



ENGENHARIA DE SOFTWARE

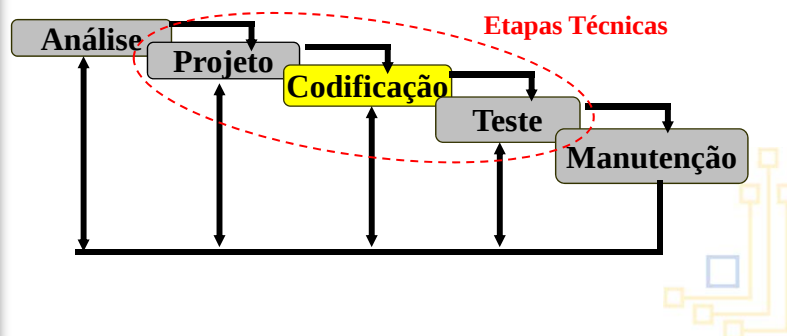
UNIDADE 5 – Codificação e Linguagens de Programação
(Aula 8 – Aspectos de Desenvolvimento de Programas)

Prof. Ivan Nunes da Silva

1. Codificação e Linguagens

● Função da Etapa de Codificação

- Tradução das representações do projeto para uma linguagem de programação, resultando em instruções executáveis pelo computador.



2. Programação de Computadores

● Classificação dos Softwares

– Software de Sistema (ou Sistema Operacional)

- Capaz de oferecer ao usuário, ou a outros softwares, facilidades de acesso aos recursos do computador.
- Administra os arquivos, controla periféricos e executa utilitários.

– Software Utilitário

- Programas desenvolvidos por especialistas ou mesmo por usuários experimentados.
- Tem por objetivo facilitar a realização de determinadas atividades correntes no uso dos computadores:
 - Detecção e eliminação de vírus, compactação de arquivos, otimização de espaço em disco, etc.

– Software Aplicativo

- Programas desenvolvidos ou adquiridos pelos usuários para algum fim específico.
- De natureza profissional, educacional ou mesmo de lazer (jogos).

3

2. Programação de Computadores

● Linguagem de Programação

– Definida pelos seguintes aspectos:

- Conjunto limitado de **instruções** (vocabulário).
- Associado a um conjunto de regras (**sintaxe**):
 - Define como as instruções podem ser associadas.
 - Como se pode compor os programas para a resolução de um determinado problema.

● Existem Diversas Linguagens de Programação

- Algumas de uso mais geral.
- Outras concebidas para áreas de aplicação específicas.

4

3. Níveis de Linguagens de Programação

● Classificação das Linguagens de Programação

- Podem ser classificadas em níveis de linguagens:
 - Sendo que os níveis mais baixos são mais próximos da linguagem interpretada pelo processador e mais distante das linguagens naturais.
- Principais Níveis:
 - Nível de Linguagem (Alto Nível).
 - Nível de Sistema Operacional.
 - Nível de Máquina.
 - Nível de Microprogramação.
 - Nível de Lógica Digital.



5

4. Linguagens de Alto Nível

● Linguagem de Alto Nível

- Assim denominadas por apresentarem uma sintaxe mais próxima da linguagem natural.
- Fazem uso de palavras reservadas extraídas do vocabulário corrente (como READ, WRITE, TYPE, etc.).
- Permitem a manipulação dos dados nas mais diversas formas:
 - números inteiros, reais, vetores, listas, etc.

● Origem e Exemplos

- Surgiram entre o final da década de 50 e início dos anos 60:
 - Fortran, Cobol, Algol e Basic.
- Algumas delas têm resistido ao tempo e às críticas:
 - Fortran ainda é visto como uma linguagem de implementação para muitas aplicações de engenharia.
 - Cobol foi uma linguagem bastante utilizada no desenvolvimento de aplicações comerciais.



6

4. Linguagens de Alto Nível

● Classificação Quanto ao Uso

– Linguagens de Uso Geral

- Podem ser utilizadas para implementação de programas com as mais diversas características e independente da área de aplicação.
- C, Pascal e Modula-2.

– Linguagens Especializadas

- São orientadas ao desenvolvimento de aplicações específicas.
- Prolog, Lisp e Forth.

– Linguagens Orientadas a Objetos

- Oferecem mecanismos sintáticos e semânticos de suporte aos conceitos da programação orientada a objetos.
- Smalltalk, Eiffel, C++, Visual Basic e Delphi.



7

5. Desenvolvimento de Programas

● Ambiente de Desenvolvimento

- Desenvolvimento de programas é associado ao uso de ferramentas ou ambientes de desenvolvimento;
 - que acompanham o programador desde a etapa de codificação propriamente dita até a geração e teste do código executável.
- Principais etapas de geração de um programa:
 - Codificação do código fonte.
 - Tradução do código fonte.
 - Linkagem.
 - Depuração.
- Engenharia de Software
 - Metodologias mais completas que definem os passos para o desenvolvimento de programas.

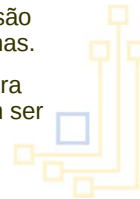


8

5. Desenvolvimento de Programas

● Tradução do Código Fonte (código objeto)

- Compreende três atividades: análise léxica, análise sintática e geração de código.
 - São processadas pelos módulos do tradutor: analisadores léxico, sintático e gerador de código.
- **Análise Léxica:**
 - É o processo de reconhecer quais cadeias de símbolos do programa-fonte representam entidades indivisíveis.
 - Exemplos:
 - Três símbolos **153** não devem ser interpretados como 1, seguido por 5, seguido por 3, mas são reconhecidos como um valor numérico apenas.
 - Palavras que aparecem no programa, embora compostas de caracteres individuais, devem ser interpretadas cada qual como uma unidade inseparável.



9

5. Desenvolvimento de Programas

● Tradução do Código Fonte (código objeto)

– Analisador Léxico

- Analisador léxico identifica um grupo de símbolos que representam uma única entidade:
 - Classifica como sendo um valor numérico, ou uma palavra, ou um operador aritmético, e assim por diante.
 - Gera um padrão de bits conhecido como átomo (*token*), indicativo da classe do elemento.
- Átomos são os dados de entrada do analisador sintático.



10

5. Desenvolvimento de Programas

● Tradução do Código Fonte (código objeto)

– Análise Sintática

- Processo de identificação da estrutura gramatical do programa, fazendo o reconhecimento do papel de cada um dos seus componentes.
- Processo de análise sintática é feito com base em um conjunto de regras sintáticas que definem a sintaxe da linguagem de programação.
- Uma forma de expressar regras sintáticas é através de diagramas de sintaxe.

11

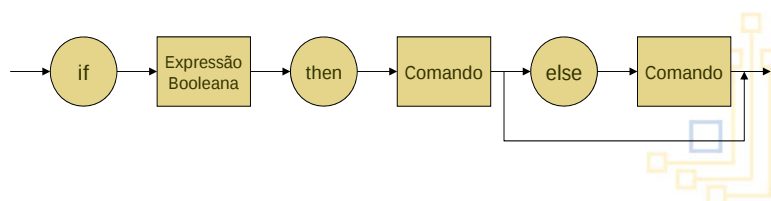


5. Desenvolvimento de Programas

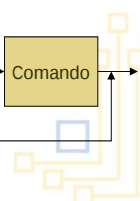
● Tradução do Código Fonte (código objeto)

– Diagramas de Sintaxe

- Representações gráficas da estrutura gramatical de um programa.
- Símbolos **Terminais** são identificados por elipses (Círculos).
- Símbolos **Não-Terminais** são identificados por retângulos.



12



5. Desenvolvimento de Programas

● Tradução do Código Fonte (código objeto)

- À medida que um analisador sintático recebe átomos do analisador léxico, ele vai analisando os comandos e ignorando os comentários.
- As informações extraídas das declarações são tabeladas em uma estrutura conhecida como **Tabela de Símbolos**:
 - A tabela guarda informações sobre as variáveis declaradas, os tipos de dados e as estruturas de dados associadas a tais variáveis.
- Analisador sintático utiliza como base estas informações ao analisar comandos tais como:
 - $\text{Total} := \text{Custo} + \text{Imposto}$



13

5. Desenvolvimento de Programas

● Depuradores ou *Debuggers*

- Auxilia o programador a eliminar (ou reduzir) a quantidade de “bugs” (erros) de execução no seu programa.
- Executam o programa gerado através de uma interface apropriada que possibilita uma análise efetiva do código do programa .
- Isto graças aos seguintes aspectos:
 - Execução passo-a-passo (ou instrução por instrução) de partes do programa.
 - Visualização do “estado” do programa através das variáveis e eventualmente dos conteúdos dos registros internos do processador.
 - Alteração em tempo de execução de conteúdos de memória ou de variáveis ou de instruções do programa.



14

6. Paradigmas de Programação

- **Existem vários paradigmas de programação:**

- Os paradigmas são os estilos utilizados pelos programadores para conceber um programa.
- Os mais conhecidos são:
 - Programação Não-estruturada.
 - Programação Procedural.
 - Programação Orientada a Objetos.



15

7. Paradigmas de Programação (Não Estruturada)

- **Programação Não-estruturada**

- Pessoas aprendem a programação escrevendo programas pequenos e simples:
 - Consistindo apenas de um programa principal.
 - Consistindo ainda de uma sequência de comandos ou declarações que modificam dados que são acessíveis a todos os pontos do programa.
- Técnica de programação não estruturada:
 - Têm várias desvantagens no caso de programas grandes:
 - Se a mesma sequência é necessária em localizações diferentes, a mesma deve então ser copiada.
 - Não permite a organização do programa.



16

7. Paradigmas de Programação (Não Estruturada)

```
PROGRAM MaiorN;  
VAR  
  Num1,Num2,Num3,Maior : integer;  
BEGIN  
  (* Leitura dos números *)  
  write('Entre com o primeiro número: '); readln(Num1);  
  write('Entre com o segundo número: '); readln(Num2);  
  write('Entre com o terceiro número: '); readln(Num3);  
  (* Determinação do maior numero *)  
  Maior := Num1;  
  IF Maior < Num2  
  THEN Maior := Num2;  
  IF Maior < Num3  
  THEN Maior := Num3;  
  (* Impressão do maior numero *)  
  writeln('Programa de Determinação do Maior Número');  
  writeln;  
  writeln('>> O maior número é: ',Maior);  
END.
```

17

8. Paradigmas de Programação (Procedural)

● Programação Procedural

- Programa é visto como uma sequência de chamadas de procedimentos:
- Uma chamada de procedimento é usado para invocar o procedimento:
 - Podendo ser passado alguns parâmetros.
- Após a sequência ser executada:
 - Controle do fluxo de execução retorna justo após o ponto de chamada do procedimento.
- Programas podem ser escritos mais estruturados e livres de erro:
 - Introduzindo parâmetros tão bem quanto procedimentos de procedimentos (sub-procedimentos):
 - Por exemplo, se um procedimento é correto, toda vez que ele é usado ele produz um resultado correto.
 - No caso de erros pode-se direcionar a busca para aqueles lugares que não são livres de erros.

18

8. Paradigmas de Programação (Procedural)

```
PROGRAM MaiorN1;
VAR
  Num1,Num2,Num3,Maior : Integer;

PROCEDURE LeNum(VAR N1: Integer; VAR N2: Integer; VAR N3:
  Integer);
BEGIN
  write('Entre com o primeiro numero -> '); readln(N1);
  write('Entre com o segundo numero -> '); readln(N2);
  write('Entre com o terceiro numero -> '); readln(N3);
END; (* LeNum *)

FUNCTION DetMaior(N1:Integer;N2:Integer;N3:Integer): Integer;
var M: Integer;
BEGIN
  M := Num1;
  IF M < Num2 THEN M := Num2;
  IF M < Num3 THEN M := Num3;
  DetMaior:=M;
END; (* DetMaior *)

BEGIN (* Programa Principal *)
  LeNum(Num1,Num2,Num3);
  Maior:=DetMaior(Num1,Num2,Num3);
  ImpMaior(Maior);
END.
```

19

9. Linguagens Interpretadas/Compiladas

● Linguagens de Programação Compiladas

- Etapas de desenvolvimento anteriores consideram o uso de linguagens de programação compiladas, ou seja:
 - Produzirão um programa na forma da linguagem de máquina do processador.
 - Instruções definidas pelo programador usando uma linguagem de alto nível serão traduzidas para as instruções na linguagem de máquina.

● Linguagens de Programação Interpretadas

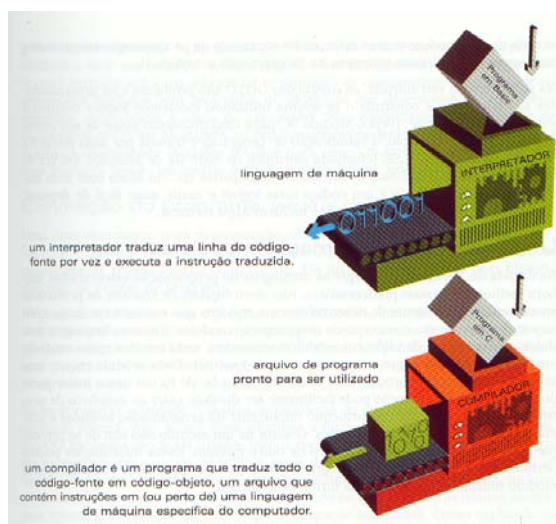
- Interpretação do programa usando outro programa, chamado de interpretador.
- Linguagem interpretada mais conhecida: a linguagem Java.
 - Interpretada por uma máquina virtual chamada JVM (*Java Virtual Machine*).

● Interpretador

- Programa que executa as instruções escritas em linguagem de alto nível.
 - Geralmente translada as instruções de alto nível em uma forma intermediária que é executada.

20

9. Linguagens Interpretadas/Compiladas



21

9. Linguagens Interpretadas/Compiladas

● Compilador Versus Interpretador

- Programa compilado executa mais rapidamente que um programa interpretado.
- Vantagem do interpretador é que ele não necessita passar por um estágio de compilação durante no qual as instruções de máquina são gerados:
 - Processo de tradução pode consumir muito tempo se o programa é longo.
 - Interpretador pode executar imediatamente os programas de alto-nível.
- Interpretadores são algumas vezes usados durante o desenvolvimento de um programa:
 - Isto ocorre quando um programador deseja testar rapidamente seu programa.
 - Exemplo: Uso do Matlab para prototipação rápida.

22

10. Linguagens Interpretadas/Compiladas

● Compilador Versus Interpretador

- Tanto os interpretadores como os compiladores são disponíveis para muitas linguagens de alto nível.
 - Java, Basic, LISP e Matlab são especialmente projetadas para serem executadas por um interpretador.
- Linguagens interpretadas podem possibilitar a portabilidade.
 - Não é gerado um código de máquina, e sim um código intermediário que será interpretado por uma máquina virtual.
 - Pode-se obter a portabilidade do código se esta máquina virtual for desenvolvida para várias plataformas:
 - Este é o caso da linguagem Java.



23

9. Linguagens Interpretadas/Compiladas

● Máquina Virtual

- É um ambiente operacional:
 - Ambiente onde os usuários executam o programa.
 - Este ambiente é autocontido que se comporta como se fosse um computador separado.
- **Exemplo: applet Java** executa em uma Máquina Virtual Java que não acessa o sistema operacional do computador hospedeiro.
- Este projeto tem duas vantagens:
 - Independência de sistema:
 - Uma aplicação poderá ser executada em qualquer máquina virtual, sem se preocupar com o hardware e software, dando então suporte ao sistema.
 - Segurança:
 - Como a máquina virtual não tem contato com o sistema operacional, existem poucas possibilidades de um programa interpretado danificar outros arquivos ou aplicações.
 - Esta vantagem traz consigo uma limitação: os programas executando em uma máquina virtual não pode tomar vantagem das funcionalidades do sistema operacional.



24