NOVA PRÁTICA DE ENSINO NA PROGRAMAÇÃO DE COMPUTADORES

Hellen Raile O. de Lemos[1] e Leonardo R. Lucena[2]
E-mail: hellen.lemos@academico.ifrn.edu.br[1]; leonardo.lucena@ifrn.edu.br[2]

RESUMO

Este artigo apresenta uma proposta do uso de linguagens multiparadigma para ensinar Programação de Computadores. A nossa experiência em sala de aula tem mostrado que enfatizar Programação Funcional, mas sem abrir mão, melhora o processo de ensino-aprendizagem. A linguagem Scala é um exemplo típico de uma linguagem multiparadigma moderna que se adequa a essa proposta.

PALAVRAS-CHAVE: Programação Funcional, Scala, Ensino.

New teaching practice in Computer Programming

ABSTRACT

This paper presents a proposal of using multiparadigm languages to teach Computer Programming. Our experience in the classroom has shown that emphasize Functional Programming, but without giving up, improves the process of teaching and learning. The Scala language is a typical example of a modern multiparadigm language that fits this proposal.

KEYWORDS: Functional Programming, Scala, Teaching.

1 INTRODUÇÃO

Nos cursos de Computação, é comum um grande número de desistência dos alunos. Isso se dá, entre outras razões, por conta da temível programação de computadores. Não é fácil estruturar um programa para solucionar determinados problemas e, além disso, preocupar-se com detalhes de sintaxe e semântica da linguagem de programação. A forma de estruturar esses programas, é uma das principais dificuldades encontradas pelos alunos iniciantes dos cursos de computação. O aluno muitas vezes desvia o foco do que o programa deve fazer para se preocupar como deve ser feito. É nesse contexto, que estamos propondo uma nova abordagem de ensino de programação. A ideia é usar linguagens multiparadigma com ênfase em programação funcional, mas sem abrir mão da programação orientada a objetos.

A organização do artigo segue inicialmente com a seção 2, discute problemas encontrados pelos alunos iniciantes dos cursos de computação, em seguida na seção 3 apresentamos nossa proposta como uma forma de lidar com o problema. Na seção 4 abordaremos o uso da linguagem Scala para o ensino da programação. Finalizamos o artigo fazendo uma comparação entre as linguagens Scala, Portugol e Java, sucedida das considerações finais.

2 PROBLEMA

Para resolver um problema em um computador é necessário que seja ditado uma sequência de passos para que a máquina execute determinada tarefa, essa sequência de passos denomina-se algoritmo, um conjunto finito de regras que fornece uma sequência de operações para resolver problemas. A forma de estruturar, ou a criação de algoritmos é uma das maiores dificuldades encontradas por alunos iniciantes dos cursos de computação. A falta de recursos, de instrução pessoal, o mal entendimento do problema também são fatores de dificuldades.

As maiores dificuldades dos alunos iniciantes são na compreensão de como projetar um programa para resolver uma determinada tarefa, encontrar erros de seus próprios programas. Conceitos de programação como recursão, ponteiros e referências, tipo de dados abstratos, tratamento de erros, e o uso de bibliotecas de linguagem, são também fatores de dificuldades desses alunos. Outro fator encontrado nos estudos deste artigo foi o nível experiência dos estudantes, mais da metade (58,6%) dos estudantes que participaram da pesquisa já tinham experiência com programação, isso acaba tornando difícil ensinar alunos de diferentes níveis de experiência. Um ponto a observar é que quase a metade (40,6%) dos que tinham experiência em programação, afirmam que suas habilidades foram moderadas.

Outro aspecto é a relação da Computação com a Matemática. A matemática costuma ser apresentada como a base para computação. Mas os alunos logo se deparam com situações do tipo:

2.1 Precisões de valores numéricos

Os valores contínuos da matemática precisam ser discretizados na computação. Variáveis que mudam de valor ao longo do programa. Isto é impensável na matemática.

3 SOLUÇÃO

Analisando as dificuldades encontradas pelos alunos iniciantes dos cursos de computação vimos a necessidade de se mudar um pouco a forma de ensinar programação. Ao invés de ensinar, a princípio, programas com variáveis que mudam de valor durante a execução, optamos por algo mais simples, como a Programação Funcional, que imita as funções matemáticas no maior grau possível. Esta forma de solucionar problemas é uma forma totalmente diferente das abordagens usadas com linguagens imperativas. Em uma linguagem imperativa, que geralmente é apresentada aos alunos na disciplina de Introdução à Programação, uma expressão é avaliada e depois é armazenada em uma posição de memória, representada como uma variável de programa. O modelo de computação é baseado na alteração do estado das variáveis, o que dificulta o entendimento dos programas. Em uma linguagem de programação puramente funcional o conceito de variável é diferente, as variáveis são nomes que damos a um valor. Elas são uma forma de armazenar o resultado da avaliação de uma expressão para depois pode usar o mesmo valor em outra parte do programa. Assim, o entendimento dos programas é mais fácil já que é possível compreender um trecho do código apenas analisando estaticamente.

Diferentemente das linguagens imperativas, as linguagens funcionais podem ter uma estrutura sintática muito simples, como também a semântica. As Figuras 1 e 2 mostram a sequência de Fibonnaci, primeiro escrita em Scala (Figura 1) e depois em C (Figura 2).

```
1  object Fibonnaci extends App {
2    print(fib() take 10 toList)
3
4    def fib(a: Long = 1, b: Long = 1): Stream[Long] = a #:: fib(b, a + b)
5  }
6
7
```

Figura 1: Sequência de Fibonnaci escrito em Scala

```c
// Inclui o arquivo <"stdio.h">
// stdio.h é responsável pelas funções de entrada e saída.
#include "stdio.h"


void main()
{

  int a, b, c, i, n;


  a = 0;
  b = 1;


  printf("Digite um número: ");

  scanf("%d", &n);
  printf("Série de Fibonacci:\n");
  printf("%d\n", b);


  for(i = 0; i < n; i++)
  {
    c = a + b;
    a = b;
    b = c;


    printf("%d\n", c);
  }
}
```

Figura 2: Sequência de Fibonnaci escrito em C.

Devido aos programas funcionais poderem chegar a 10% do tamanho de seus correspondentes imperativos (Wadler, 1998), alguns programadores da comunidade de programação funcional afirmam ter um grande aumento de produtividade. Embora esse número tenha sido mostrados em problemas específicos, muitos programas funcionais são cerca de 25% do tamanho das soluções imperativas para os mesmo problemas. Esses fatores fazem com que as linguagens funcionais sejam mais produtivas.

3.1 Porque usar Programação Funcional

Com o rápido crescimento da demanda de software, entre os anos 60 e 70, houve uma crise, onde grandes projetos de software falhavam por causa de ferramentas impróprias, e como forma de passar por isso, as linguagens funcionais foram de grande utilidade.

As linguagens funcionais dão ênfase no processo de identificar blocos e partes repetidas de códigos (como a leitura de arquivo ou cálculo de uma raiz) e constroem funções sem efeito colateral que encapsulam a funcionalidade dentro de uma simples definição, isso faz com que se aumente a manutenibilidade e a confiabilidade.

Na programação funcional, os subprogramas são enxergados como funções, eles recebem parâmetros e retornam soluções simples, baseada na entrada desses parâmetros. Possui também um alto nível de abstração, nos dando alta produtividade, programas mais concisos, mais fáceis de entender, com menos erros, não dependendo de atribuições, nos dando a liberdade de programar em diferentes ordens, sem precisar pensar em uma forma sequencial de passos, como a Programação Imperativa.

4 SCALA

A programação funcional está voltando com grande força, podemos perceber isso em linguagens de programação como Erlang, F# e Scala. A nossa proposta de ensino, está voltada para uma linguagem que não só tem suporte a Programação Funcional como também suporte ao modelo Orientado a Objetos. Linguagens como Scala mostram que o paradigma Funcional e o paradigma Orientado a Objetos são ortogonais, é possível unir os dois e o resultado dá um ganho que isoladamente eles não teriam. O motivo de escolhermos está linguagem foi por causa de sua grande portabilidade, e por estar também em grande uso nas mais diversas empresas (Twitter, Foursquare). No nosso projeto estamos preparando material didático para avaliar na prática o uso da linguagem Scala para o ensino de programação.

4.1 Como Scala surgiu?

Depois de muito tempo programando em Java, Martin Odersky passou por muitas frustrações com as centenas de milhares de linha da linguagem de programação Java. Insatisfeito com a linguagem, Martin resolveu utilizar o conhecimento da comunidade acadêmica para melhorar a programação em Java, sua solução foi desenvolver o Generic Java incorporado oficialmente ao J2SE 5.0, que por sinal foi muito bem aceito pela comunidade Java. Ainda insatisfeito, Martin Odersky resolveu juntar os recursos do modelo Orientado a Objetos e Funcional, a partir dessa ideia, nasce em 2001, Scala.

4.2 A linguagem Scala

Criada por Martin Odersky em 2001 e sendo disponibilizada em 2003 para a comunidade de programadores, Scala é uma linguagem de programação de propósito geral projetado para expressar padrões de programação comuns de forma concisa. Ela roda na Java Virtual Machine e mantém uma forte interoperabilidade com Java e a plataforma .Net. Sua fácil integração com outras linguagens vem atraindo a atenção dos desenvolvedores como também das empresas. No Twitter houve uma migração do processamento das mensagens da linguagem Ruby para Scala. Essa mudança foi impulsionada pela necessidade de a empresa expandir de forma confiável o seu funcionamento para atender a rápida taxa de crescimento de Tweet, já atingindo 5.000 por minuto.

Como podemos observar, Scala é uma linguagem multiparadigma, ou seja, suporta mais de um paradigma de programação, que como já citados, são o Orientado a Objetos, que significa

dizer que todo o valor é um objeto, e Funcional onde cada função é um valor, logo toda função também é um objeto.

Scala possui inferência de tipos de dados, por exemplo, ao declarar uma variável, não precisamos definir explicitamente o tipo dela, `val x = 10`. Os tipos omitidos são analisados pelo compilador através dos valores que são atribuídos a variável, logo ele saberá se é Int, Float, Double e etc. Tipagem forte e estática também é uma de suas características, esses atributos facilitam a descoberta de muitos erros, que na maioria das vezes são provocados pelo mau uso das variáveis. Scala encoraja o uso de variáveis como valores (no sentido de programação funcional), facilitando o entendimento do código, mas permite também usar variáveis como referência a posição de memória (como na programação imperativa) para alguns casos onde a alteração do valor de uma variável simplifica o código. Nesse último caso é remendável restringir o uso de variáveis a um escopo bem pequeno para não "contaminar" o restante do programa já que usar valores alteráveis força o leitor executar mentalmente o programa para entender seu funcionamento.

5 COMPARAÇÃO ENTRE SCALA, PORTUGOL E JAVA

Aqui nós comparamos a linguagem Scala com outras duas linguagens muito usadas nas disciplinas de Introdução a Programação: Java e Portugol

5.1 Declaração de Variável

5.1.1 Portugol

```
var idade: inteiro
...
idade <- 18
```

Declaram-se as variáveis seguidas dos tipos das mesmas. A declaração e atribuição de um valor a uma variável é realizado em dois passos.

5.1.2 Java

```
int idade =18;
```

Declaram-se variável dizendo inicialmente seu tipo. No Java, ao declarar a variável você pode informar o valor dela, pode inicializá-la, dizendo por exemplo, que idade = 18. Ou pode pedir para que o usuário, ou programador digite a idade. Para isso precisa-se usar a classe Scanner, que é utilizado para entrada de dados via teclado.

5.1.3 Scala

```
val idade = 18
```

Declaram-se a variável e o valor. Em Scala, o programador é encorajado a utilizar **val**, que é uma declaração funcional (constante). Você pode usar o **var** também, porém a utilização de variáveis

constantes facilita o encontro de erros no código. A inferência de tipos também é uma grande característica do Scala, por exemplo:

```
scala> val responda = 8 * 5  + 2
responda: Int  = 42
```

Neste exemplo, ao declarar 'responda' atribuímos diretamente o valor a ela, sem precisar dizer o tipo. O resultado é que nos informa, dizendo que a saída é um inteiro, isto ocorre, porque o compilador analisa os valores.

5.2 Estrutura de decisão simples

5.2.1 Portugol

```
var idade: inteiro

leia(idade)
se (idade > = 18) então
escreva("Você é maior de idade!")
senao
escreva("Você é menor de idade!")
fimse
```

5.2.2 Java

```
Scanner scan = new Scanner(System.in);
int idade;
idade = scan.nextInt();
if(idade >= 18){
   System.out.println("Você é maior de idade!");
}else{
   System.out.println("Você é menor de idade!");
}
```

5.2.3 Scala

```
val idade = readInt

if(idade >= 18)

  println("Você é maior de idade!")

else

  println("Você é menor de idade")
```

Note que não precisamos usar o (;) no final de cada linha do código, assim como em Java.

5.3 Estrutura de decisão de múltipla escolha

5.3.1 Portugol

```
escolha opcao
caso 1
escreva(“Sair de casa as 8 horas da manhã.”)
caso 2
escreva(“Sair de casa às 2 horas da tarde.”)
caso 3
escreva(“Sair de casa ao meio-dia.”)
outrocaso
escreva(“Já que não optou, fique em casa mesmo e leia um livro.”)
fimescolha
fim
```

5.3.2 Java

```
switch(opcao){
case 1: System.out.println(“Sair de casa as 8 horas da manhã.”);break;
case 2: System.out.println(“Sair de casa às 2 horas da tarde.”); break;
case 3: System.out.println(“Sair de casa ao meio-dia.”); break;
default: System.out.println(“Esta opção não é válida.”);
}
```

5.3.3 Scala

```
opcao match {
  case 1 => println(“...”)
  case 2 => println(“...”)
 case _ => println(“…”)
}
```

5.4 Estrutura de repetição

5.4.1 Portugol

```
para contador de 1 ate 5 passo 1 faca
escreval(“Bem vindo a Portugol!”)
fimpara
```

5.4.2 Java

```
for(int i = 1; i < 5; i++){
   System.out.println(“Bem vindo a Java!”);
```

```
}
```

O laço de repetição em Java, irá se repetir até que a condição seja verdadeira, ou seja, se i é igual a 1, enquanto i for menor que 5 escreva “Bem vindo a Java!”. Ao final do teste a variável i será incrementado é um.

5.4.3 Scala

```
for(i <- 1 to 5){
   println(“Bem vindo a Scala!”)
}
```

Como podemos perceber, Scala tem uma sintaxe muito simples, comparado a outras. Sua legibilidade torna fácil o entendimento e a leitura do código, como também traz muita confiança para quem está desenvolvendo programas.

6 CONSIDERAÇÕES FINAIS

Neste artigo apresentamos uma nova abordagem para a disciplina de Introdução a Programação. A nossa ideia é usar uma linguagem de programação que facilite a aprendizagem dos conceitos de programação. Os cursos de Introdução à programação normalmente começam ensinando os conceitos de programação usando os paradigmas imperativo ou orientado a objetos. No entanto nesses paradigmas o aluno precisa conhecer o modelo de execução baseado em variáveis. Isto dificulta o entendimento e a elaboração dos programas.

Usando uma linguagem multiparadigma que dá ênfase ao paradigma funcional permite apresentar os conceitos de programação de uma maneira mais simples e mais próxima dos conceitos de Matemática que os alunos já conhecem. Os programas estruturados de uma forma muito mais declarativa em oposição a uma forma operacional das linguagens imperativas.

O próximo passo do nosso trabalho é aplicar à abordagem em disciplinas de introdução a programação tanto dos cursos superiores como nos cursos técnicos. Planejamos fazer experimentos com os alunos para avaliar os pontos positivos e negativos da nossa abordagem.

7 AGRADECIMENTOS

CNPq pela bolsa de pesquisa.

Pablo pelas críticas e sugestões durante a fase de elaboração deste artigo.

8 REFERÊNCIAS BIBLIOGRÁFICAS

Architects, I. (s.d.). *Paradigma Funcional: Conceitos Básicos*. Fonte: http://www.din.uem.br/ia/ferramentas/lisp/lisp3.htm

BORBA, L. (27 de Abril de 2010). Falando sobre tecnologia. *Porque linguagens funcionais são importantes* .

CRUZ, ADRIANO JOAQUIM DE OIVEIRA;. (1997). Fonte: Algortimos: http://equipe.nce.ufrj.br/adriano/c/apostila/algoritmos.htm

ESSI LAHTINEN, K. A. (Junho de 2005). Em ITiCSE '05: Anais da 10 ª conferência anual sobre SIGCSE. *A Study of the Difficulties of Novice Programmers* , p. 5.

HORSTMANN, C. (2012). *Scala for the Impatient.* San Francisco: Copyright.

LAUSANNE, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE;. (2013). *Scala*. Fonte: The Scala Progamming Language: http://www.scala-lang.org/

NÉLIO ALESSANDRO AZEVEDO CACHO, KEIVILANY JANIELLE DE LIMA COELHO;. (s.d.). *Linguagem de Programação e Algoritmos.* Fonte: Metrópole Digital: http://www.metropoledigital.ufrn.br/aulas/disciplinas/logica/aula_08.html

SEBESTA, ROBERT W.;. (2011). *Conceitos de Linguagem de Programação.* Porto Alegre: Bookman.

SILVA, P. (5 de Agosto de 2012). *Scala: sua próxima linguagem de programação*. Fonte: EstruturaOpen: http://estruturaopen.com/2012/08/05/sua-proxima-linguagem-de-programacao-scala/