# Linguagens de Programação
# Conceitos e Técnicas

Introdução

Prof.

# Razões para Estudar LPs

- Maior capacidade de desenvolver soluções computacionais para problemas
- Maior habilidade ao usar uma LP
- Maior capacidade para escolher LPs apropriadas
- Maior habilidade para aprender novas LPs
- Maior habilidade para projetar novas LPs

# Papel de LPs no PDS

- O objetivo de LPs é tornar mais efetivo o processo de desenvolvimento de software (PDS)
- PDS visa geração e manutenção de software de modo produtivo e garantia de padrões de qualidade

# Papel de LPs no PDS

- Principais Propriedades Desejadas em um Software
  - Confiabilidade
  - Manutenibilidade
  - Eficiência

# Papel de LPs no PDS

- Etapas do PDS
  - Especificação de Requisitos
  - Projeto do Software
  - Implementação
  - Validação
  - Manutenção

# Propriedades Desejáveis em LPs

- Legibilidade
  - Marcadores de Blocos
  - *C*

```
if (x>1)
    if (x==2)
        x=3;
else
    x=4;
```

  - *Lua*

```
if x>1
    if x==2
        x=3
    end
else
    x=4
end
```

# Propriedades Desejáveis em LPs

- identação de Blocos
  - *Python*

```
if x>1:
    if x==2:
        x=3
else:
    x=4
```

# Propriedades Desejáveis em LPs

- Desvios Incondicionais (goto)

```
int x = 100, y = 15;
rotulo1: x += 3;
if (x < 120) {
    rotulo2: x += 5;
    goto rotulo1;
}else{
    if (y > 20) {
        x -= 90;
    }else{
        y += 10;
        goto rotulo2;
    }
}
```

- Duplicação de Significado de Vocábulos

```
*p = (*p)*q;
```

# Propriedades Desejáveis em LPs

- Efeitos Colaterais

```
int x = 1;
int retornaCinco() {
   x = x + 3;
  return 5;
}
int main() {
   int y;
   y = retornaCinco ();
   y = y + x;
}
```

# Propriedades Desejáveis em LPs

- Redigibilidade
  - Tipos de Dados Limitados
    - FORTRAN (ausência de estruturas)
    - C, C++, Java (ausência de listas)
    - Python

```
print "Produtos:"
lista = ["café", "leite", "macarrão", "fubá"]
for produto in lista:
    print ">" + produto
```

# Propriedades Desejáveis em LPs

- Redigibilidade
  - Ausência de Conversão Implícita de Tipos (ADA)

```
um_float : Float := 10.0;
um_inteiro : Integer := 60;
outro_float : Float;
begin
    outro_float := um_float + Float (um_inteiro);
```

  - Conflito Ocasional com Legibilidade

```
void f(char *q, char *p) {
    for  (;*q=*p; q++,p++);
}
```

# Propriedades Desejáveis em LPs

- Confiabilidade
  - Existência de Tipo Booleano

```
boolean u = true;
int v = 0;
while (u && v < 9) {
   v = u + 2;
   if  (v == 6) u = false;
}
```

  - Tratamento de Exceções

```
try {
      System.out.println(a[i]);
} catch (IndexOutofBoundsException) {
      System.out.println("Erro de Indexação");
}
```

# Propriedades Desejáveis em LPs

- Confiabilidade
  - Tipagem Dinâmica

```
minha_variavel = 10
while minha_variavel > 0:
        i = f(minha_variavel)
        if i < 100:
                minha_variavel++
        else
                minha_varaivel = (minha_variavel + i) / 10
```

# Propriedades Desejáveis em LPs

- Eficiência
  - Verificação Dinâmica de Tipos

```
if x>1:
    x = "erro"
else:
        x = 4
x  /= 10
```

# Propriedades Desejáveis em LPs

- Facilidade de Aprendizado
  - Uso extensivo de conceitos de nível implementacional (ponteiros e desvios)
    - C
      - ponteiros – aritmética de ponteiros

        *(p + 5)
  - Múltiplas Formas é Prejudicial

    c = c + 1;

    c+=1;

    c++;

    ++c;

# Propriedades Desejáveis em LPs

- Ortogonalidade

```
int x, y = 2, z = 3;
byte a, b = 2, c = 3;
x = y + z;
a = b + c;
```

- Modificabilidade

```
const float pi = 3.14;
const int tam_max = 100;
float v[tam_max];
```

# Propriedades Desejáveis em LPs

- Reusabilidade

```
void troca (int *x, int *y) {
    int z = *x;
      *x = *y;
    *y = z;
}
int main() {
       int a = 10, b = 20, c = 30, d = 40;
      troca (&a, &b);
      troca (&c, &d);
}
```

# Propriedades Desejáveis em LPs

- Portabilidade
  - Fonte X Objeto
  - Padronização
    - ISO, IEEE, ANSI, NIST
  - Rigor no Projeto
    - int em C e Java
  - Pode Contrastar com Eficiência
  - Compatibilidade de Versões

# Especificação de LPs

- Léxico x Sintaxe x Semântica

  a = b;

- Sintaxe

  *<expressão> ::= <valor>|<valor><operador><expressão>*

  *<valor> ::= <número>|<sinal><número>*

  *<número> ::= <semsinal>|<semsinal>.<semsinal>*

  *<semsinal> ::= <dígito>|<dígito><semsinal>*

  *<dígito> ::= 0|1|2|3|4|5|6|7|8|9*

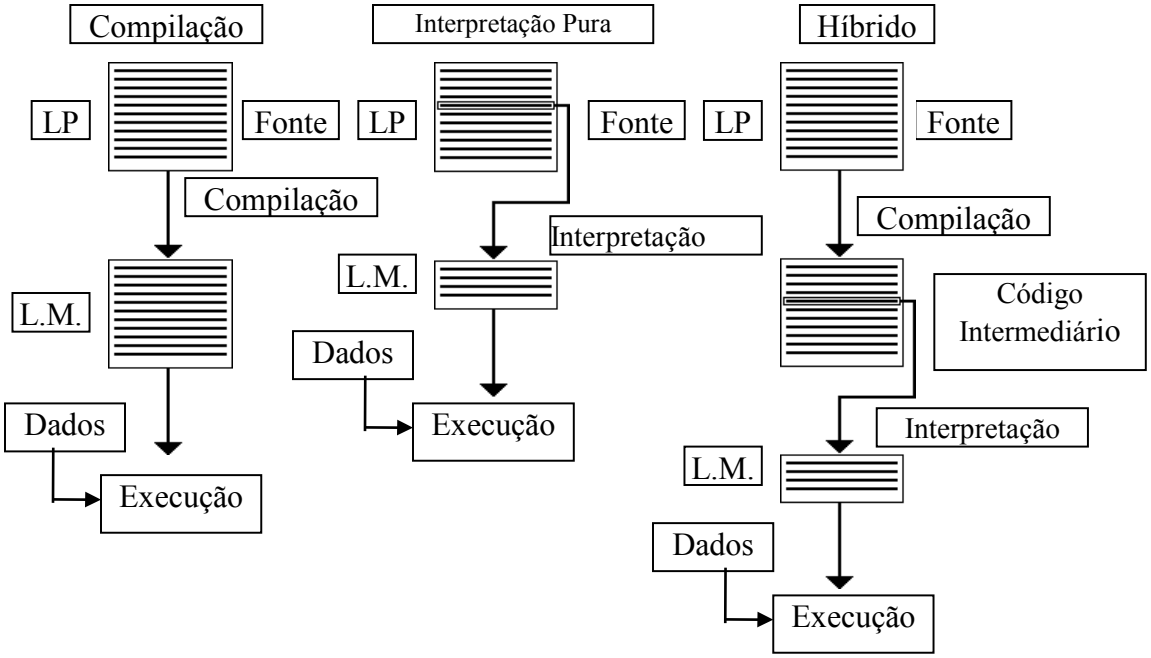
  *<sinal> ::= + | –*

  *<operador> ::= +|–|/|**

# Especificação de LPs

- Semântica
  - Enfoque Informal
    - Descrição textual - mais comum e problemática
  - Enfoque Operacional
    - Tradução para linguagem mais básica (assembler)
  - Enfoque denotacional
    - Descrição formal – mais rigorosa e menos comum

# Implementação de LPs



# Implementação de LPs

- Teoricamente, qualquer LP pode ser interpretada ou compilada
- Nenhuma é completamente interpretada ou compilada
  - Mesmo código de máquina é implementado em parte via microcódigo, sendo portanto também interpretado
  - LPs interpretadas são compiladas para algum nível de representação intermediária
    - bytecode, árvore de sintaxe abstrata
- Forma mais comum de implementação (ou existência de um passo explícito de compilação) acaba sendo associada a LP
  - Compiladas: C, C++, Java
  - Interpretadas: Perl, PHP, Ruby, Python, Lua

# Implementação de LPs

- Compilação
  - Eficiência de Execução
  - Sem Portabilidade
  - Falta de Acesso ao Fonte na Depuração
- Interpretação Pura
  - Flexibilidade, Portabilidade e Facilidade para Prototipação e Depuração
  - Redução na Eficiência de Execução
  - Raramente Usada – shell scripts (bash)
- Híbrido
  - Une Vantagens dos Outros Métodos
  - JVM

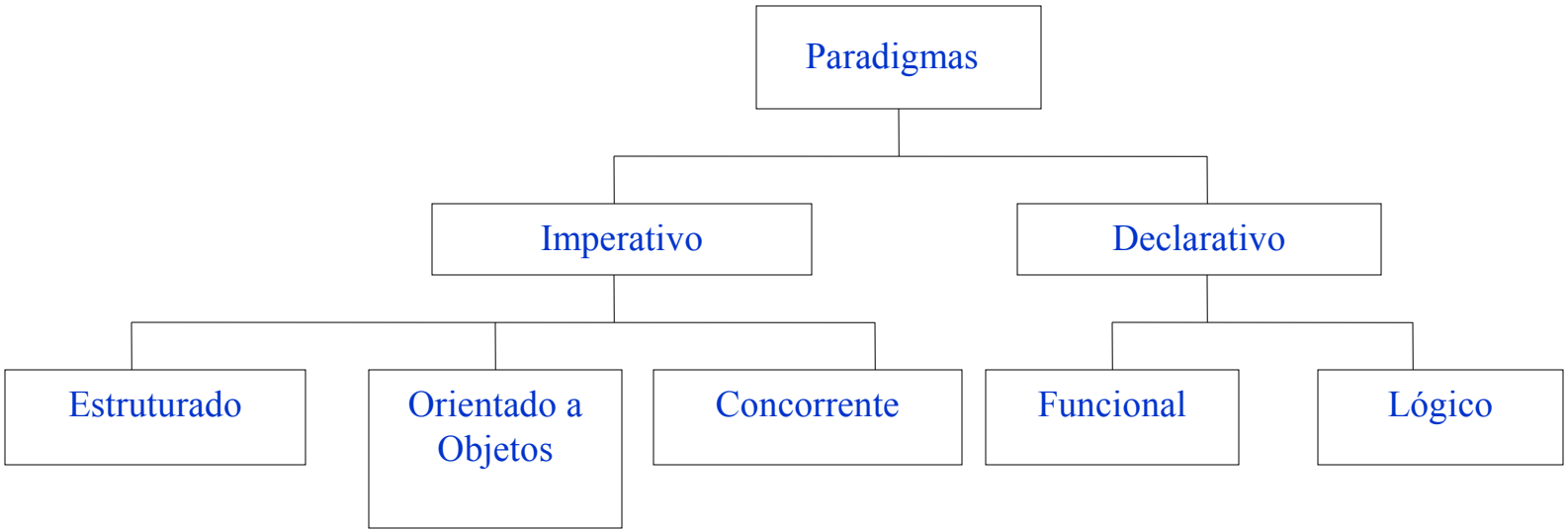
# Implementação de LPs

- Modos de Compilação
  - Para linguagem de máquina (gcc)
  - Para bytecode (javac)
  - Dois passos: para bytecode (javac) e de bytecode para linguagem de máquina, antes da execução (java jit)
  - GNU gcj possibilita os três modos
- Compilação para bytecode
  - Java: um único passo
  - Perl e Ruby: sempre antes de executar
  - Python: sempre que uma modificação no fonte é detectada antes da execução

# Implementação de LPs

- Compilação Dinâmica
  - Compila o código durante a execução (Common Lisp)
    - Código pode ser gerado dinamicamente e compilado

# Paradigmas de LPs



# Paradigmas de LPs

- Imperativo
  - Processo de Mudanças de Estados
  - Variável, Valor e Atribuição
  - Células de Memória
- Estruturado
  - Refinamentos Sucessivos
  - Blocos Aninhados de Comandos
  - Desestímulo ao uso de desvio incondicional

# Paradigmas de LPs

- Orientado a Objetos
  - Abstração de Dados
- Concorrente
  - Processos Executam Simultaneamente e Concorrem por Recursos

# Paradigmas de LPs

- Declarativo
  - Especificações sobre a Tarefa a Ser Realizada
  - Abstrai-se de Como o Computador é Implementado
- Funcional
  - Programa Composto por Funções
- Lógico
  - Predicados
  - Dedução Automática

# Evolução de LPs

- Dificuldade de Programação em Linguagens de Máquina
- Foco de Primeiras LPs era Eficiência de Processamento e Consumo de Memória
- Baixa Produtividade de Programação
  - Programação Estruturada
  - Tipos Abstratos de Dados
  - Orientação a Objetos
  - Linguagens Script

# Origem de LPs

- FORTRAN (1957 - 2008)
  - aplicações numéricas
- LISP (1959)
  - programação funcional
  - inteligência artificial
  - LP programável
- ALGOL (1958 – 1960 - 1968)
  - programação estruturada

# Origem de LPs

- COBOL (1960-2002)
  - aplicações comerciais
- BASIC (1964)
  - ensino para leigos
- PASCAL (1971)
  - ensino de programação estruturada
  - simplicidade
- C (1972 – 1999)
  - implementação de UNIX

# Origem de LPs

- PROLOG (1972)
  - programação lógica
- SMALLTALK (1972 - 1980)
  - programação orientada a objetos
- ADA (1983 – 1995 - 2005)
  - programação concorrente
- C++ (1985 – 1998, 2003)
  - disseminação da programação orientada a objetos

# Origem de LPs

- PERL (1987-)
  - script (unix) – usada tb para web
  - facilidades para processamento de textos
- Haskell (1990 – 1998 – 2010)
  - programação funcional
  - prova de correção de programas
  - elegância

```
qsort [] = []
qsort (x:xs) = qsort smaller ++ [x] ++ qsort bigger
     where smaller = filter (<x)  xs
                bigger  = filter (>=x) xs
```

# Origem de LPs

- Python (1990 – 2000 - 2008)
  - multi-paradigma (estruturado, oo, funcional)
  - aplicações independentes e scripts (web)
  - ambiente interpretador
  - tipagem dinâmica e forte
- Lua (1993-)
  - leve (interpretador = 150 Kb)
  - script (web)
  - brasileira – Tecgraf – Puc-Rio - Petrobras
  - jogos

# Origem de LPs

- Ruby (1995-)
  - multi-paradigma (estruturado – oo – funcional)
  - linguagem script mais poderosa que Perl e mais oo que python – LP certinha
  - sem especificação – várias implementações
  - ruby on rails – framework web famoso
- JAVA (1995 - 2010)
  - mais simples e confiável que C++
  - Internet
  - Software livre desde 2006
  - Evolução frequente - JSE 6 (Julho de 2010)

# Origem de LPs

- PHP (1995 -)
  - script (web) – Personal Home Pages - dinâmicas
  - Sintaxe baseada em C