



Paradigmas de Linguagens de Programação

Prof. Salvador Ramos Bernardino
sramosbs@gmail.com / ssilva@uea.edu.br

WhatsApp: (92) 996 08-4591

Materiais da Disciplina:

<https://sites.google.com/a/uea.edu.br/salvador/plp>



Paradigmas de Linguagens de Programação

Esta disciplina...

- é um estudo **amplo** dos conceitos de linguagens de programação.
- **não** é um estudo **aprofundado** de linguagens de programação específicas.
- Examinaremos **questões sobre os fundamentos estruturais** que tornam uma linguagem de programação o que ela é.



Tópicos a serem estudados


- Domínios de Programação
- Critérios de Avaliação de Linguagens
- Influências sobre o Projeto de Linguagens
- Categorias de Linguagens
- Métodos de Implementação
- Ambientes de Programação



Linguagens de Programação


O que é uma linguagem de programação?

- Pergunte a vários cientistas da computação e você obterá várias respostas!
 - Notação formal para realizar computações
 - Ferramenta para escrever programas
 - Meio de comunicação entre programadores
 - Notação para algoritmos
 - Ferramenta para experimentar soluções para problemas
 - Meio para controlar dispositivos computadorizados...




Porque estudar conceitos de linguagens de programação?

- Aumenta a capacidade de expressar ideias
 - Da mesma forma que a linguagem natural
 - Novos paradigmas e novas formas de pensamento
- Mais conhecimento na escolha apropriada de linguagens
 - Conhecer os recursos da linguagem ajuda na adequação
- Maior capacidade para aprender novas linguagens
 - www.tiobe.com/tiobe_index/index.htm: Java, C, C++, C#, Python



Porque estudar conceitos de linguagens de programação?

- Entender a importância da implementação
 - Pode levar a um uso mais inteligente da linguagem, como foi projetada para ser usada.
 - Ex. repetidas chamadas a subprogramas pode ser ineficiente.
- Otimizar o uso da linguagem
 - O estudos dos conceitos pode levar ao conhecimento de partes da linguagem antes desconhecidos.
- Avanço global da computação
 - Algol 60 x FORTRAN




Porque estudar conceitos de linguagens de programação?

- Técnicas de computação em geral, e as linguagens de programação em particular, mudam tão rapidamente que o conhecimento detalhado de uma dada linguagem de programação torna-se obsoleto em uma década.



Domínios de Programação

- Aplicações Científicas
 - Década de 1950: Computação com matrizes e números de ponto flutuante (Assembly e Fortran-1957)
- Aplicações Comerciais
 - Década de 1960: Produção de relatórios, números decimais e caracteres (COBOL-1960)
- Inteligência Artificial
 - Manipulação de símbolos em lugar de números (1959 - Lisp, início dos anos 1970 Prolog)
- Programação de Sistemas
 - Software básico, eficiência necessária e uso contínuo (IBM-PL/S, Digital-BLISS, Burroughs-ExALGOL, UNIX-C)
- Linguagens de Scripting
 - Arquivo com lista de comandos para serem executados



O que é um paradigma de programação?

- Modelo, padrão ou estilo de programação suportado por linguagens que agrupam certas características comuns
- A classificação de linguagens em paradigmas é uma consequência de decisões de projeto que impactam radicalmente a forma na qual uma aplicação real é modelada do ponto de vista computacional



Principais paradigmas de programação

- Imperativo
 - Características centradas em variáveis, atribuições e iteração.
 - C, Pascal, Ada
- Funcional
 - Principal forma de fazer computação é aplicando funções sobre parâmetros fornecidos
 - LISP, Scheme, ML, Haskell, Miranda, Erlang



Principais paradigmas de programação

- Lógico
 - Baseadas em regras e fatos
 - Não há ordem especial nas regras
 - Prolog
- Orientado a objeto
 - Objetos encapsulam dados e comportamento
 - Herança e ligação dinâmica de tipo
 - Evoluíram a partir das LPs imperativas
 - C++, Java, Ada 95, Smalltalk



Outras Categorias de linguagens

- Linguagens paralelas
 - Suportam criar mais facilmente threads que podem ser executadas em paralelo
 - Devem também suportar coordenação and compartilhamento
- Linguagens de marcação (mark-up languages)
 - HTML, XML
 - Não especificam computações, logo não são LPs
 - Projeto, avaliação implementação similar a PL
 - Não discutido aqui
- Linguagens visuais (de 4a. geração)
 - Visual BASIC, Visual BASIC .net.
 - Geração de parte do código com *drag-and-drop*.
 - Facilitam desenvolvimentos de GUIs.



Custo/Benefício no projeto de linguagens

- Confiabilidade X Custo de execução
 - Verificação de limites de array (Java, não C)
 - Aumentar confiabilidade, diminuir eficiência
- Legibilidade X Capacidade de escrita
 - APL tem muitas funções internas (com símbolos especiais)
 - Fácil de escrever; difícil de ler
 - Perl tem problemas similares
- Flexibilidade X Segurança
 - Registros variantes. Podem armazenar valores diferentes no mesmo campo.
 - Flexível, mas perigoso em tempo de execução.

Critérios de Avaliação de uma Linguagem

Critérios

Características	Legibilidade	Facilidade de escrita	Confiabilidade
Simplicidade	X	X	X
Ortogonalidade	X	X	X
Tipos de dados e estruturas	X	X	X
Projeto da sintaxe	X	X	X
Suporte à abstração		X	X
Expressividade		X	X
Checagem de tipo			X
Tratamento de exceção			X
Restrição de aliases			X



Critérios e características: Legibilidade

- Facilidade com que os programas podem ser lidos e entendidos.
- Deve ser considerada no contexto do domínio de aplicação.
- Antes de 1970, escrita de código era vista como mais importante.
- Após 1970 (advento do ciclo de desenvolvimento de software: análise de requisitos à operação), a manutenção passou ser mais importante (custo).
 - A legibilidade passou a ser uma medida importante da qualidade dos programas e das linguagens.
- Manutenção depende da boa legibilidade!



Critérios e características: Legibilidade

- Simplicidade global
 - Excesso de recursos é ruim (curva de aprendizado grande; programador aprende subconjunto da LP)
 - Poucos recursos é ruim (exemplo: assembly)
 - Multiplicidade de recursos é ruim (mais de uma maneira de fazer algo)
 - `count = count + 1; count += 1; count++; ++ count;`
... basicamente produzem o mesmo efeito!
 - Sobrecarga de operadores
 - Pode reduzir legibilidade se programador sobrecarrega sem critério
 - Exemplo: Sobrecarga de `+` para significar a soma de todos os elementos de um vetor.



Critérios e características: Legibilidade

- Ortogonalidade
 - Conjunto pequeno de primitivas de construção podem ser combinadas em um número relativamente pequeno de maneiras para construir estruturas de controle e de dados da linguagem.
 - Qualquer combinação de primitivas é legal e significativa.
 - Exemplo:
 - 4 tipos primitivos: int, float, double, char
 - 2 operadores de tipo: array, pointer
 - Operadores podem ser aplicados a si próprios e aos tipos primitivos
 - Se não fosse permitido **ponteiros para ponteiros para arrays**, isso limitaria as possíveis estruturas de dados.
 - **Falta** de ortogonalidade conduz a exceções às regras.



Critérios e características: Legibilidade

- Exemplo de pouca ortogonalidade (C):
 - Funções podem retornar *structs*, mas não *arrays*.
 - Um membro de um *struct* não pode ser outro *struct* ou um *void*.
 - Um elemento de um *array* não pode ser uma função ou um *void*.
 - Todos os parâmetros são passados por valor exceto *arrays*.
- Excesso de ortogonalidade é ruim (ALGOL)
 - Pode haver qualquer declaração do lado esquerdo de uma atribuição (mesmo uma declaração *if* !)
- Boa ortogonalidade: linguagens funcionais



Critérios e características: Legibilidade

- Tipos de dados
 - A presença de mecanismos adequados para definir tipos e estruturas de dados.

Exemplo:

Em uma linguagem que tem o tipo boolean é mais legível

terminou = **true**


do que

terminou = 1



Critérios e características: Legibilidade

- Considerações sobre sintaxe
 - Identificadores: identificadores grandes são mais legíveis (BASIC, FORTRAN 77)
 - Palavras especiais
 - Legibilidade é afetada por palavras como **while**, **for**...
 - Palavras especiais para finalizar estruturas de controle (**end loop**, **end if**) fazem a linguagem mais legível
 - Palavras especiais como identificadores? (FORTRAN)
 - Forma e significado
 - Semântica deveria seguir da sintaxe/forma
 - Não usar palavras especiais para 2 significados distintos (**static** em **C**; dentro de funções cria variável em *compile time*, e fora restringe uso ao arquivo).



Critérios e características: Facilidade de escrita (*writability*)

- Medida de quão facilmente uma linguagem permite criar programas para um domínio de problema escolhido.
- Tudo que afeta legibilidade, afeta capacidade de escrita.
- Deve ser comparada no contexto de um domínio de problema alvo
 - C e Visual Basic têm boa facilidade de escrita em domínios diferentes.



Critérios e características: Facilidade de escrita

- Simplicidade e ortogonalidade
 - Difícil entender/usar corretamente linguagens “grandes”
 - Melhor ter poucas primitivas de construção e conjunto consistente de regras (ortogonalidade)
 - Muita ortogonalidade: tudo pode ser combinado; difícil de capturar erros
- Suporte a abstração
 - Permite ocultar detalhes para usá-los mais facilmente.
 - Dois tipos de abstração:
 - **processos**: subprogramas/módulos
 - **dados**: registros, vetores, classes, etc.



Critérios e características: Facilidade de escrita

- Expressividade

Formas convenientes para expressar estruturas de dados e computações

- Incremento em C, C++, Java, C#: **x++** em lugar de

x = x + 1

- Expressar loop de contagem:

for ou **while** ou **repeat**?

- Expressar classes:

struct de C, **record** de Pascal e Modula2? Ou

class de C++, Java, C#, Ruby, etc?

- Avaliação de curto circuito:

C/C++, Java sempre fazem;

Ada possui **and then**



Critérios e características: Confiabilidade

- Um programa confiável se comporta conforme suas especificações sob todas as possíveis condições de entrada
- Fatores:
 - Verificação de tipo
 - Tratamendo de exceções
 - *Aliasing*
 - Legibilidade e capacidade de escrita



Critérios e características: Confiabilidade

- Fatores:
 - Verificação de tipo (tempo de compilação)
 - Linguagens com verificação de tipo em tempo de compilação eliminam, praticamente, todos os erros de tipo.
 - C original: uma variável tipo **int** poderia ser usada como um parâmetro real em uma chamada a uma função que esperava um tipo **float**.
 - Tratamendo de exceções
 - presente em Ada, C++, Java, não em C ou Fortran
 - *Aliasing*
 - Duas variáveis apontam para o mesmo endereço de memória (como ponteiros).
 - Pode ser perigoso.
 - Restringir pode aumentar a confiabilidade.



Critérios e características

- Custo de...
 - treinamento: função da simplicidade, ortogonalidade e experiência dos programadores
 - escrita: função da capacidade de escrita, adequação ao domínio da LP (+exp. programador).
 - Compilação: velocidade, precisão, confiabilidade, etc. Exemplo: primeiro compilador Ada era lento!
 - Execução dos programas (eficiência)
 - Tempo de compilação X otimização
 - Sistema de “liberação” da LP (Java é gratuito e universal; Ada era originalmente cara)
 - Confiabilidade: máquina de raios X, usina nuclear
 - Manutenção em programas (legibilidade).



Critérios e características

- Custo
 - Categorias mais importantes:
 - Desenvolvimento de programas
 - Manutenção
 - Confiabilidade
 - Os três dependem de confiabilidade e de capacidade de escrita.
- Outros critérios
 - Portabilidade (padronização)
 - Generalidade (boa para muitos domínios de problemas)
 - Boa definição, etc.

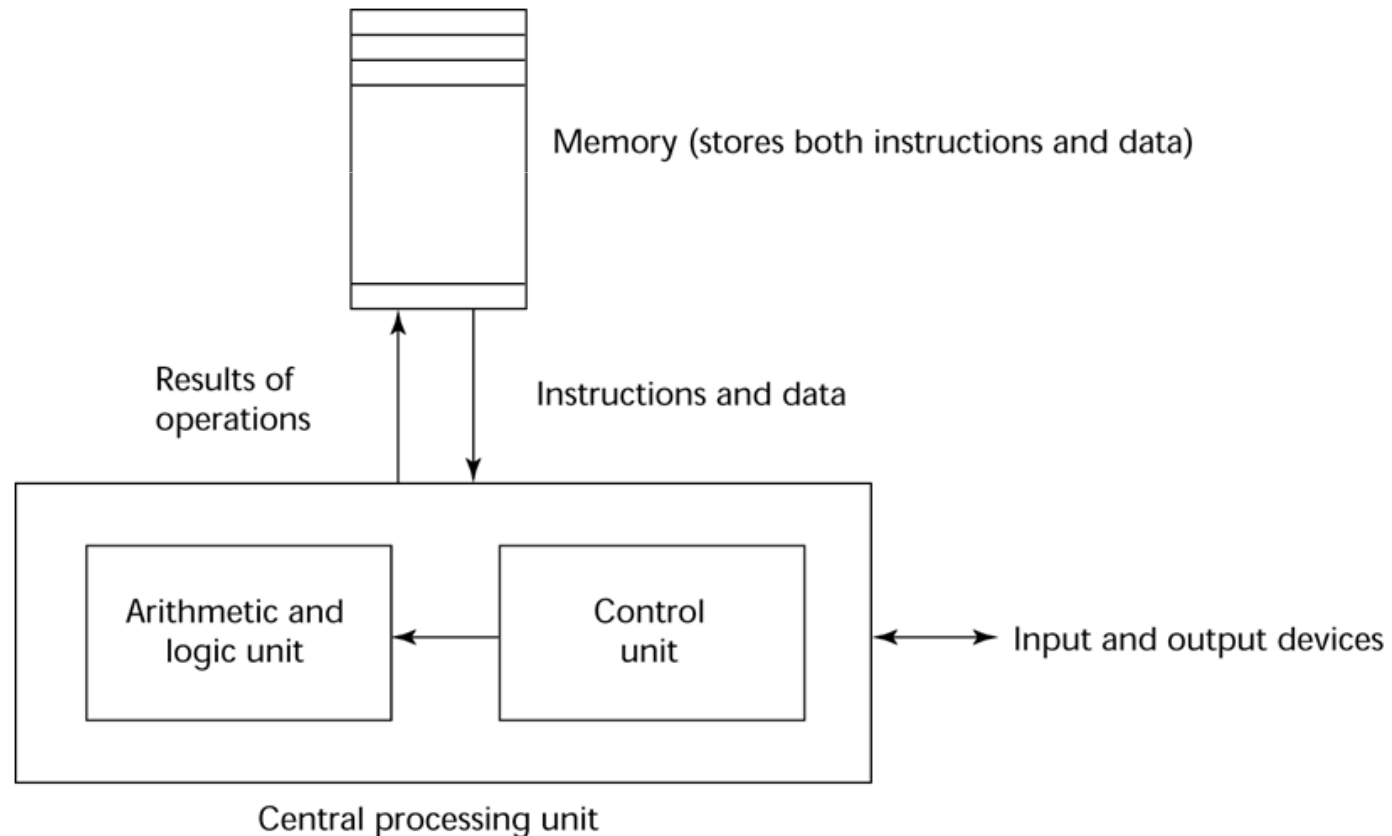



Critérios e características

- Há três pontos de vista a se considerar sobre uma linguagem de programação:
 - **Designer** (o inventor da linguagem)
Elegância
 - **Implementador** (quem escreve o compilador)
Simplicidade de implementação dos construtos e características
 - **Usuário** (o programador)
capacidade de escrita/legibilidade

Influências sobre o projeto de uma linguagem


- Arquitetura dos computadores atuais:
Von Neumann






Influências sobre o projeto de uma linguagem

- Linguagens imperativas são inspiradas em máquinas von Neumann
 - Dados e programas armazenados na mesma memória
 - Memória separada da CPU
 - Instruções e dados são transferidos da memória para a CPU
 - Base para linguagens imperativas
 - Variáveis modelam células de memória
 - Atribuições modelam transferências
 - Iteração é eficiente (instruções próximas umas das outras na memória)




Influências sobre o projeto de uma linguagem

- Linguagens funcionais
 - Computação é a aplicação de funções sobre parâmetros
 - Não usa variáveis, atribuição, iteração
 - problema: arquitetura von Neumann não é eficiente para implementar linguagens funcionais.
 - Por exemplo, recursão é custoso
 - Arquiteturas paralelas dão suporte a imperativas (usam Fortran!)
 - Não há arquiteturas para linguagens funcionais!




Influências sobre o projeto de uma linguagem

- Metodologias de programação
 - Anos 1950 e início de 1960:
 - Aplicações “simples”
 - Preocupação sobre eficiência da máquina.
 - Hardware caro!
 - Final de 1960:
 - Eficiência das pessoas tornou-se importante: legibilidade, melhores estruturas de controle (fora goto!), verificação de tipos, etc.
 - Programação estruturada.
 - Projeto *Top-down* e refinamentos sucessivos.
 - Software caro!



Influências sobre o projeto de uma linguagem

- Metodologias de programação
 - Final de 1970: Processo orientado a dados
 - Abstração de dados (TADs, módulos)
 - SIMULA 67 primeira LP a suportar TADs
 - Meados de 1980: Programação orientada a objetos
 - Classe/objeto, encapsulamento, herança, ligação dinâmica de métodos.
 - Smalltalk primeira LP OO pura (1989)
 - Fácil suportar LPs imperativas: C++, Ada 95, Java, ...
 - Também em LISP (1988) e em Prolog++ (1994)



Influências sobre o projeto de uma linguagem

- Metodologias de programação
 - Atualmente: concorrência
 - Arquiteturas Multi-core
 - Suporte em Ada e Java