Universidade Federal da Bahia
Instituto de Computação – DCC/UFBA

Universidade Federal
da Bahia

Avaliação de Linguagens de Programação

Prof.: Claudio Junior (claudiojns@ufba.br)

Paradigmas de Linguagens de Programação (MATA56)

2023.1

Na aula anterior...

- Paradigma de linguagem de programação:
  - Modelo, padrão, exemplo ou ponto de vista;
  - Define como abstrair um problema do mundo real para a computação;
  - A visão que se tem sobre um programa ou linguagem de programação.
- Razões para estudar os conceitos de linguagem de programação:
  - Aumento na capacidade de expressar ideias;
  - Embasamento para escolha de linguagens apropriadas;
  - Aumento da capacidade para aprendizado de novas linguagens;
  - Uso da linguagem de forma mais eficiente;
  - Avanços tecnológicos existentes.
- Domínios de programação:
  - Aplicações científicas: casos de aplicações com grande número de operações;
  - Aplicações comerciais: relatórios, negócio;
  - Inteligência artificial;
  - Programação de Sistemas;
  - Softwares para Web

Agenda

- Critérios de Avaliação de uma LP
- Legibilidade
- Capacidade de Escrita
- Confiabilidade
- Custo

Critérios de Avaliação de uma LP

- Dificuldade para definir:
  - Inúmeras possibilidades;
  - Granularidade;
  - Dependência do contexto;
  - Opiniões divergentes.

Critérios específicos

Aplicabilidade	Confiabilidade	Facilidade de aprendizado	Eficiência	Portabilidade	Suporte ao Método de Projeto
Evolutibilidade	Reusabilidade	Integração com outros softwares	Custo	Escopo	Expressões e comandos
Tipos primitivos e compostos	Gerenciamento de memória	Persistência de dados	Passagem de parâmetros	Encapsulamento e proteção	Sistemas de tipos
Verificação de tipos	Exceções	Concorrência			

Comparação entre Linguagens de Programação

- Dificuldade para realizar;
- Não existe uma LP melhor do que outra
  - Depende do contexto de uso;
  - Depende do critério avaliado.
- Comparações refletem uma visão pessoal.

Programar é uma atividade criativa, ou seja, não é algo que pode ser, a princípio, automatizado, pois cada um de nós pode criar programas diferentes, com recursos diferentes, para resolver um mesmo problema (SEBESTA, 2018).

Critérios de Avaliação de uma LP

Legibilidade (*Readability*)

- Quão fácil ler e entender um programa

Capacidade escrita (*Writability*)

- Quão fácil usar uma linguagem para criar programas

Confiabilidade (*Reliability*)

- Conformidade com as especificações de acordo as condições impostas

Custo

- O custo final é um dos principais elementos na avaliação de qualquer LP

(Sebesta, 2018)

Critérios de Avaliação de uma LP

Características (Sebesta)	Critérios			
	Legibilidade	Capacidade de escrita	Confiabilidade	Custo
Simplicidade	✓	✓	✓	
Ortogonalidade	✓	✓	✓	
Tipos de dados e estruturas	✓	✓	✓	
Sintaxe	✓	✓	✓	
Suporte para abstração		✓	✓	
Expressividade		✓	✓	
Verificação de tipos			✓	
Manipulação de exceções			✓	
Aliasing restrito			✓	

Legibilidade

- Facilidade de leitura e entendimento dos programas;
- Importante para manutenção dos programas;

Antes (1970)

- Escrita do código
- Eficiência e legibilidade de máquina

Depois (1970)

- Ciclo de vida de software
- Codificação 2º plano
- Importância na manutenção
- Facilidade da manutenção em função da legibilidade

Legibilidade

- Deve ser considerada no nível do contexto do domínio do problema:
  - Um programa escrito em uma linguagem não apropriada se mostra antinatural e confuso, difícil de ser lido e consequentemente de ser entendido ... manutenido ... evoluído ...

Legibilidade – Simplicidade Global

- Afeta a legibilidade;
- Uma linguagem com um grande número de componentes básicos é mais difícil de ser manipulada do que uma com poucos desses componentes:
  - *Programadores que precisam usar uma linguagem grande tendem a aprender um subconjunto dela e ignorar seus outros recursos;*
  - Problemas de legibilidade: quando o leitor do programa aprende um conjunto diferente de recursos daquele que o autor aplicou em seu programa.

Legibilidade – Simplicidade Global

- Multiplicidade de recursos:
  - Existência de mais de uma maneira de realizar a mesma operação
  - Incrementar um contador:

i = i + 1

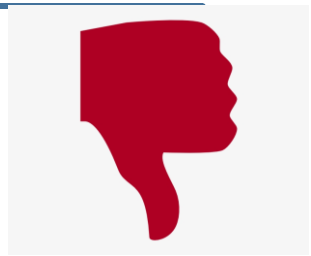
i += 1

i++

++i

Mesmo significado quando usadas em expressões separadas

Legibilidade – Simplicidade Global

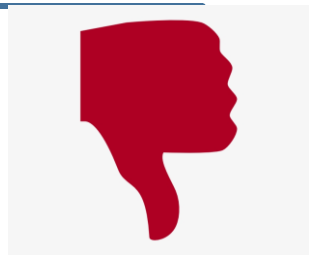


- Sobrecarga (*overloading*):
  - ... de operador, acontece quando um único símbolo tem mais de um significado;
  - Sinal de adição (+) pode ser usado para adição de números inteiros, para matrizes (arrays):

    10 + 10 resulta 20
    “10” + “20” resulta “1020”

  - Programação Orientada a Objetos: um mesmo método com diferentes assinaturas; uso de herança, sobrescrevendo um método da classe-pai (BERTAGNOLLI, 2009)

Legibilidade – Simplicidade Global



- Sobrecarga (*overloading*):
  - ... de operador, acontece quando um único símbolo tem mais de um significado;
  - Sinal de adição (+) pode ser usado para adição de números inteiros, reais, concatenar strings, somar vetores e ma matrizes (arrays):

    10 + 10 resulta 20
    “10” + “20” resulta “1020”
  - *Programação Orientada a Objetos: um mesmo método com diferentes assinaturas; uso de herança, sobrescrevendo um método da classe-pai (BERTAGNOLLI, 2009)*

Legibilidade – Simplicidade Global

- E o Assembly?
  - *Formas e modelos de simplicidade*

```
Endereço  Código        Assembly
--------  ------------  ------------------------------
1B8D:0100 01D8          ADD     AX,BX
1B8D:0102 C3            RET
1B8D:0103 16            PUSH    SS
1B8D:0104 B03A          MOV     AL,3A
1B8D:0106 380685D5      CMP     [D585],AL
1B8D:010A 750E          JNZ     011A
1B8D:010C 804E0402      OR      BYTE PTR [BP+04],02
1B8D:0110 BF86D5        MOV     DI,D586
1B8D:0113 C6460000      MOV     BYTE PTR [BP+00],00
1B8D:0117 E85F0B        CALL    0C79
1B8D:011A 8B7E34        MOV     DI,[BP+34]
1B8D:011D 007C1B        ADD     [SI+1B],BH
```

- A Falta instruções de controle mais complexas torna necessário o uso de mais códigos do que os necessários em linguagens de alto nível.

Legibilidade - Ortogonalidade

- Possibilidade de combinar entre si, sem restrições, os componentes básicos da LP:
  - Exemplo: permitir combinações de estruturas de dados como arrays de registros;
  - Contra exemplo: não permitir que um array seja usado como parâmetro de um procedimento.

Legibilidade - Ortogonalidade

- Falta de ortogonalidade em C:
  - C possui dois tipos de dados estruturados (*arrays* e registros/*structs*):
    - Registros podem ser retornados de funções, arrays não;
    - Um membro de estrutura pode ser qualquer tipo de dado, exceto *void* ou uma estrutura do mesmo tipo;
    - Um elemento de *array* pode ser qualquer tipo de dado, exceto *void* ou uma função;
    - Parâmetros são passados por valor, a menos que sejam *arrays* – que obrigatoriamente são passados por referência.

Legibilidade - Ortogonalidade

- Falta de ortogonalidade em C:
  - A + B
    - Valores de A e B são obtidos e adicionados juntos;
    - Se A for um ponteiro, afeta o valor de B;
      - Se A aponta para um valor de ponto flutuante que ocupa 4 bytes, o valor de B deve ser ampliado (multiplicado por 4) antes que seja adicionado a A;
      - Assim, o tipo de A afeta o tratamento do valor de B;
      - O contexto de B altera o seu significado.

Legibilidade - Ortogonalidade

- A falta de ortogonalidade acarreta exceções às regras de uma linguagem de programação, fazendo com que os desenvolvedores de *software* tenham que encontrar outras formas e/ou recursos para realizar suas implementações.

Legibilidade – Instruções de Controle

- A revolução da programação estruturada da década de 70 foi, em parte, uma reação à má legibilidade causada pelas limitas instruções de controle das linguagens das décadas de 50 e 60;
- Instruções de Controle que envolvem desvios (*goto*) dificultam e legibilidade. Em algumas LPs a instrução *goto* se ramifica para cima e são necessárias.
- Deve-se utilizar programação estruturada e eliminar comandos de desvios para aumentar e legibilidade:
  - Sub-rotinas (procedimentos e funções);
  - Métodos (orientação a objetos).

Legibilidade – Instruções de Controle


```
while (incr < 20) {
      while (sum <= 100) {
        sum += incr;
      }
      incr++;
}
```

```
loop1:
      if (incr >= 20) goto out;
loop2:
      if (sum > 100) goto next;
            sum += incr;
            goto loop2;
next:
      incr++;
      goto loop1;
out:
```

- Pode-se restringir o uso de "*goto*" para tornar o programa mais legível:
  - Devem preceder seus alvos, exceto quando usados para formar laços;
  - Seus alvos nunca devem estar muito distantes;
  - Número deve ser limitado (ou até nulo, se possível).

Legibilidade – Instruções de Controle

- No final da década de 60 as LPs projetadas passaram a ter instruções de controle suficientes. A necessidade do goto foi quase eliminada;
- O projeto de estrutura de controle de uma LP é agora um fator menos importante na legibilidade do que no passado.

while do...while repeat...until for...next

Legibilidade – Tipos de Dados e Estruturas

- Presença de facilidades adequadas para definir tipos de dados e estruturas de dados auxilia na legibilidade;
- LPs modernas possuem estrutura de dados já implementadas, permitindo utilizar pilhas, filas, listas e outras estruturas;
- Em outras LPs é preciso criar métodos para implementar tais estruturas.
- Suponha que em uma linguagem não exista um tipo de dado booleano e um tipo numérico seja usado para substituí-lo:
  - timeOut = 1 (significado não claro)
  - timeOut = true (significado claro)

Legibilidade – Considerações sobre a sintaxe

Palavras especiais ou reservadas

- Formas das palavras especiais de uma linguagem, tais como métodos para formar instruções compostas ou grupos de instruções (begin – end, abrir e fechar chaves{});
- Pascal exige pares begin/end para formar grupos em todas construções de controle (exceto repeat). A linguagem C usa chaves;
- FORTRAN 90 / ADA usam sintaxe distinta para cada tipo de grupo de instrução (if...end if / loop ... end loop)
- Palavras especiais de uma linguagem podem ser usadas como nomes de variáveis? DO e END no FORTRAN 90.

Legibilidade – Considerações sobre a sintaxe

Formas identificadoras

- Restringir os identificadores a tamanhos muito pequenos prejudica a legibilidade:
  - FORTRAN 77, tamanho máximo dos identificadores: 6 caracteres;
  - BASIC ANSI, uma letra ou uma letra e um número.

Capacidade de Escrita

- É a medida de quão facilmente uma linguagem pode ser utilizada para criar programas para um domínio de problema escolhido;
- A maioria das características da linguagem que afetam a legibilidade também afetam a Capacidade de Escrita;

Capacidade de Escrita – Simplicidade e ortogonalidade

- Poucos construtores, um pequeno número de primitivas, um pequeno conjunto de regras para combiná-los;
- Se uma LP contém um grande número de construções, alguns programadores não estarão familiarizados com todas;
  - Pode acarretar o uso incorreto de recursos.

Capacidade de Escrita – Suporte para abstração

- A capacidade de definir e de usar estruturas ou operações complexas de maneira que permita ignorar muitos dos detalhes;
- Exemplo:
  - Uso de subprogramas (algoritmo de ordenação);
- Tipos de Abstração:
  - Processo – algoritmos de classificação, elementos de interface gráfica;
  - Dados – tipo moeda, tipo string, tipo data.

Capacidade de Escrita - Expressividade

- Formas convenientes de especificar computações:
  - Uma expressão representa muitas computações;
  - Exemplo:
    - **i++** *no lugar* de i = i + 1;
    - **for** *no lugar* de while;
    - **ReadIn** do Pascal *no lugar* de realine do Java.

```
(Java)
BufferedReader teclado;
String linea;
teclado = new BufferedReader(
    new InputStreamReader(System.in) );
linea = teclado.readLine();
```

```
(Pascal)
linha: string[20]
readln(linha)
```

Confiabilidade

- Um programa é considerado confiável se ele se comportar de acordo com as suas especificações sob todas as condições;
- Quanto mais fácil é escrever um programa, maior a probabilidade de se estar correto;
- Pouca legibilidade ou pouca facilidade de escrita tendem a gerar programas poucos confiáveis, de difícil escrita e modificação;
- Verificado por meio de testes que devem ser realizados antes do *software* ser disponibilizado aos usuários.

Confiabilidade - Medidas

Verificação de tipos

- Testar se existem erros de tipos

Manipulação de Exceções

- Capacidade de interceptar erros em tempo de execução e por em prática medidas corretivas

Apelidos (Aliasing)

- Presença de dois ou mais métodos, ou nomes, distintos que referenciam a mesma célula de memória

Característica do Custo

- Para se determinar o custo final de uma LP deve-se considerar:
  - Treinamento – quanto maiores a complexidade e os recursos da LP, maior o grau de dificuldade de aprendizado;
  - Programação – fatores de simplicidade;
  - Testes – testes visam a confiança;
  - Implementação – LP que seja executada apenas em hardwares caros terá menos chance de se tornar popular;
  - Manutenção – quanto mais fácil é escrever um programa, mais fácil a sua manutenção;
  - Evolução – difícil prever a evolução da linguagem.

Custo

- Treinamento dos programadores para usar a linguagem;
- Escrita de programas na linguagem;
- Compilação programas na linguagem;
- Execução dos programas;
- Sistema de implementação da linguagem: existência de compiladores *free*;
- Confiabilidade baixa leva a altos custos;
- Manutenção dos programas;
- Evolução da linguagem.

Custo

- Custo de criação, teste e uso de programas:
  - O esforço gasto para resolver um problema através da implementação de uma aplicação deve ser minimizado;
  - A linguagem de programação deve prover ferramentas que facilitem estas tarefas:
    - (1) Ambiente gráfico para desenvolvimento;
    - (2) composição (componentes) ao invés de implementação;
    - (3) Ferramentas de debug;
    - (4) Automação de testes;
    - (5) Gerenciadores de versões, ...

Custo – Manutenção de programas

- O tempo gasto com a manutenção de *software* é maior do que o tempo gasto com o seu desenvolvimento.
- Manutenção inclui reparos de erros, mudanças nos requisitos originais, inserção de novos requisitos (novas demandas de mercado)
- Linguagens de programação devem facilitar a manutenção de *software*.

Custo

Sabendo que esses são funções da capacidade de escrita e da legibilidade.

Custo x Benefício no projeto da linguagem

- Confiabilidade *versus* Custo de Execução:
  - Exemplo: Java requer que todas as referências a vetores sejam checadas para garantir que os índices estejam dentro dos limites, mas isso aumenta o custo de execução.

- Readability *versus* writability:
  - Exemplo: A LP provê muitos operadores poderosos (e uma grande quantidade de novos símbolos), permitindo que computações complexas sejam escritas em programas compactos, porém isso dificulta a leitura.