Linguagens de Programação

Aspectos Preliminares

M.F. Caetano

mfcaetano@unb.br

Departamento de Ciência da Computação
Universidade de Brasília

Conteúdo

Motivos para Estudar Linguagens de Programação

1. Aumentar a capacidade de expressar ideias;
   - Uma linguagem impõem limites às formas de algoritmos (POO, Estruturada, Funcional);
2. Melhorar a capacidade de escolher LP adequadas;
   - "*Quando só se tem um martelo, tudo parece prego*";
3. Facilitar aprendizagem de novas linguagens;
4. Entender melhor a importância da implementação;
   - Implementação de Matrizes em C;
   - algoritmo recursivo, normalmente, é mais lento do que iterativo equivalente;
5. Aumentar a capacidade de projetar novas LP;
   - não necessariamente uma linguagem completa (exemplo, parser);

Conteúdo

Domínios Típicos de Programação

O domínio da aplicação força o projeto de Linguagens de Programação no sentido de atender as suas especificações.

1. **Aplicações Científicas** (anos 40): Fortran;
   - Estrutura de Dados simples (arrays);
   - Grande volume de cálculos aritméticos;
   - estruturas de controle;
2. **Aplicações Comerciais** (anos 50): Cobol, planilhas e banco de dados;
   - Facilidade para produzir relatórios;
   - Precisão ao descrever e armazenar números;
   - Capacidade de especificar aritmética decimal;
3. **Inteligência Artificial**: LISP e Prolog;
   - Manipulação simbólica e não numérica;
   - Estruturas de dados: listas encadeadas;
   - Capacidade de executar código *runtime*;

Domínios Típicos de Programação

5. **Programação de Sistemas**: C, C++ e Assembly
   - Software básico (SO e ferramentas de suporte)
   - Eficiência na execução;
   - Recursos de baixo nível para interface com dispositivos externos;
6. **Linguagens *Scripting***: bash, csh, Perl e AWK
   - Permite a criação de um arquivo com uma lista de comandos a serem executados;
7. **Linguagens de Propósitos Especiais**: SPSS, Linguagem R
   - Características e aplicabilidade muito específicas torna difícil compará-las;

Conteúdo

Critérios para Avaliação da Linguagem

Em sua maioria, os critérios de avaliação de linguagem são subjetivos. De acordo com Sebesta, estão classificados em:

1. **Legibilidade** – facilidade como os programas podem ser lidos e entendidos (facilidade de manutenção vs custo);
2. **Capacidade de Escrita** – quão facilmente uma linguagem pode ser usada para criar programas para um domínio de problema escolhido;
3. **Confiabilidade** – associada ao comportamento do programa de acordo com suas especificações sob todas as condições;
4. **Custo** – valor efetivamente gasto com o sistema;
5. **Outros**

Critérios de Avaliação: Legibilidade

Simplicidade Global:

- Muitos componentes básicos é ruim (Assembly vs C – **for**);
- **Multiplicidade** de recursos (mais de uma maneira de fazer a mesma coisa);

```
count = count + 1;
count += 1;
count++;
++count;
```

- **Sobrecarga** (*overloading*) – um operador tem mais de um significado. C++, sobrescrever operador + com comportamento diferente. Ex. Soma elementos vetor.
- Linguagem demasiadamente simples (falta de estruturas de controle, por exemplo). Ex. Assembly.

Critérios de Avaliação: Legibilidade

Ortogonalidade: Toda combinação possível de conceitos primitivos é legal e significativo.

- Significado é independete de contexto;
- Torna a linguagem fácil de aprender e ler;
- Ex. Ponteiros em C. Ponteiros devem ser capazes de apontar para qualquer tipo de variável ou estrutura;
- Entretanto, muita ortogonalidade torna a linguagem mais complexa;

Instruções de Controle: Um programa que pode ser lido de cima para baixo é muito mais fácil de entender.

- Uso de Goto em Fortran.

Critérios de Avaliação: Legibilidade

Tipos de dados e Estruturas: auxilia em obter significado do código. Ex. Tipos booleanos, registros e listas.

Aspectos da Sintaxe:

- Identificadores pequeno é muito ruim. Ex. nome de variáveis;
- palavras especiais (delimitadores): bom;
- forma e semântica compatível: bom.
  - uso de **static** em C. Uso com *variável local* significa que variável é criada em tempo de compilação. Uso em *variável global* significa restrição do escopo ao arquivo corrente.

Critérios de Avalição: Capacidade de Escrita

Medida da facilidade de se usar uma Linguagem de Programação para criar programas em um certo domínio.

Simplicidade e Ortogonalidade:

- Desejável poucos componentes básicos e regras consistentes para combiná-los;
- Muita ortogonalidade é de difícil depuração (erros podem não ser detectados, uma vez que todas as combinações de primitivas são legais).

Suporte para Abstração: A capacidade de definir/usar estruturas ou operações complicadas, abstraindo detalhes, facilitando a escrita de programas.

- Abstração de processo: funções;
- Abstração de dados: classes, registros, etc...

Critérios de Avalição: Capacidade de Escrita

Expressividade: Existência de formas convenientes de especificar computações que facilita a escrita;

Na linguagem C, utilizar:

```
count++;
```

Ao invés de:

```
count = count + 1;
```

Critérios de Avalição: Confiabilidade

Programa é confiável quando se comporta de acordo com as especificações em todas as situações.

Verificação de Tipos: Verificação em tempo de compilação (desejável) ou execução.

Tratamento de Exceções: Capacidade de interceptar erros na execução, por em prática medidas corretivas e prosseguir.

```
try {
  file.open ("test.txt");
  while (!file.eof()) file.get();
  file.close();
} catch (std::ifstream::failure e) {
  std::cerr << "Exception opening/reading/closing file\n";
}
```

Critérios de Avalição: Confiabilidade

***Aliasing*:** ter dois ou mais métodos, ou nomes, distintos para fazer referência a mesma célula de memória (Membros de união e ponteiros): **perigoso!**

Legibilidade e Capacidade de Escrita: Quanto mais fácil for escrever um programa, mais provável que ele seja correto, mais fácil entender e mais fácil mantê-lo.

Critérios de Avalição: Custo

O custo final de uma Linguagem de Programação é em função de suas características.

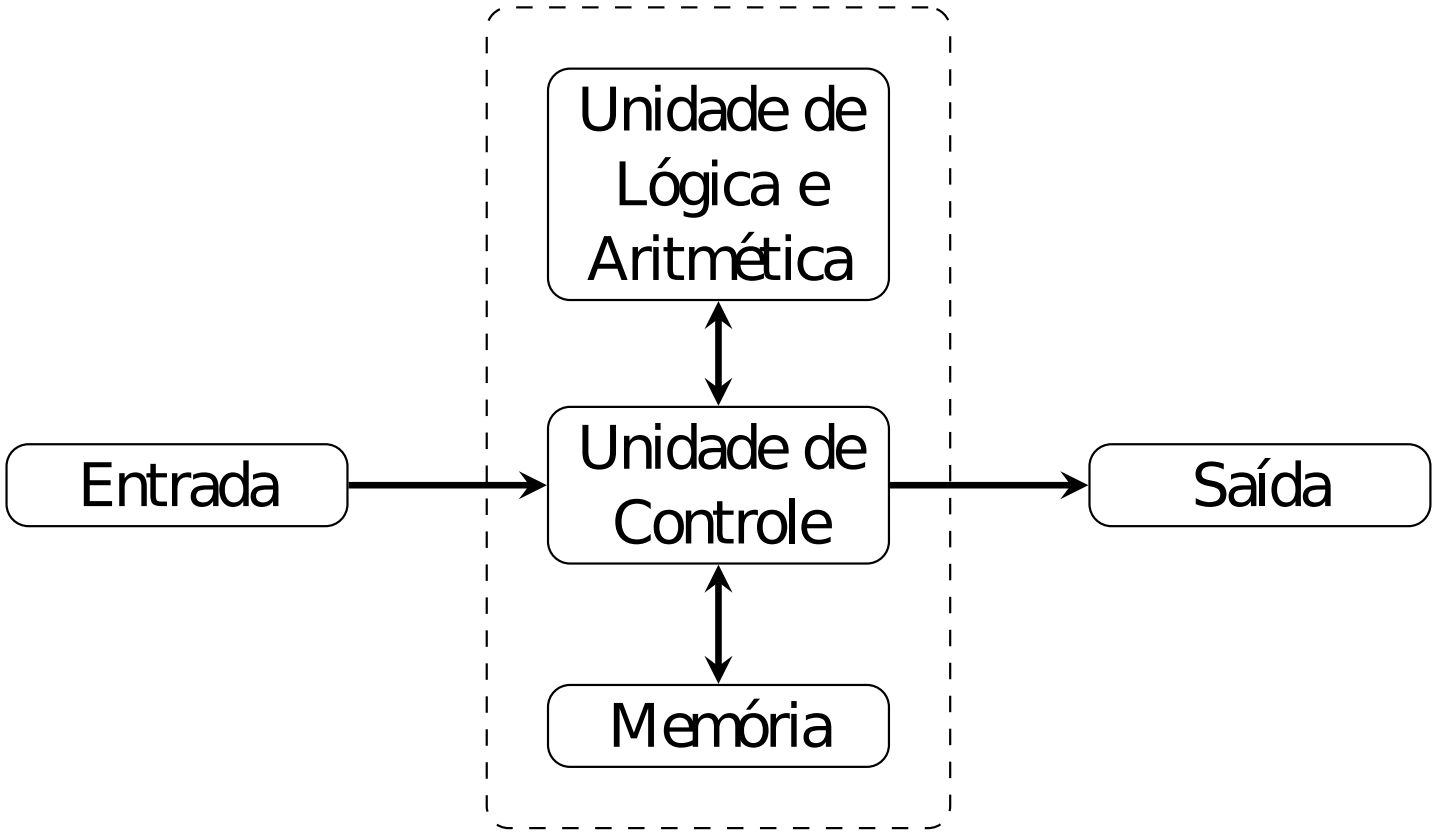
- Custo de treinamento;
- Custo para escrever programas;
- Custo para compilar programas (No Gentoo, kde +- 8 horas);
- Custo para executar programas;
- Custo de implementação do sistema;
- Custo de Manutenção (2 a 4 vezes custo de implementação).

Critérios de Avalição: Outros

- **Portabilidade** – Facilidade em que os programas podem ser migrados entre plataformas;
- **Generalidade** – aplicação a um ampla faixa de aplicações;
- **Qualidade de Definição** – Documentação oficial e especificação da Linguagem de Programação.

Conteúdo

Influências Sobre o Projeto da Linguagem



- Arquitetura exerceu influência sobre o projeto de LP;
- Memória (armazena tanto instruções como dados);
- Linguagens imperativas: recursos centrais são variáveis (carrega, executa, armazena);

Influências Sobre o Projeto da Linguagem

"*Usamos linguagens imperativas porque usamos a arquitetura de Von Neumann*".

- Dados e programas armazenados na memória;
- Ciclo de busca-decodificação-busca de operandos (*fetch-execute cycle*);
- Processamento é sequencial (localidade de referência);
- Execução rápida pois não são armazenados em posições adjacentes da memória;
  - Efeito do desvio condicional (goto)?
- Recursão torna-se ineficiente nesta arquitetura (alocação recursiva de recurso).
- **Gargalo:** capacidade de transferência dos barramentos (CPU p/ memória e periféricos p/ CPU).

Influências Sobre o Projeto da Linguagem

Técnicas para redução do gargalo na arquitetura Von Neumann:

- **Redução na diferença de acesso entre CPU e memória:**
  - Memória cache;
  - Processadores com pipelining;
- **Redução na diferença de acesso entre Memória de Acesso Direto e Dispositivos E/S:**
  - blocagem e bufferização;
  - DMA (Acesso Direto à Memória);
  - Canais de controle de dispositivos de E/S;

Influências Sobre o Projeto da Linguagem

Metodologias de Programação: Preocupação migrou de maximizar uso do hardware para maximizar a produtividade e manutenibilidade.

- **Anos 50 e início dos anos 60:**
  - Aplicações simples;
  - Preocupação: eficiência do uso do hardware;
- **Final dos anos 60 (programação estruturada):**
  - Pessoas tornam-se importantes (programadores);
  - Preocupação: legibilidade e melhores estruturas de controle;
  - projeto *top-down* e refinamento sucessivo;
- **Final dos anos 70:**
  - Abstração de dados;
  - Preocupação: concentração no uso de tipos de dados abstratos para resolver problemas (ex. *structs*);
- **Meados dos anos 80 (Orientação Objetos: reuso):**
  - Abstração de dados (classes), encapsulamento e herança;

Conteúdo

Paradigmas de Linguagens de Programação

1. **Imperativas:**
   - Algoritmo é especificado com detalhes;
   - A ordem de execução dos comandos é incluída;
   - Possui comando de atribuição;
2. **Funcionais (ou aplicativas):**
   - Principal meio de fazer computações é aplicando funções a determinados parâmetros;
   - Não possui comando de atribuição;
   - Utiliza recursão extensamente;
   - Fluxo de execução é determinado pelas funções;
3. **Lógicas:**
   - Regras especificadas sem ordem particular;
   - Implementação escolhe a ordem de execução;
   - Fluxo de execução é determinado pelas regras;
4. **Orientadas a Objetos:**
   - abstração de dados encapsula processamento com objetos de dados e oculta o acesso a eles;
   - herança e vinculação de tipos em tempo de execução;

Conteúdo

Compromissos de Projeto

Existem tantos critérios importantes, contudo, conflitantes.

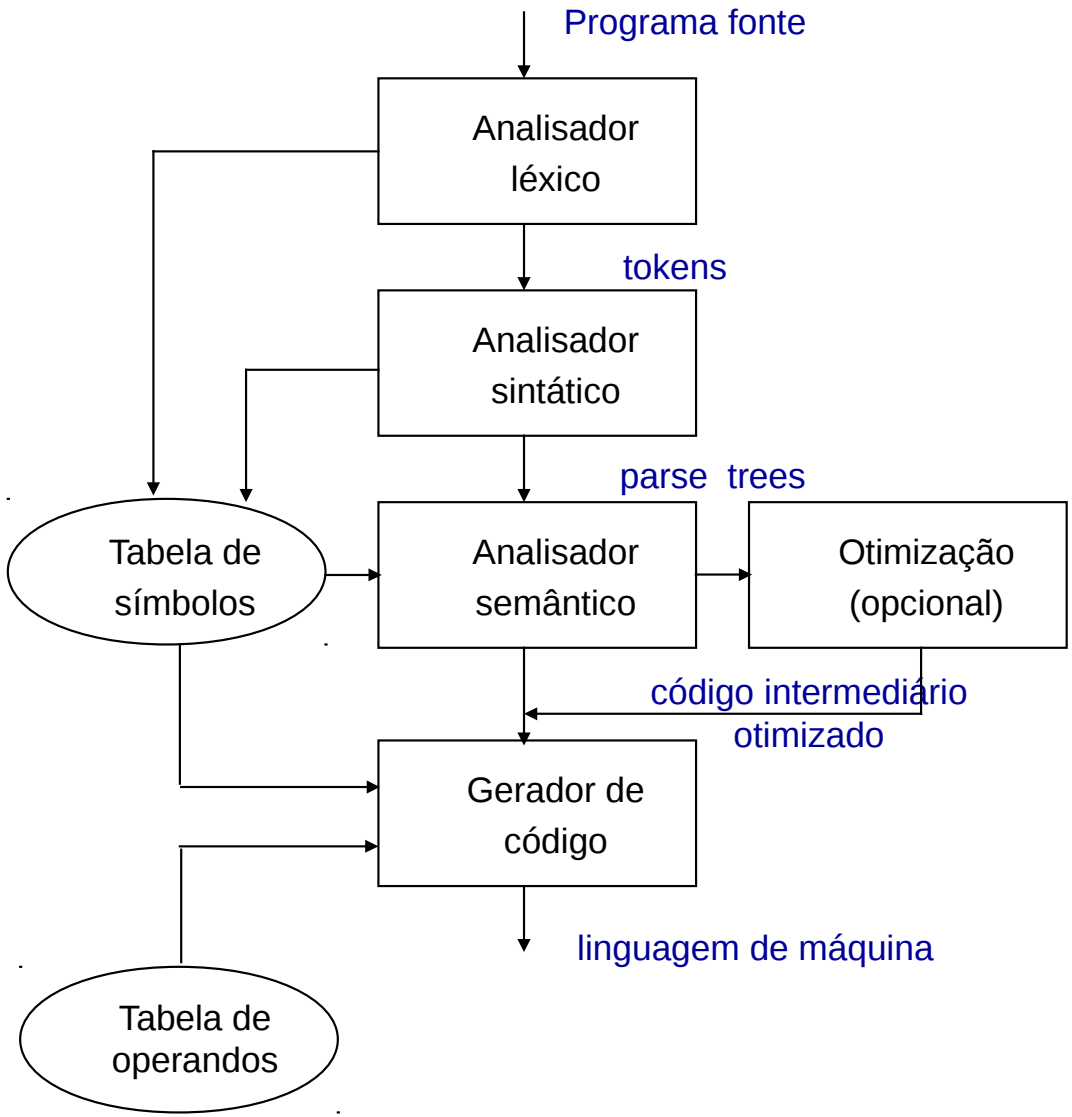
- **Confiabilidade vs Custo de Execução:**
  - **Validação de Subscritos** – verificação de todas as referências aos elementos do *array* para assegurar que o índice estejam em suas faixas legais;
- **Capacidade de escrita vs legibilidade:**
  - **Linguagem APL** – uma enorme quantidade de computação pode ser especificada em um programa muito compacto (**Fácil de Escrever e Difícil de Entender**).
    - muitos operadores para operações de *arrays*;
    - expressões longas e complexas;
- **Flexibilidade vs segurança:**
  - **Linguagem Pascal** – os **registros variantes** (*variant records*) que permitem que uma célula de memória contenha diferentes valores de tipo em diferentes momentos.
    - impossibilidade de verificação estática de tipos.

Conteúdo

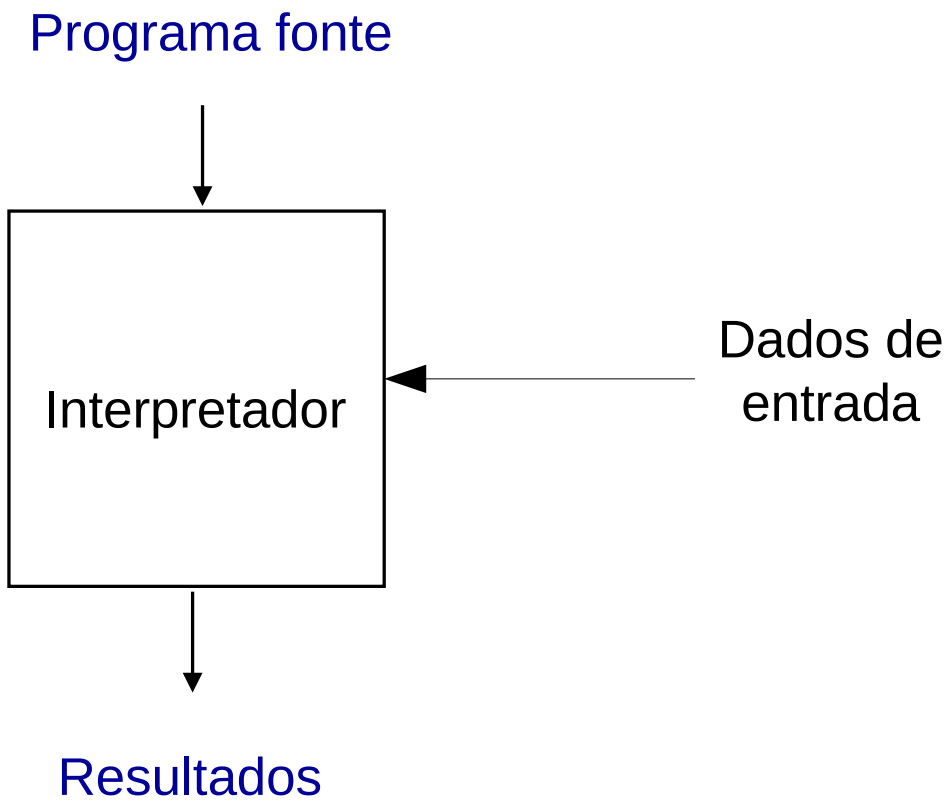
Métodos de Implementação

- **Compilação:**
  - Tradução de instruções de alto-nível para código de máquina;
  - Tradução demorada;
  - Execução rápida;
- **Interpretação Pura:**
  - Sem tradução;
  - Execução lenta;
  - Raro hoje em dia, sendo mais utilizada para linguagens scripts;
- **Sistema de Interpretação Híbrida:**
  - Custo de tradução é pequeno;
  - Velocidade média de execução.

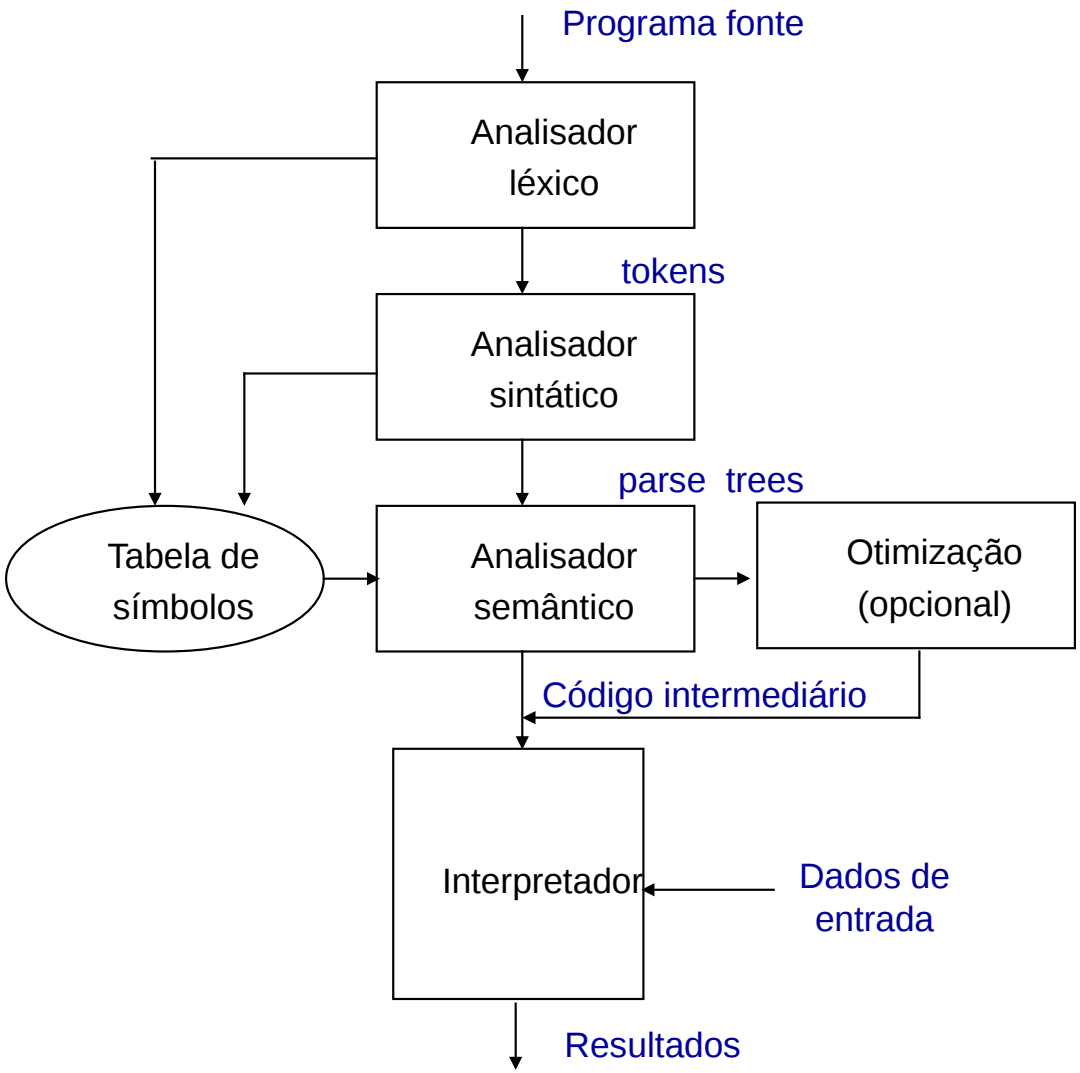
Métodos de Implementação: Compilação



Métodos de Implementação: Interpretação



Métodos de Implementação: Híbrida



Conteúdo

Computadores Virtuais