Modelos de Linguagens

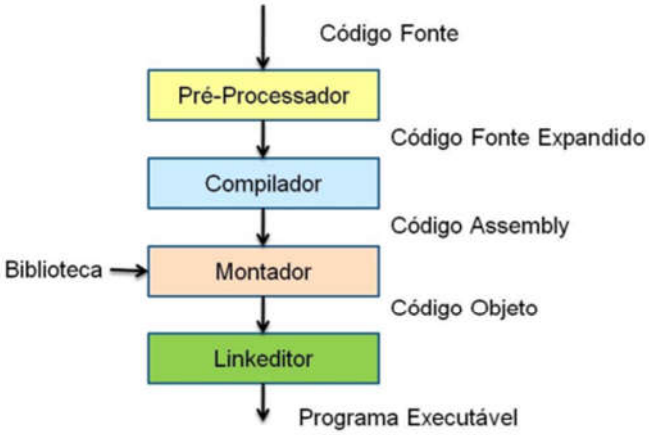
- Modelo imperativo
  - Linguagens expressam sequências de comandos que realizam transformações sobre dados
  - base: máquina de von Neumann
  - orientadas a procedimentos
- Modelo declarativo
  - Linguagens que não possuem o conceito de sequências de comandos
  - Linguagens funcionais, baseadas em funções
  - Linguagens lógicas, baseada em axiomas lógicos

Modelos de execução

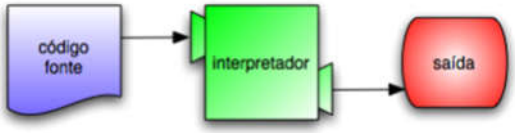
- Sequencial
  - Uma computação é realizada após o término da anterior
  - Controle de fluxo de execução interno ao programa:
    - Sequência
    - Seleção
    - Iteração
    - Invocações
- Concorrente
  - Múltiplas computações podem ser executadas simultaneamente
  - Computações paralelas
    - Múltiplos processadores compartilham memória
  - Computações distribuídas
    - múltiplos computadores conectados por uma rede de comunicação

Modelo de Execução/Construção

Compilada



Interpretada

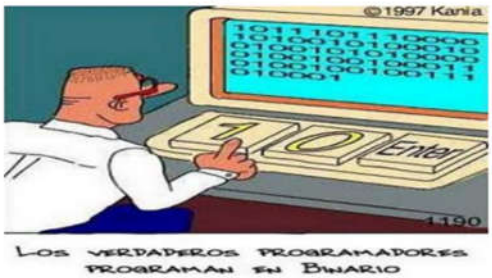


Modelo de Nível

Baixo Nível

- Próximo à Maquina

```
MOV AX,0002
MOV BX,0001
ADD AX,BX
```



Alto Nível

- Próximo ao programador

```
int A, B;
int main()
{
      A = 2;
      B = 1;
      A = A + B;
}
```

FIM…CONTINUA NA PRÓXIMA AULA. POR: COMPILADO NELSON ALI COM RECURSO A FONTES ABERTAS NA INTERNET.

Modelo de Aplicação

- Científica
  - Algol, Fortran, Matlab
- Comerciais
  - Cobol, Dataflex
- Processamento de Listas
  - Lisp
- Gerais
  - Java, C, C++, Visual Basic
- Inteligência Artificial
  - PROLOG
- Web
  - JSP, PHP, HTML

Critérios para avaliação de LP

- Legibilidade
- Simplicidade
- Expressividade
- Ortogonalidade
- Confiabilidade
- Portabilidade

Legibilidade

- Facilidade de ler e escrever programas
- Legibilidade influi:
  - desenvolvimento e depuração de programas
  - manutenção de programas
  - desempenho de equipes de programação
- Fatores que melhoram a legibilidade:
  - Abstração de dados
  - Comandos de controle
  - Modularização de programas
  - Documentação
  - Convenções léxicas, sintaxe e semântica
    - Exemplo em Java: nomes de classes iniciam por letra maiúscula nomes de campos usam letras minúsculas

Simplicidade

- Representação de cada conceito seja simples de aprender e dominar:
  - Simplicidade sintática exige que a representação seja feita de modo preciso, sem ambiguidades
    - Contraexemplo 1: A++; A=A+1; A+=1; ++A.
    - Contraexemplo 2: a, b := x+1, y + 3;
  - Simplicidade semântica exige que a representação possua um significado independente de contexto
    - Contraexemplo: private: B b; class Campo: private Campo
  - Simplicidade não significa concisão
    - A linguagem pode ser concisa mas usar muitos símbolos especiais: Ex. Linguagens Funcionais

Expressividade

- Representação clara e simples de dados e procedimentos a serem executados pelo programa
  - Exemplo: tipos de dados em Pascal
- Expressividade x concisão
  - Muito concisa: falta expressividade?
  - Muito extensa: falta simplicidade?
- Linguagens mais modernas
  - Incorporam apenas um conjunto básico de representações de tipos de dados e comandos
  - Aumentam o poder de expressividade com bibliotecas de componentes
    - Ex. BigInteger em Java

Ortogonalidade

- conjunto de construções primitivas pode se combinar em um número grande de maneiras para construir as estruturas de uma linguagem
- Possibilidade de combinar entre si, sem restrições, os componentes básicos da LP
  - Exemplo: permitir combinações de estruturas de dados, como arrays de qualquer tipo.
  - Contraexemplo: não permitir que um array seja usado como parâmetro de uma função
- Componente de primeira ordem: pode ser livremente usado em expressões, atribuições, como argumento e retorno de procedimentos

Portabilidade

- Multiplataforma: capacidade de um software rodar em diferentes plataformas sem a necessidade de maiores adaptações
  - Sem exigências especiais de hardware/software
  - Exemplo: aplicação compatível com Linux e Windows
- Longevidade:
  - ciclo de vida útil do software e o do hardware não precisam ser síncronos; ou seja, é possível usar o mesmo software após uma mudança de hardware

Confiabilidade

- Mecanismos que facilitem a produção de programas que atendam às sua especificações
  - Tipagem forte: o processador da linguagem deve:
    - assegurar que a utilização dos diferentes tipos de dados seja compatível com a sua definição
    - evitar que operações perigosas, tal como aritmética de ponteiros, seja permitida
  - Tratamento de exceções: sistemas de tratamento de exceções permitem construir programas que:
    - possuam definições de como proceder em caso de comportamento não usual
    - possibilitem tanto o diagnóstico quanto o tratamento de erros em tempo de execução

Critérios para escolha de uma LP

- Implementação
  - Disponibilidade quanto a plataforma;
  - Eficiência;
- Competência na LP
  - Experiência do programador;
  - Competência do grupo envolvido;
- Portabilidade
  - Execução em várias plataformas
- Sintaxe
  - Certos tipos de problemas acomodam-se melhor a certos modelos;
- Semântica
  - Aplicação X facilidade;
- Ambiente de Programação
  - Ferramentas diminuem o esforço;
  - Uso de bibliotecas
- Modelo de computação

Estudo de Caso

Histórico das Linguagens de Programação

- Década de **1960**
  - Programação baseada na arquitetura do hardware existente
    - Computadores caros e inacessíveis
    - Seqüência de instruções
    - Sem preocupação com reutilização de código
  - Exemplos de linguagens populares:
    - Uso acadêmico: ALGOL
    - Uso científico: FORTRAN
    - Uso comercial: COBOL

Histórico das Linguagens de Programação

- Década de **1970**: surgimento da Engenharia de Software
  - Abstração de dados: definição de tipos
  - Abstração de controle: comandos, procedimentos
  - Inicia preocupação com *"programming in the large"*:
    - módulos e programação Estruturada
- Exemplos de linguagens populares:
  - Uso acadêmico: Pascal (tipos de dados)
  - Uso comercial: COBOL Estruturado (arquivos/relatórios)

Histórico das Linguagens de Programação

- Década de **1980**: modularização
  - Ênfase em mecanismos de LP e abstrações
  - Correção de programas: verificação de tipos, exceções
  - Programação concorrente e distribuída e tempo real
  - Programação baseada em objetos (implementação de Tipos Abstratos de Dados)
  - Programação orientada a objetos (herança)
- Exemplos de linguagens
  - Uso acadêmico: Pascal / Modula / C
  - Programação de tempo real: Ada 83
  - Orientada a objetos: SmallTalk

Histórico das Linguagens de Programação

- Década de **1990**: base na Arquitetura
  - Estruturação de dados: encapsulamento
  - Estruturação da computação: classe
  - Estruturação do programa: classes e objetos
  - Programação para Internet: plataforma neutra
- Exemplos de linguagens
  - Object-Pascal / Delphi
  - C / C++ / Objective-C
  - Ada83 / Ada95
  - Java

Histórico das Linguagens de Programação

- Atualmente
  - Mecanismos alternativos de modularidade
    - programação orientada a aspectos
  - RAD (Rapid Application Development)
  - Maior ênfase na distribuição e mobilidade
  - Integração com bases de dados, incluindo XML e bancos de dados relacionais.
  - Uso extensivo de frameworks
    - Spring, Hibernate, EJB
  - Crescimento das linguagens voltadas para Internet

Exercício (Entrega até o final da aula)

Nome: _____ Data: _____

- 1 - Por que é importante estudar os paradigmas de linguagens de programação?
- 2 - Qual sua linguagem de programação preferida? Por que?

UNIVERSIDADE JOAQUIM CHISSANO
FACULDADE DE CIÊNCIAS E TECNOLOGIA
CURSO DE ENGENHARIA EM TECNOLOGIAS E SISTEMAS DE INFORMAÇÃO

RESUMO EXPOSITIVO DOS CONCEITOS SOBRE PARADIGMAS DE PROGRAMAÇÃO

1. O que é paradigma de programação?

Um **paradigma** é um estilo de programação, uma metodologia. Não se trata de uma linguagem, mas a forma como você soluciona problemas usando determinado código. Existem muitas linguagens de programação conhecidas, mas todas elas precisam seguir algumas regras quando implementadas. E essas regras são os paradigmas.

Para exemplificar, imagine a seguinte situação: você precisa ir da sua casa ao trabalho. O problema é a locomoção e a forma como você pode ir (a pé, de bicicleta, de carro ou transporte público) será o paradigma. Logo, a denominação paradigma é a maneira como você resolve uma questão.

Todas as formas (paradigmas) envolvem um esforço e demandam tempo, alguns mais (ir a pé ou de bicicleta), outros menos (ir de transporte público ou de carro), podendo ser algo simples e rápido de resolver ou mais trabalhoso e moroso. Assim é na programação: você possui diversas soluções, basta escolher a que resolverá o problema da melhor forma, preferencialmente da maneira mais rápida, fácil e eficiente.

Mas, para escolher bem, é preciso antes conhecer as alternativas. Desta forma, quando uma nova linguagem de programação é desenvolvida, conforme suas peculiaridades, ela tende a se enquadrar em um paradigma.

2. Qual a diferença de paradigmas de programação e linguagem de programação?

Em alguns casos, as pessoas confundem linguagem com paradigmas de programação devido às similaridades, mas, eles não tratam do mesma concenso (contextualidade).

Os **paradigmas** são modelagens de escrita de código que podem ser aplicados a várias linguagens. É possível ainda usar mais de um paradigma a uma mesma solução em uma linguagem previamente escolhida.

A **linguagem de programação** é o meio que os humanos usam para instruir e comunicar os computadores a fazer diferentes tarefas e ações. Elas também possuem seus próprios vocabulários e regras gramaticais para desenvolver essas instruções.

3. Porquê é importante aprender sobre os paradigmas de programação?

Cada problema pode ter mais de uma solução, logo existem diversos paradigmas para resolvê-los. Por isso, um paradigma pode ser mais vantajoso do que outro, conforme o desenvolvimento de determinado sistema, oferecendo técnicas apropriadas para uma aplicação específica.

Ao escolher o paradigma de desenvolvimento adequado ao projeto, é possível que sejam desenvolvidas aplicações com maior produtividade, possibilitando a singularidade na orientação de escrita do código entre a equipe, tornando-o mais legível e facilitando a manutenção ao longo de sua existência.

Além disso, manter o mesmo paradigma determinará o objetivo do devenvolvidor sobre a composição da estrutura e execução do sistema. Isso permite que as técnicas adequadas sejam utilizadas. E é essencial manter essa metodologia durante todo o trabalho.

Por isso é importante entender sobre os diferentes tipos de paradigmas, pois fará com que haja uma ligação entre o você e o projeto - seja ainda mais profissional. Saber qual tipo de paradigma a escolher na resolução de um problema tornará seu raciocínio diferenciado. Dessa forma, antes de refletir sobre a solução de um problema, devemos pensar na modelagem dessa solução e sobre o paradigma a ser utilizado.

4. Quais são os principais paradigmas de programação?

Basicamente, existem **seis** (6) principais tipos de paradigmas. Cada um deles criado para cumprir diferentes finalidades no desenvolvimento web e, portanto, com prós e contras distintos, conforme a aplicação. Dependendo da linguagem de programação utilizada, o desenvolvedor pode usar mais de um.

"Os principais paradigmas de programação, basicamente, pertencem a dois grupos: imperativos ou declarativos."

5. Paradigma Imperativo ou Procedural

No **paradigma imperativo**, também chamado de **procedural**, o foco da execução ou da solução de um problema está em como ele deve ser feito. Nesse tipo de construção, as instruções devem ser passadas ao computador na sequência em que devem ser executadas, onde o programador ou programadora descreve um passo a passo detalhado do que deve ser cumprido pela máquina.

Nesse caso, a solução do problema será muito dependente da experiência e criatividade de quem trabalha com a programação. O foco da resolução estará em "como" deve ser feito.

Esse paradigma tem a vantagem de ser eficiente e de permitir uma modelagem tal qual o mundo real, além de ser bem estabelecido e flexível. Por outro lado, o código-fonte gerado é de difícil legibilidade.

Por se tratar de um paradigma relativamente complexo, não é indicado para a construção de aplicações que necessitam de manutenção no curto prazo ou mudanças muito frequentes. Vários tipos de linguagem de programação suportam esse tipo de paradigma, sendo Cobol, Pascal e Fortran as principais.

6. **Paradigma Orientado a Objetos**

O **paradigma orientado a objetos** é o mais popular devido aos seus benefícios, como a modularidade do código e a capacidade de associar diretamente problemas reais em termos de código. Com o objetivo de facilitar o desenvolvimento de aplicações web, este foi o primeiro paradigma a permitir a programação multiplataforma.

Com ele, não é necessário criar uma mesma aplicação de maneiras diferentes de acordo com o sistema operacional onde ele rodará: os sites, aplicativos e softwares são desenvolvidos uma única vez e são interpretados por diferentes plataformas sem obstáculos. Neste caso, o programa é escrito como uma coleção de classes e objetos para uma boa comunicação. A entidade menor e básica é objeto e todo tipo de cálculo é realizado apenas nos objetos.

As principais linguagens de programação que utilizam este paradigma são Python, C++ e Java, PHP e Ruby. É indicado a sua utilização quando vários programadores atuam juntos e não precisam entender tudo sobre cada componente ou quando são previstas muitas mudanças no projeto.

7. **Paradigma Orientado a Eventos**

Os **paradigmas orientados a eventos** são usados por toda linguagem de programação que tem uso de recursos gráficos, como jogos e formulários, e depende de uma ação prévia do usuário para efetuar um movimento.

Dessa forma, a execução do programa se dá à medida que determinados eventos são disparados. Portanto, quem usa é responsável pelo momento em que o programa é executado.

As principais linguagens de programação que utilizam este paradigma são: Visual Basic e Delphi.

8. **Paradigma Funcional**

O **paradigma funcional** é aquele que destaca o uso das funções onde o problema é dividido em blocos e, para sua resolução, são implementadas atribuições que definem variáveis em seu escopo que podem ou não retornar resultados.

É indicado quando a solução requerida depende de uma base matemática. Assim, subdivide-se o problema proposto e as funções implementadas farão os cálculos matemáticos. Ao final, o programador ou programadora deve integrar a solução entregue.

As principais linguagens de programação que utilizam este paradigma são Haskell, Scheme e LISP.

9. **Paradigma Declarativo**

O **paradigma declarativo**, ao contrário do imperativo, está mais focado em "no que" deve ser resolvido do que em "como" isso será feito.

Ele leva este nome porque, ao utilizá-lo, o programador ou programadora declara verdades lógicas imutáveis para as quais os resultados serão sempre os mesmos após suas interações.

O nível de abstração aqui é muito maior e as principais linguagens declarativas são também de marcação: HTML, XML, XSLT e XAML.

10. **Paradigma Lógico**

O **paradigma lógico**, também conhecido como restritivo, é um pouco distinto dos demais e deriva do declarativo. Muito popular no setor de Inteligência Artificial por obter resultados através da análise lógico-matemática, ele utiliza formas de lógica simbólica como padrões de entrada e saída. A partir daí, realiza inferências para produzir os resultados. Os principais elementos deste paradigma são: proposições, regras de inferência e busca.

Dentre as linguagens de programação que utilizam esse paradigma, podemos citar QLISP, Mercury e Prolog.

Além dos paradigmas citados, a computação paralela é uma forma de resolução de problemas onde vários computadores trabalham simultaneamente para chegar a um mesmo objetivo, permitindo que muitos processadores executem um programa em menos tempo, dividindo-os. Essa solução, muitas vezes, exige um esforço de trabalho maior, por isso podem ser aplicações mais robustas e que sirvam a muitos usuários.

Essa abordagem é recomendada, geralmente, quando você tem um sistema que possui mais de uma CPU ou processadores multinúcleos ou precisa resolver problemas computacionais que podem levar até dias para serem resolvidos. As linguagens que suportam a abordagem de processamento paralelo são C e C++.

Pelo que você pôde ler até aqui, os paradigmas de programação são uma ótima fonte de conhecimento adicional. Certamente, conhecê-los bem e dominar sua aplicabilidade coloca quem trabalha com desenvolvimento de softwares em um nível acima da média.