], X).
maximo([X | Xs], X) :- maximo(Xs, M), X > M.
maximo([P | Xs], X) :- maximo(Xs, X), X >= P.
```

Classes de linguagens

Nível de abstração

- Baixo nível
  - Poucas abstrações sobre os detalhes do computador
- Alto nível
  - Abstrações sobre os detalhes do computador

Classes de linguagens

Linguagens de *scripting*

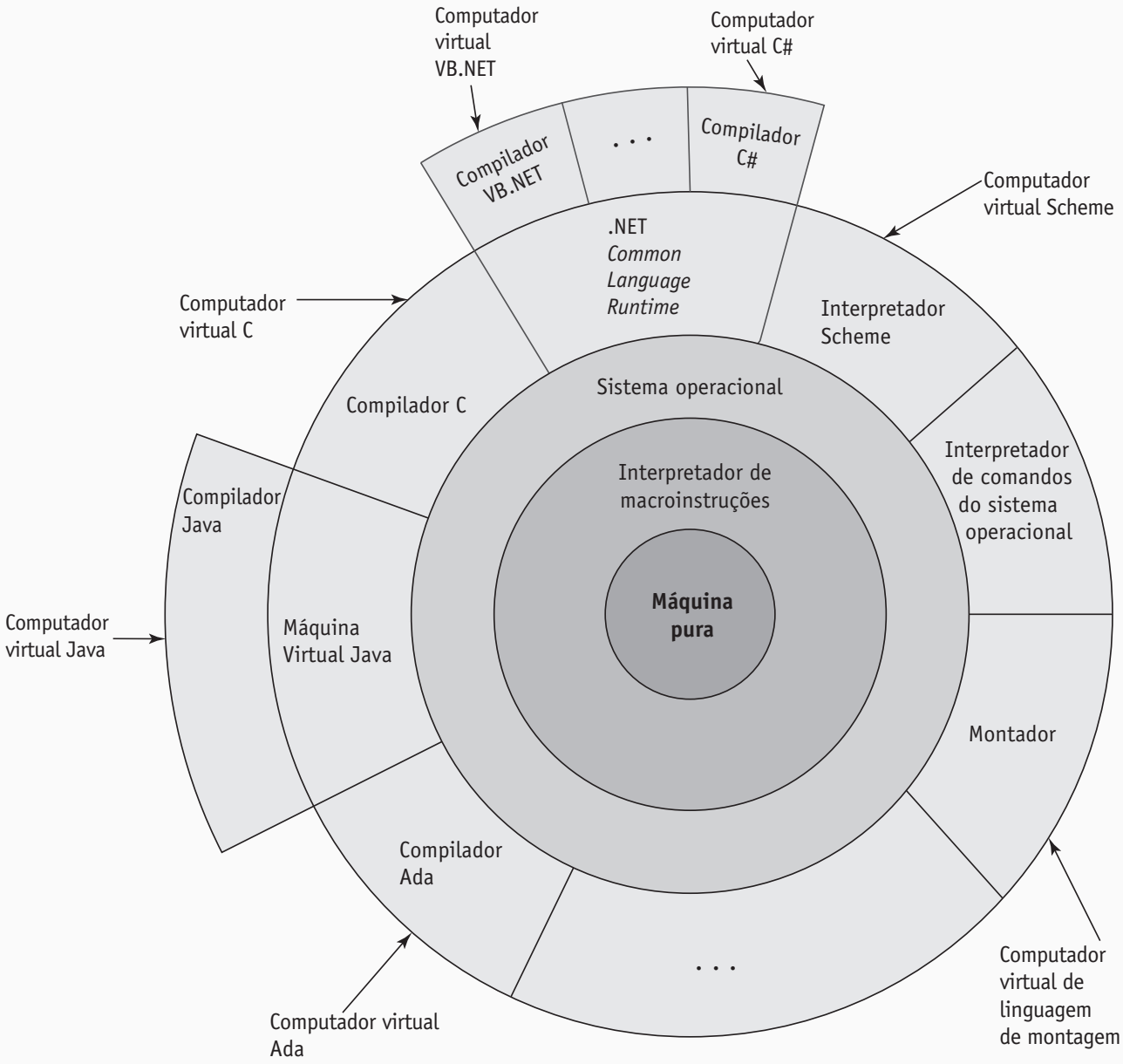
- "Automatizar" execução de tarefas que poderiam ser executadas manualmente
- Também usado para referenciar linguagens dinâmicas de propósito geral

Métodos de implementação

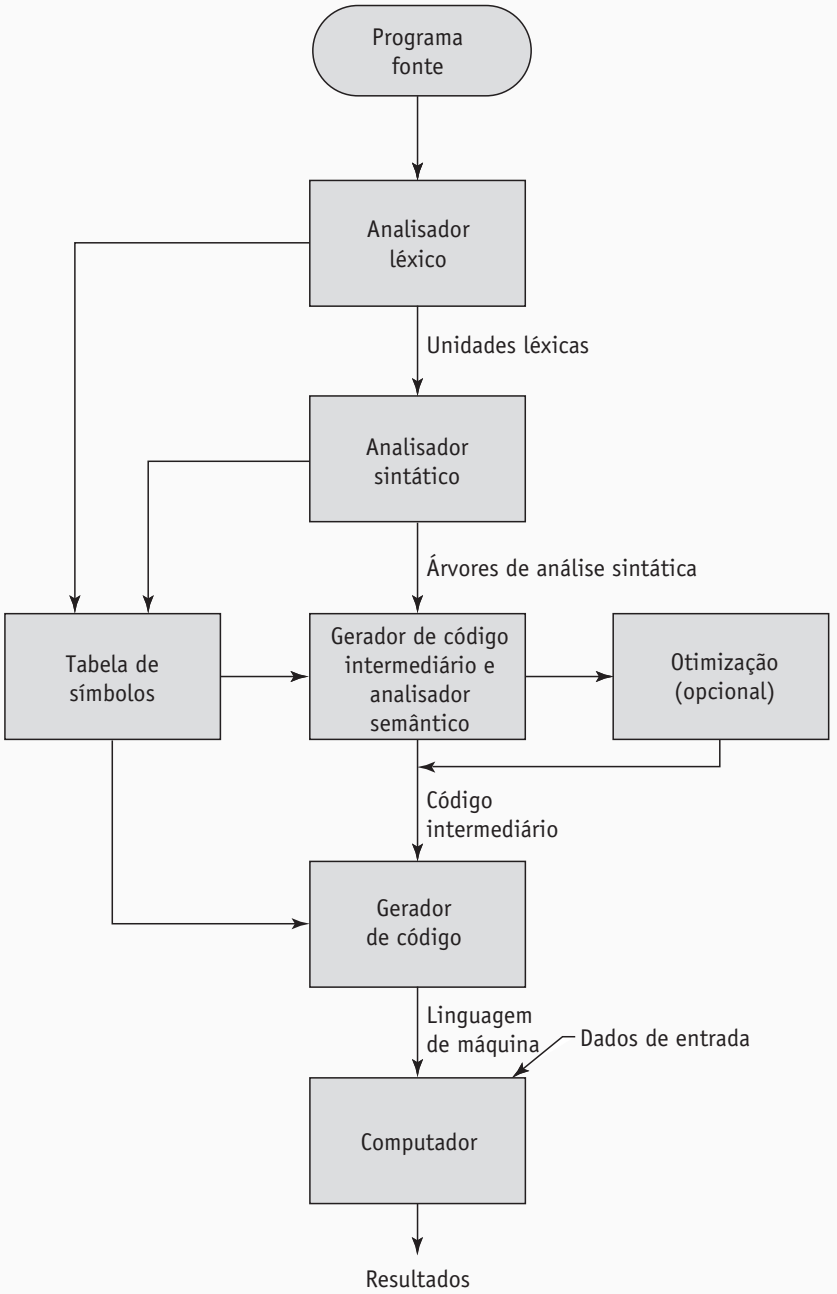
Métodos de implementação

- Compilação
- Interpretação
- Híbrido

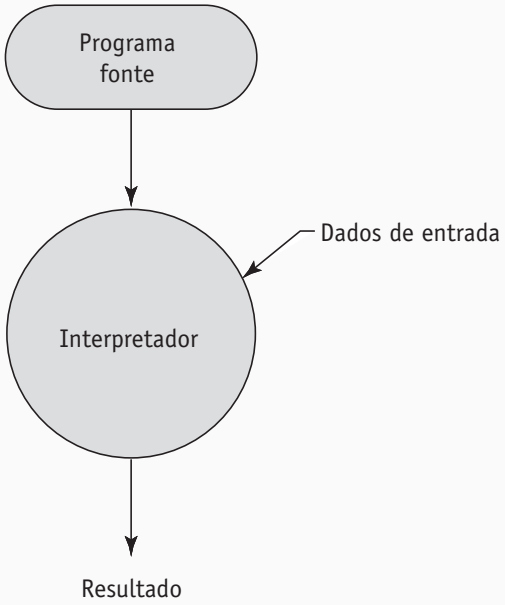
Interface em camadas de computadores virtuais, fornecida por um sistema de computação típico



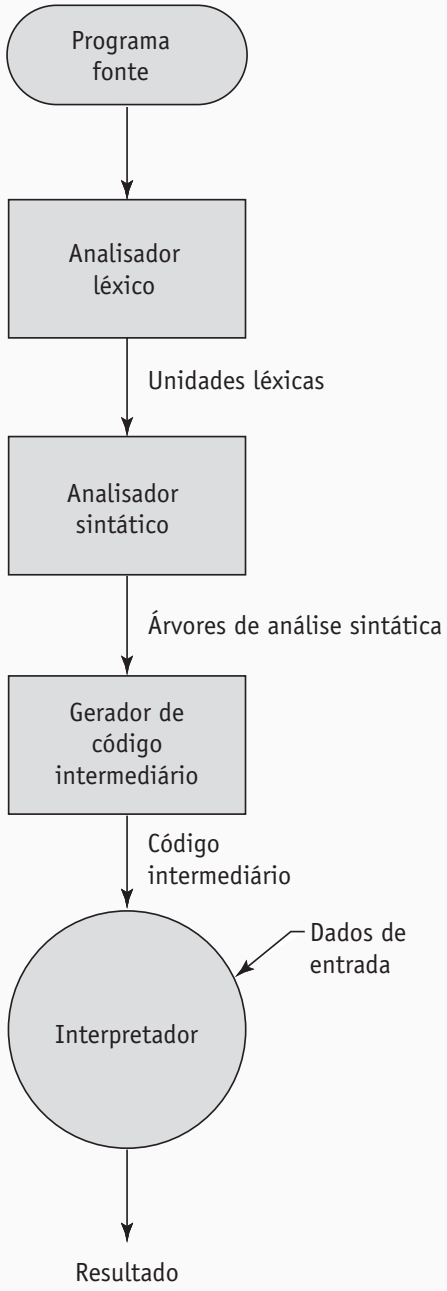
Compilação


Programa fonte
Analisador léxico
Unidades léxicas
Analisador sintático
Árvores de análise sintática
Tabela de símbolos
Gerador de código intermediário e analisador semântico
Otimização (opcional)
Código intermediário
Gerador de código
Linguagem de máquina
Dados de entrada
Computador
Resultados

Interpretação



Híbrido


Programa fonte
Analisador léxico
Unidades léxicas
Analisador sintático
Árvores de análise sintática
Gerador de código intermediário
Código intermediário
Dados de entrada
Interpretador
Resultado

Critérios para avaliação de linguagens

Critérios para avaliação de linguagens

- Como os recursos das linguagens influenciam o desenvolvimento de software?
- Alguns critérios podem ser controversos
- Alguns critérios são objetivos, enquanto outros não
- Algumas pessoas valorizam mais alguns critérios do que outros
  - O Sebesta valoriza muito as características que permitem que erros possam ser detectados em tempo de compilação, mas estas características podem tornam os programas mais difíceis de serem mantidos do que o necessário

Critérios para avaliação de linguagens

- Legibilidade (facilidade de leitura)
  - Deve ser considerada em relação ao domínio do problema
- Facilidade de escrita
  - Deve ser considerada em relação ao domínio do problema
- Confiabilidade
- Custo

Critérios para avaliação de linguagens

Característica	Legibilidade	Facilidade de escrita	Confiabilidade
Simplicidade	•	•	•
Ortogonalidade	•	•	•
Tipos de dados	•	•	•
Projeto de sintaxe	•	•	•
Suporte para abstração		•	•
Expressividade		•	•
Verificação de tipos			•
Tratamento de exceções			•
Apelidos restritos			•

Legibilidade

Simplicidade

- Um conjunto bom de características e construções
- Poucas formas de expressar cada operação
- Sobrecarga de operadores?
- Muito simples não é bom (assembly)

Legibilidade

Ortogonalidade

- Poucas características podem ser combinadas de várias maneiras
- Uma característica deve ser independente do contexto que é usada (exceções a regra são ruins)
- Muito ortogonalidade não é bom (Algol68)
- Linguagens funcionais oferecem uma boa combinação de simplicidade e ortogonalidade

Exemplo de falta de ortogonalidade em C

```c
typedef struct par {
    int primeiro;
    int segundo;
} par;

void f(par x) {
    x.primeiro = 10;
}

void test_f() {
    par x = {1, 2};
    // passagem de parâmetro
    // por valor
    f(x);
    assert(x.primeiro == 1);
}
```

```c
void g(int x[2]) {
    x[0] = 10;
}

void test_g() {
    int x[2] = {1, 2};
    // vetores não podem ser passados
    // como parâmetro por valor!
    g(x);
    assert(x[0] == 10);
}
```

Legibilidade

Tipos de dados

- Tipos pré-definidos adequados

Legibilidade

Sintaxe

- Flexibilidade para nomear identificadores
- Forma de criar sentenças compostas
- A forma deve ter relação com o significado

Facilidade de escrita

Simplicidade e ortogonalidade

- Poucas construções e um conjunto consistente de formas de combinação

Facilidade de escrita

Suporte para abstração

- Definir e usar estruturas e operações de maneira que os detalhes possam ser ignorados
- Suporte a subprogramas
- Suporte a tipos abstratos de dados

Facilidade de escrita

Expressividade

- Maneira conveniente de expressar a computação

Facilidade de escrita

Algumas linguagens tem a sintaxe e/ou a semântica tão densas e bizarras que são chamadas de “Linguagens somente de escrita”!

Confiabilidade

Um programa é dito confiável quando está de acordo com suas especificações em todas as condições

- Verificação de tipos
- Tratamento de exceções
- Apelidos (um ou mais nomes para acessar a mesma célula de memória)
- Facilidade de leitura e escrita

Confiabilidade

Os testes automatizados são extremamente importantes para aumentar a confiabilidade/manutenibilidade dos programas

- Especificação executável
- Frameworks de testes

Uso de apelidos em C++, Java e Rust

C++

```
vector<int> v = {10, 20, 30};
int soma = 0;
for (int &x : v) {
    soma += x;
    if (x == 20) {
        v.push_back(1);
    }
}
assert(soma == 61);
```

Pode ou não falhar... `x` pode referenciar memória desalocada

Java

```
ArrayList<Integer> lista =
    new ArrayList<>(
        asList(10, 20, 30));
int soma = 0;
for (Integer x : lista) {
    soma += x;
    if (x == 20) {
        lista.add(1);
    }
}
assert soma == 61;
```

Falha na execução: `lista.add` gera o erro `java.util.ConcurrentModificationException`

Rust

```
let mut v = vec![10, 20, 30];
let mut soma = 0;
for &x in &v {
    soma += x;
    if x == 20 {
        v.push(1);
    }
}
assert_eq!(soma, 61);
```

Falha na compilação: cannot borrow `v` as mutable because it is also borrowed as immutable

Custo

Os fatores que mais afetam os custos são

- Desenvolvimento
- Manutenção
- Confiabilidade

Custo

- Treinar programadores
- Escrever programas
- Aspectos importantes
  - Ambiente de desenvolvimento (IDE)
  - Gerenciamento de pacotes

Custo

- Compilar programas
- Executar programas
- Aspectos importantes
  - Compiladores/Interpretadores (software livre)
  - Linguagens com execução eficiente

Custo

- Confiabilidade
- Manutenção

Outros critérios para avaliação de linguagens

- Portabilidade
- Padronização

Diferentes visões

- Programador
- Projetista da linguagem
- Implementador da linguagem

Influências no projeto de linguagens

Influências no projeto de linguagens

Arquitetura do Computador

- Arquitetura de von Neumann
- Arquiteturas multicore
- Outras?

Metodologias de Programação

- Orientada a processos
- Orientada a dados
- Orientada a objetos

Referências

- Robert Sebesta, Concepts of programming languages, 9ª edição. Capítulo 1.