Paradigmas de Programação

Fabrício Olivetti de França

07 de Junho de 2018

Paradigmas de Programação

Paradigmas de Programação

Definição: estilo de programação, a forma como você descreve a solução computacional de um problema.

Paradigmas de Programação

Em muitos cursos de Computação e Engenharia iniciam com paradigma imperativo e estruturado (vide Processamento da Informação e Programação Estruturada).

Exemplo clássico da receita de bolo (que não é a melhor forma de descrever o conceito de algoritmo).

Paradigmas de Programação

Alguns dos paradigmas mais populares:

- **Imperativo:** um programa é uma sequência de comandos que alteram o estado atual do sistema até atingir um estado final.
- **Estruturado:** programas com uma estrutura de fluxo de controle e uso de procedimento e funções.
- **Orientado a objeto:** organização através de objetos que contém dados, estados próprios e métodos que alteram ou recuperam os dados / estados. Os objetos se comunicam entre si para compor a lógica do programa.

Paradigmas de Programação

- **Declarativo:** especifica o que você quer, mas sem detalhar como fazer.
- **Funcional:** programas são avaliações de funções matemáticas sem alterar estados e com dados imutáveis.
- **Lógico:** especifica-se um conjunto de fatos e regras, o interpretador infere respostas para perguntas sobre o programa.

Paradigmas de Programação

Muitas das linguagens de programação são, na realidade, multi-paradigmas, porém favorecendo um dos paradigmas.

Paradigma Imperativo

- Descrevo passo a passo o que deve ser feito.
- Infame goto.
- Evoluiu para o procedural e estruturado com if, while, for.

Paradigma Imperativo

```
    aprovados = {};
    i = 0;
inicio:
    n = length(alunos);
    if (i >= n) goto fim;
    a = alunos[i];
    if (a.nota < 5) goto proximo;
    nome = toUpper(a.nome);
    adiciona(aprovados, nome);
proximo:
    i = i + 1;
    goto inicio;
fim:
    return sort(aprovados);
```

Paradigma Imperativo

😄 O fluxo de controle é explícito e completo, nada está escondido.

😄 Linguagem um pouco mais alto nível do que a linguagem de máquina.

☹️ Ao seguir o passo a passo, você chega no resultado...mas pode ser difícil racionalizar qual será ele.

☹️ Qualquer um pode usar suas variáveis globais e suas funções, para o seu uso intencional ou não...

Paradigma Estruturado

- Estrutura o código imperativo em blocos lógicos.
- Esses blocos contém instruções imperativas, esperançosamente para cumprir um único objetivo.
- Elimina o uso de *goto*.

Paradigma Estruturado

```
aprovados = {}
for (i = 0; i < length(alunos); i++) {
    a = alunos[i];
    if (a.nota >= 5) {
        adiciona(aprovados, toUpper(a.nome));
    }
}
return sort(aprovados);
```

Paradigma Estruturado

😄 O programa é dividido em blocos lógicos, cada qual com uma função explícita (se for bom programador).

😄 Estimula o uso de variáveis locais pertencentes a cada bloco lógico.

☹️ Não evita que certas informações sejam utilizadas fora do seu contexto.

☹️ Usa mudança de estados, o que pode levar a *bugs*.

Orientação a Objetos

- Encapsula os dados em classes cada qual contendo seus próprios estados e métodos, não compartilhados.
- Um objeto pode se comunicar com outro, não usa (ou evita) variáveis globais.

```
class Aprovados {
    private ArrayList aprovados;
    public Aprovados () {
        aprovados = new ArraList();
    }
    public addAluno(aluno) {
        if (aluno.nota >= 5) {
            aprovados.add(aluno.nome.toUpper());
        }
    }
    public getAprovados() {
        return aprovados.sort();
    }

}
```

Orientação a Objetos

😄 Encapsula códigos imperativos para segurança e reuso.

😄 Evita o uso de variáveis globais, inibe uso indevido de trechos de códigos.

☹ Nem tudo faz sentido ser estruturado como objetos.

☹ Composição de funções é feito através de herança, que pode *bagunçar* o fluxo lógico do código.

☹ Uso intensivo de estados mutáveis.

☹ Difícil de paralelizar.

Paradigma Declarativo

- O fluxo lógico é implícito.
- Linguagens de alto nível, o programador escreve o que ele quer obter, não como.

Paradigma Declarativo

```sql
SELECT   UPPER(nome)
FROM     alunos
WHERE    nota >= 5
ORDER BY nome
```

Paradigma Declarativo

😄 Utiliza uma linguagem específica para o domínio da aplicação (*Domain Specific Language* - DSL), atingindo um nível mais alto que outros paradigmas.

😄 Minimiza uso de estados, levando a menos bugs.

☹️ Fazer algo não suportado nativamente na linguagem pode levar a códigos complexos ou uso de linguagens de outros paradigmas.

☹️ Pode ter um custo extra na performance.

Paradigma Lógico

- Especifica-se apenas fatos e regras de inferência.
- O objetivo (retorno) é escrito em forma de pergunta.

Paradigma Lógico

```
aprovado(X) :- nota(X,N), N>=5.

sort(
  findall(Alunos, aprovado(Alunos), Aprovados)
    )
```

Paradigma Lógico

😄 O compilador constrói o programa para você, baseado em fatos lógicos.

😄 Provar a corretude do programa é simples.

☹️ Algoritmos mais complexos podem ser difíceis de expressar dessa forma.

☹️ Costuma ser mais lento para operações matemáticas.

Paradigma Funcional

- Baseado no cálculo $lambda$.
- Programas são composições de funções.
- Não utiliza estados.
- Declarativo.

Paradigma Funcional

```
sort [nome aluno | aluno <- alunos, nota aluno >= 5]
```

Paradigma Funcional

😄 Expressividade próxima de linguagens declarativas, mas sem limitações.

😄 Não existe estado e mutabilidade, isso reduz a quantidade de *bugs*.

😦 Como fazer algo útil sem estados?

😦 A ausência de mutabilidade dificulta o gerenciamento de memória, intenso uso de *Garbage collector*.

Paradigma Funcional

Características

- Funções puras
- Recursão
- Avaliação Preguiçosa

Efeito colateral e estados

Efeito colateral ocorre quando uma função altera algum estado global do sistema:

- Alterar uma variável global
- Ler entrada de dados
- Imprimir algo na tela

Funções Puras

Funções puras são funções que não apresentam efeito colateral.

Ao executar a mesma função com a mesma entrada **sempre** terei a mesma resposta.

Funções Puras

Se não temos efeito colateral...

- ...e o resultado de uma expressão pura não for utilizado, não precisa ser computado.
- ...o programa como um todo pode ser reorganizado e otimizado.
- ...é possível computar expressões em qualquer ordem (ou até em paralelo).

Funções Puras

```
double dobra(double x) {
  return 2*x;
}
```

Funções Impuras

```
double i = 0;

double dobraMaisI(double x) {
  i += 1;
  return 2*x + i;
}
```

Pergunta

Classifique as seguintes funções em puras ou impuras:

- *strlen*
- *printf*
- *memcpy*
- *getc*

Funções Impuras

```c
double media (int * valores, int n) {
    double soma = 0;
    int i;
    for (i = 0; i < n; i++)
        soma_valor(&soma, valores[i]);
    return soma / n;
}

void soma_valor (double * soma, int valor) {
    soma += valor;
}
```

Programação sem bugs

A ausência de estados permite evitar muitos erros de implementação.

O lema da linguagem Haskell: “se compilou, o código está correto!” (e não só pela pureza).

Iterações vs Recursões

Em linguagens funcionais os laços iterativos são implementados via recursão, geralmente levando a um código enxuto e declarativo.

Iterações vs Recursões

```
int gcd (int m, int n) {
  int r = m % n;
  while(r != 0) {
      m = n; n = r; r = m%n;
  }
  return m;
}
```

Iterações vs Recursões

```
mdc 0 b = b
mdc a 0 = a
mdc a b = mdc b (a `rem` b)
```

Avaliação Preguiçosa

Algumas linguagens funcionais implementam o conceito de avaliação preguiçosa.

Quando uma expressão é gerada, ela gera uma promessa de execução.

Se e quando necessário, ela é avaliada.

Avaliação Estrita

```
int main () {
    int x = 2;
    f(x*x, 4*x + 3);
    return 0;
}

int f(int x, int y) {
    return 2*x;
}
```

Avaliação Estrita

```c
int main () {
    int x = 2;
    f(2*2, 4*2 + 3);
    return 0;
}

int f(int x, int y) {
    return 2*x;
}
```

Avaliação Estrita

```
int main () {
    int x = 2;
    f(4, 4*x + 3);
    return 0;
}

int f(int x, int y) {
    return 2*x;
}
```

Avaliação Estrita

```c
int main () {
    int x = 2;
    f(4, 11);
    return 0;
}

int f(int x, int y) {
    return 2*x;
}
```

Avaliação Estrita

```
int main () {
    int x = 2;
    8;
    return 0;
}

int f(int x, int y) {
    return 2*x;
}
```

Avaliação Preguiçosa

```
f x y = 2*x

main = do
  let z = 2
  print (f (z*z) (4*z + 3))
```

Avaliação Preguiçosa

```
f x y = 2*x

main = do
  let z = 2
  print (2 * (z*z))
```

Avaliação Preguiçosa

```
f x y = 2*x

main = do
  let z = 2
  print (8)
```

A expressão $4 * z + 3$ nunca foi avaliada!

Avaliação Preguiçosa

Isso permite a criação de listas infinitas:

```
[2*i | i <-[1..]]
```

Haskell

- Linguagem puramente funcional (não é multi-paradigma)
- Somente aceita funções puras (como tratar entrada e saída de dados?)
- Declarativa
- Avaliação Preguiçosa
- Dados imutáveis
- Tipagem estática e forte

Sobre a disciplina

Sobre as aulas

Nas aulas de laboratório colocaremos em prática o que foi aprendido durante as aulas teóricas

As aulas teóricas serão expositivas e, sempre que possível, contendo fundamentação matemática

Site da disciplina

https://folivetti.github.io/teaching/2018-summer-teaching-3

ou

`https://folivetti.github.io/` $\rightarrow$ Teaching $\rightarrow$ Paradigmas de Programação

Exercícios e atividades

Lista de exercícios para praticar (não valendo nota)

Atividades em laboratório valendo nota

Avaliação

- 02 Atividades = total de 3 ptos (10/07 e 21/08)
- 01 Prova teórica = 4 ptos (12/07)
- 01 Projeto = 4 ptos (entrega dia 14/08)

Avaliação

O projeto deve ser:

- Implementação de algum algoritmo de outras disciplinas (com um nível médio-difícil de complexidade)
- Tutorial de uso de alguma biblioteca do Haskell (para manipular imagens, cálculo numérico, jogos - não vale traduzir)
- Tutorial de algum conceito avançado de Haskell ou Programação Funcional

Até 04 alunos por grupo e entrega até 14/08. Link para entrega será disponibilizado no momento oportuno.

Nota - Conceito

```haskell
conceito :: Double -> Char
conceito nota
  | nota >= 8 = 'A'
  | nota >= 7 = 'B'
  | nota >= 6 = 'C'
  | nota >= 5 = 'D'
  | otherwise = 'F'
```

Atendimento

Terças e Quintas - 07:30 - 08:00 (sala 522-2) Terças - 12:00 - 12:30 (sala 522-2) Terças - 08:00 - 10:00 (sala 522-2 ou L110)

Atendimento Online

Atendimento via Piazza:

`https://piazza.com/class/jhmdck6aeoq4pw`

- Até 1 pto por participar ativamente respondendo no piazza.
- Até 0.5 pto por participar ativamente perguntando no piazza.

Recuperação

Prova teórica valendo 10 pontos compreendendo toda o conteúdo da disciplina.

Conversão nota - conceito:

```
conceito :: Double -> Char
conceito nota
  | nota >= 7 = 'C'
  | nota >= 5 = 'D'
  | otherwise = 'F'
```

Cronograma

No site.