REVISTA DE CIÊNCIAS EXATAS E TECNOLOGIA

LÓGICA DE PROGRAMAÇÃO
Arte, técnica ou ambas?

Jorge Luiz Surian – *Centro Universitário Anhanguera de São Paulo - unidade Brigadeiro*

RESUMO: Nesse breve artigo, discutindo o ensino de Lógica de Programação, relativamente a seus aspectos formais e técnicos, mas salientando também os aspectos criativos que normalmente são esquecidos ou deixados em segundo plano. Além disso, também é discutida a questão vocacional do estudante para aprender a programar computadores.

PALAVRAS-CHAVE:
Lógica de Programação; Algoritmos; Estruturas de Programação.

ABSTRACT: In this brief article, argued the education of Logic of Programming, relatively its formal aspects and technician, but also pointing out the creative aspects that normally are forgotten or left in second plain. Moreover, also is argued the vocational question of the student to learn to program computers.

KEYWORDS:
Logic Programming, Algorithms, Programming Structures.

Artigo Original
Recebido em: 30/06/2011
Avaliado em: 14/02/2014
Publicado em: 28/04/2014

Publicação
Anhanguera Educacional Ltda.

Coordenação
Instituto de Pesquisas Aplicadas e Desenvolvimento Educacional - IPADE

Correspondência
Sistema Anhanguera de Revistas Eletrônicas - SARE
rc.ipade@anhanguera.com

1. INTRODUÇÃO

Algum tempo atrás numa aula de mestrado, um professor começou a discorrer sobre a temática lógica de programação de computadores como uma arte que exige vocação aliada a alguns conhecimentos técnicos. A aula seguia, até meio modorrenta, quando um dos mestrandos decretou: Programar não é arte, não passa de técnica!

Afirmou isso com tanta segurança que o professor vacilou por um instante. Recuperando-se, perguntou ao afrontoso aluno em que linguagens programava, qual era sua experiência que o levava a discordar de forma tão veementemente do professor.

O aluno, candidamente, disse que não sabia programar. Aliás, tratava-se de um cineasta que resolvera fazer aquele mestrado na área de administração, quase que por diletantismo.

A discussão seguiu em diante, com resultado inconclusivo, mas esse fato foi a primeira centelha de muitas outras, que motivou uma pesquisa voltada a esse tema, uma vez que lecionando "lógica de programação" por quase duas décadas, em alguma cadeira de algum curso na área de desenvolvimento de software, quase que obriga a reflexão dessa controversa questão que intitula esse artigo.

Ao longo desse tempo, ensinando lógica de programação a alunos e profissionais da área de desenvolvimento, não houve como manter o modelo clássico da construção do conhecimento que habitualmente qualquer professor pratica.

Piaget (399) afirmou não ter "a ambição de decidir um tudo e um nada, mas apenas a candura de querer diferenciar as situações e de julgar que ganhei em todos os planos". Assim cabe ao educador buscar meios que lhe façam crer, genuinamente, no progresso de seus estudantes. Mesmo que não seja inútil propormos exercícios, práticas e atividades em grau de dificuldade crescente, para ao final de algum tempo termos formado um programador, alguma coisa sempre parecia faltar...

2. O ENSINO DE LÓGICA

Essa forma de ensinar "programar" que sempre se inicia por fazer o aluno entender como entram os dados e saem informações; como se tomam decisões; como se faz para que um conjunto de instruções seja repetido, e, finalmente, como se constroem estruturas de dados consistentes. Isso forma programadores, mas não forma um desenvolvedor de sistemas. Ora, como o mercado deseja este e não aqueles, algo parecia estar, como escreveu o compositor Caetano Veloso, fora da ordem...

Em sua brilhante resposta a Piaget, Noam Chomsky afirma "Ele qualifica muito justamente as minhas concepções como sendo, para usarmos os seus termos, uma forma de 'inatismo'." (Piatelli-Palmarini, 50)[1]

[1] Piatteli-Palmarini organizou livro resultante do debate promovido em 1975 onde se punham lado a lado as teorias de Chomsky (a liguística generativa) e de Piaget (a epistemologia genética).

Mergulhando mais profundamente nos estudos do controvertido linguista, foi se tornando cada vez mais difícil ignorar o fato de que determinados alunos pareciam sempre "já saber" aquilo que lhes era ensinado e, outros tantos, por mais exercícios e exemplos que fossem dados, continuavam a olhar para as estruturas de programação como que contemplando uma esfinge, sem terem nenhuma resposta a ser dada... Esses alunos, olhando os alunos "programadores" como se estes, tal qual modernos édipos, tivessem decifrado um enigma que aos reles mortais jamais teria sua resposta revelada...

Na condição de professor, e todo professor bem sabe o quanto, doloroso ver um aluno debater-se contra a ignorância, por mais que o professor se esforce para esclarecer uma dívida. Ver o aluno sucumbir, como se estivesse a ser tragado por areia movediça, sem que o professor nada possa fazer para evitar que isso ocorra. Todavia, sempre se tenta descobrir outra forma de ensinar, outra forma de ver a questão, outra maneira de pensar, buscando através de alguma maneira fazer aquele estudante desesperado, por vezes já com lágrimas nos olhos, entender aquelas estruturas de programação, como se fosse uma implacável Esfinge.

Muitas vezes, depois de v rias tentativas, as lágrimas nos olhos davam lugar a um brilho diferente no olhar do estudante, que repentinamente parecia ter entendido não apenas a dívida que o havia exasperado tanto, mas aparentemente ter descoberto todos os segredos da lógica de programação. Quando um professor passa por essa experiência, num primeiro momento confundem-se dois sentimentos quase que antagônicos: alegria e alívio.

Alegria, pois qual professor não se sente recompensado quando percebe que um seu aluno encontrou a luz. Tal qual um moderno Prometeu, o professor se vê levando o fogo do conhecimento a seu discípulo. Talvez esse seja o maior prazer de todo professor, que merece artigos e mais artigos...

Contudo, parece mais adequado analisar o outro sentimento citado e que, menos nobre: o alívio.

O estudante que luta desesperadamente contra sua incapacidade de entender um conceito, vendo outros seus iguais que parecem entendê-los tão facilmente, algo para eles tão distante e misterioso, faz o professor sofrer junto com o aluno!

Ora, esse professor também começa a desconfiar de sua capacidade de ensinar e de se fazer entender, assim quando finalmente o estudante passa entender o que o professor tentava a todo custo ensinar, faz-se o alívio. Talvez, o alívio seja um sentimento mais forte no professor do que no próprio aluno, o que acaba deixando a alegria de ensinar num segundo plano...

Depois de alguns anos lecionando lógica de programação, tal qual um *déjà vu* contínuo, situações como essas vão se repetindo. Se nas primeiras vezes parecem mais milagres esses "aprendizados instantâneos", com sua repetição o professor atento começa a desconfiar que algo além de uma explicação brilhante ou de um real esforço do estudante acontece.

Com o passar dos anos, parece se aplicar na programação a máxima de que é necessário 95% de esforço (ou técnica, nesse caso) e 5% de talento (ou arte, nesse caso). Mais que isso, embora o talento exigido não passe de ínfimos 5%, parece que sem esses 5% os outros 95% não sirvam para coisa alguma...

Assim, por melhor que seja o professor e os recursos disponíveis, não parece ser possível fazer um desenvolvedor de sistemas, calcado apenas na técnica.

O objetivo desse texto não ser estudar os 95% de técnica necessária a um desenvolvedor de sistemas, mas sim aos 5% de arte, costumeiramente esquecidos por todos nós que ensinamos lógica de programação.

3. APRENDIZADO "INSTANTÂNEO"

Além de entender algo momentâneo, essa compreensão parece não ter fim. Aquele aluno desajeitado, tal qual um pássaro desajeitado que ainda não sabe voar, de repente não apenas entende um conceito obscuro, mas começa a aprender todas as estruturas de programação dali em diante, pouco importando o grau de complexidade continuamente crescente.

Aliás, depois dessa fagulha nada mais parece ter poder de deter esse aluno, que em desejando, se torna o "programador" que desejar ser. Parece muito mais que daí em diante o professor acabe se tornando muito mais uma figura que atrapalha, em vez de ajudar o aluno no contínuo aprendizado, como Chomsky disse.

Ao impor suas métricas e técnicas, o professor acaba impedindo o desabrochar da criatividade e, com isso, acaba dando razão ao cineasta citado no início desse artigo.

Vista por este prisma, Lógica de Programação, não parece nada além do que mera técnica aplicada.

Mas, se assim , por que nem todos os alunos conseguem aprender essa "técnica", por mais que se esmerem? Ora, podemos ter mais ou menos capacidade para aprender a operar uma máquina, escrever, executar uma tarefa, mas todas as técnicas podem nos ser ensinadas e reproduzidas até com certa facilidade. Se lógica de programação não passa de uma mísera técnica, tão simples que quando compreendida parece não apresentar maior dificuldade para quem a entendeu, por que continua tão misteriosa a tantos outros estudantes?

Bunge (1988) entendeu que a capacidade de aprender coisas tais como uma gramática universal é uma capacidade que possuímos de nascimento, assim talvez por ser mais do que simples técnica, talvez por serem inatos que alguns têm muito, outros em quantidade suficiente e outros simplesmente não possuam o quinhão mínimo para ser capaz de entender e se tornarem capazes de escrever sistemas, não importando quantos cursos, quanto inteligente for e, para desespero dos professores, por mais brilhantes que sejam seus mestres.

Parece mais que o papel que cabe ao professor , o de remover um pesado, para uns, não tão pesado assim para outros, obstáculo existente na capacidade de aprender a programar. Uma vez removido esse empecilho, não parece existir mais limite para o recém-liberto aluno...

4. ESTRATÉGIAS DE ENSINO

O ensino de programação, não obstante a profusão de livros de lógica de programação, construção de algoritmos ou de determinada linguagem de desenvolvimento, quase sempre tem uma única estratégia, que se divide em determinadas etapas, a saber:

1. Faz-se uma introdução à lógica de programação.
2. Explica-se o que são variáveis, constantes, operadores relacionais, as entradas e saídas.
3. Entra-se pelas estruturas de seleção, enfatizando-se os aspectos de tomada de decisão.
4. Passa-se pelas estruturas de repetição, com seus "para-enquanto-faça-repita", como recitam legiões de alunos doutrinados.
5. Finalmente, se chega às estruturas de dados, com os vetores e matrizes, que parecem mais terem sido criados para tirar o sono dos alunos, do que para servirem para algo mais prático.

Um texto que fale de Lógica de Programação poderia antes falar de Lógica, de seus aspectos filosóficos, antes de chegar ao ponto de discutir a Lógica de Programação propriamente dita.

Nesse trabalho vamos inverter essa equação, usando do mesmo estratagema usado para ensinar a lógica boolena (Na matemática e na ciência da computação, as álgebras booleanas são estruturas algébricas que "capturam a essência" das operações lógicas **E**, **OU** e **NÃO**, bem como das operações da teoria de conjuntos soma, produto e complemento. Receberam o nome de George Boole, matem tico inglês, primeiro a defini-las como parte de um sistema de lógica em meados do século XIX, sem deixar metade dos alunos dormirem no meio de uma intrincada explicações de condições "e / ou" simultâneas.

Claramente, todos sabemos diferenciar o "e" do "ou". Sabemos que ganharemos determinada coisa se fizermos isso "e" aquilo. Não tentaremos, a não ser que movidos por algum sentimento menos nobre, a tentar obter o que nos foi prometido tendo feito somente "aquilo", mas não "isso". Por que, então, parece tão difícil a alguns compreender uma mera instrução

Se a = 1 e b = 1 então
imprima 'a e b são iguais a 1!'
senão
imprima 'a e b não são iguais a 1.'

Assim, em vez de passar um longo tempo tentando ensinar essas estruturas que parecem tão distantes das vidas de alguns dos estudantes, que tal usar algo mais lúdico e mais de acordo com o que cada estudante vive em seu dia-a-dia?

A idéia, se tivermos dinheiro é estiver fazendo sol, vamos para o Guarujá, uma praia famosa no litoral paulista. Já se tivermos dinheiro, mas não estiver fazendo sol, então vamos a um restaurante da rua Avanhandava. Essa, uma rua muito conhecida em São Paulo, por ter restaurantes de bom gosto. Ainda assim, mesmo que não tenhamos dinheiro, mas contemos com sol, podemos nos divertir indo ao parque Ibirapuera, o mais conhecido da cidade de São Paulo.

Todavia, se tudo der errado, ou seja não tivermos nem dinheiro tampouco sol, só nos resta então irmos assistir uma aula de lógica...

Depois dos inevitáveis sorrisos, basta escrevermos na lousa...

```
Se Dinheiro então
    Se Sol então
        Guarujá
     Senão
          Restaurante
Senão
    Se Sol então
          Ibirapuera
    Senão
           Aula
```

Em seguida basta escrever:

Se Dinheiro E Sol então Guarujá

Se não Dinheiro e Sol então Ibirapuera

Se Dinheiro e não Sol então Restaurante

Se não Dinheiro e não Sol então Aula

De uma tacada só explicamos o “E”, o “Não” e a difícil questão do encadeamento de decisões, sem falar em uma coisa ou outra. Naturalmente com um pouco de imaginação trabalhamos nesse exemplo de forma a contemplar a cláusula “OU”.

Para finalizar essa questão, depois de alguns exercícios simples para fixar o conceito, concluímos a aula com um enigma.

Nesse enigma, um certo inglês chamado Boole , embarcado numa caravela (sic) para chegar as Índias, mas para infelicidade sua a nau acaba afundando bem as costas da África. Nesse tempo imaginário, a África ainda era habitada por canibais, que tinham um estranho ritual. Todo prisioneiro capturado seria imolado a um de seus deuses no dia seguinte a captura. Se dissesse uma verdade, seria imolado ao Deus da Verdade, mas se dissesse uma

mentira no altar do Deus da Mentira. Se disser algo incompreensível aos selvagens (que, claro, falam inglês fluente, pois conversam com Boole em gentil e alto nível...) será imolado ao Deus da Mentira.

Foi dito ainda que o sujeito tudo pode na noite que antecede sua execução, podendo se regalar com comidas, bebidas e o que mais lhe aprouver. Para apimentar mais o problema, contamos que todos os prisioneiros ficam tentando escapar a noite toda, sendo inevitavelmente apanhados pelos canibais e, depois de castigados, voltam a sua cabana luxuosa. Mas, ao contrário de todos os outros, Boole não fica nem um pouco preocupado com a morte iminente. Escolhe as melhores frutas, bebidas e ainda solicita a presença da filha mais bonita do chefe da tribo em sua cabana...

A aula termina nesse ponto, tal qual um capítulo de novela. A resposta, solicitada avidamente, mas os alunos são avisados que a resposta só virá na próxima semana.

Em tempos de Google, no mesmo dia os alunos descobrem a resposta. Todavia, mais do que resolverem o enigma, os alunos começam a discutir a frase dita pelo "Boole" de nossa história **"Vou morrer no altar do Deus da Mentira.".**

Inevitavelmente, na próxima aula vários alunos vêm com a resposta na ponta da língua, com área de superioridade, muitos deles trazendo enigmas ao melhor estilo das revistas de passatempo...

Todavia, alguns outros perguntam com curiosidade ímpar, por que ele disse tal frase?

Esse simples enigma já revela ao professor aqueles alunos que não possuem vocação para programar. Muitas vezes, existe apenas uma trava, que se removida capacitar o aluno a aprender as mais intrincadas estruturas lógicas de programação. Contudo, noutras vezes, por melhor que seja a explicação o rosto com ar de dúvida permanece.

Não raro alguém diz, "professor, e se ele dissesse 'Vou morrer no altar da Verdade'?". A explicação, nesse caso é apenas praxe. A pergunta em si mesmo já revela toda dificuldade de abstração de quem fez a pergunta.

Segundo Forbellone e Eberspächer (2005, p2) lógica de programação "significa o uso correto das leis do pensamento da 'ordem da razão' e de processos de raciocínio e simbolização formais na programação de computadores, objetivando a racionalidade e o desenvolvimento de técnicas que cooperem para a produção de soluções logicamente v lidas e coerentes, que resolvam com qualidade os problemas que se deseja programar.". Mesmo concordando com essa conceituação, é forçoso observar que essa definição deixa pouco ou nenhum espaço para a criatividade.

Ao fechar esse espaço, talvez estejamos aí também aniquilando o pensamento diferente, as estratégias distintas que estão no cerne do desenvolvimento dos softwares que de fato quebram paradigmas.

Em vez disso podemos abrir esse espaço dizendo que todos nós, de um jeito ou de outro, consegue chegar ao trabalho, executar uma série de rotinas e estudar muitas coisas. Talvez não do jeito mais "correto", mas quem disse que , necessário que assim seja? A mera abertura ao pensamento dissidente cria oportunidade para o talento resolver problemas, muitas vezes de forma bastante diferente de uma solução convencional.

5. A LÓGICA E O FLUXOGRAMA

Para que serve uma ferramenta que nos auxilia na solução de algo que seria facilmente resolvido sem ela, mas que pouco adianta em problemas mais complexos?

Quase que todos os autores de livros de lógica de programação flertam com os fluxogramas, como ferramenta auxiliar no desenvolvimento de softwares, talvez pela simplicidade da técnica, talvez pela clareza dos símbolos.

Manzano e Oliveira (1996) utilizam todo primeiro capítulo de seu livro conceituando os vários tipos de fluxograma e os utilizam por toda publicação. J Forbellone e Ebersächer (2005, p9) fazem extensa justificativa para não adotarem o fluxograma e demais notações gráficas, todavia quem de fato socorre a nós professores de lógica, são Gane e Sarson (1983, p4) ao enunciarem que "fluxogramas causam mais mal do que bem", em seu memorável trabalho "Análise Estruturada de Sistemas". Difícil discordar de autores cultuados por toda uma geração de desenvolvedores de sistema. Náo o faremos aqui, tampouco. Na verdade, como dizem Forbellone e Ebersächer (2005, 9) "justificamos a opção pelos métodos textuais, que, apesar de menos puros, são mais naturais e fáceis de usar, ao optarmos pelo português estruturado, optamos por uma linguagem conhecida por todos."

Basta aplicar-se certo rigor sintético particular, onde apenas umas poucas palavras da língua portuguesa poderão ser utilizadas, para que se tenha uma ferramenta de programação.

Agora é possível responder a questão formulada anteriormente. De nada servem ferramentas que nos ajudam a solucionar problemas que poderiam ser solucionados sem elas, mas que em nada colaboram em problemas de difícil solução.

6. O ENIGMA DAS VARIÁVEIS

Depois de apresentarmos lógica de programação, chega a hora de apresentarmos as variáveis. Essa hora, especialmente crítica para o professor de lógica, pois de antemão sabe que muitos alunos terão enorme dificuldade em abstrair esse conceito. Notavelmente, a resposta de como ensinar o que, variável não parece estar nos livros de lógica, matemática ou de outras naturezas técnico-científicas, mas sim nos conceitos do Direito. Como sabemos, software é algo que mais se assemelha a um livro do que a algum outro tipo de produto. Nesse sentido, é ou ao menos deveria ser sempre regulamentado pela Lei de Direitos Autorais,

como ocorre em vários países. Temos aí uma pista importante sobre como ensinar variáveis, adotando um estratagema simples.

Todos os alunos conhecem sobejamente a história da Chapeuzinho Vermelho, com seus clássicos personagens (Lobo Mau, Vovó e Caçador). Chapeuzinho Vermelho está na memória de todos nós. Em vez da analogia direta entre personagens e variáveis, podemos escolher quatro atores de novela reconhecidos e simular cada um deles num dos papéis em questão. Uma jovem bela atriz, uma veterana atriz, um ator jovem e um ator reconhecido devem ser escolhidos. O professor deve criar um clima agradável e saber o nome da "mocinha" da novela das oito, embora não precise muito se preocupar com o Lobo Mau, pois invariavelmente algum ator com características do Lobo certamente ser apontado.

Claro que falta a ligação entre variáveis e personagens, mas isso pode ser facilmente solucionado, bastando para tanto elencarmos um quinto ator, necessariamente um comediante conhecido. Esse comediante permitirá que contemos cinco histórias diversas, pois não será difícil imaginar esse humorista, fazendo o papel de uma hilária Chapeuzinho ou de um extremamente covarde Caçador...

Nesse contexto, depois de muitos sorrisos, ficar mais fácil aos alunos entenderem que ao colocarmos o humorista no papel de Lobo, este pode ser qualquer coisa, menos Mau... Analogamente, se uma variável serve para guardar o maior número e outra para o menor, uma troca de uma pela outra nos levar a resultados totalmente distintos daqueles inicialmente esperados.

7. O PROCESSO DA TOMADA DE DECISÃO

Muito embora seja intuitivo o uso da expressão lógica "se ~ então ~ senão", caso o aluno de imediato venha a supor que há uma única maneira de resolver cada problema, rapidamente esse aluno criar para si mesmo inúmeras dificuldades para solucionar os problemas que usualmente ter que resolver.

Novamente, temos uma estratégia que leva o estudante perceber que pensamentos distintos, lógica distinta. Ainda assim, com resultados corretos. Se tudo passar a ser visto como simples distinção de estratégia adotada, facilmente se consegue quebrar o paradigma de que existe uma única lógica capaz de resolver uma questão.

O problema é um dos mais simples, constando em qualquer bom livro de lógica. Seu enunciado: "Dados três números, distintos entre si, apresente o maior deles."

Basta deixar os alunos "criarem" uma estratégia e várias das muitas possíveis estratégias para solução dessa questão, serão apresentados. Vamos analisar, por razões de espaço, somente três delas, mas que ladeadas por pelo menos outras três, servirão para demonstrar cabalmente que não há uma solução "correta" apenas.

A estratégia das comparações usando a cláusula “E” , usualmente a primeira a ser lembrada, até porque em geral ainda está na memário do estudante o exemplo do nosso intrépido e fictício Boole, do exemplo anterior.

Temos uma estrutura semelhante a apresentada a seguir:

```
leia a, b, c
se a > b e a > c então
    imprima a “, o maior”
senão
  se b > c então
     imprima b “, o maior”
  senão
     imprima c “, o maior”
```

Deve ser observado aos alunos que essa estratégia faz no máximo quatro comparações (a > b, a > c, resposta de cada uma delas , verdadeira? e b > c), usando três variáveis

Naturalmente uma questão que emerge diretamente , o porquê do “b” da segunda comparação não ter sido comparado também com o “a” e somente com o “c”. Depois de entendido por todos os alunos que uma vez que “a” já não é o maior, pois se assim fosse já teria sido impresso na primeira comparação. Analogamente se “a” e “b” não são os maiores, esse só pode ser “c”. Aí acaba se auto exemplicando a importância do “senão”. Se evita intermináveis explicações do porque não ser possível um “senão do senão” e da impraticabilidade de escrevermos código somente com “então” sem “senão”.

Uma segunda estratégia é aquela baseada na ordenação. Nem sempre , a estratégia mais lembrada, mas dada sua futura importância, já que é a base do algoritmo “buble sort”, sempre ensinado quando se pensa em ordenação de séries, em algum momento essa estratégia deve ser apresentada. Um algoritmo possível é o apresentado a seguir:

```
leia a, b, c
se a > b então
   d = a
   a = b
   b = d
se a > c então
   d = a
   a = c
   c = d
se b > c então
   d = b
   b = c
   c = d
imprima c
```

Deve ser observado aos alunos que essa estratégia faz sempre três comparações, usando quatro variáveis.

Embora seja muito mais evidente separar desde logo o maior número, a estratégia da ordenação completa, mais interessante, pois permitir que os alunos resolvam dois outros problemas, ou seja, identificar o menor número e o número médio. Embora o professor tenha que se preparar para uma saraivada de perguntas sobre médio e média, ainda assim costuma valer a pena a apresentação dessa segunda estratégia.

Outra estratégia que deve ser obrigatoriamente apresentada aos estudantes é a do encadeamento de tomada de decisões, apresentada adiante:

```
leia a, b, c
se a > b então
   se a > c então
      imprima a
   senão
     imprima c
senão
   se b > c então
      imprima b
   senão
     imprima c
```

Deve ser observado aos alunos que essa estratégia faz no máximo duas comparações, usando três variáveis. É bastante importante que os alunos entendam que a estratégia foi armada de tal forma que a segunda e a terceira tomadas de decisão são mutuamente excludentes, ou seja, se uma ocorrer necessariamente a outra não ocorre.

Mesmo que a terceira estratégia seja melhor, se considerarmos exclusivamente características técnicas, é inegável que uma estratégia que ordene os três números tem outras importantes aplicações e a estratégia usando a cláusula "E" tem compreensão mais simples e como um dos objetivos primordiais do programador , escrever programas que qualquer outro programador tenha facilidade em compreender o que foi escrito originalmente, essa estratégia não pode ser desconsiderada, pois por esse critério seria a melhor de todas.

Caberia aqui ainda uma longa discussão sobre outro importantíssimo tema: o uso de ferramentas orientadas a objeto desde cedo.

Essa questão dá por si só espaço para outro artigo voltado a discussão sobre a validade de se continuar a se utilizar ferramentas voltadas a interface caractere para apoio ao ensino de lógica.

O exemplo abaixo, usando a linguagem Pascal, mostra mesmo a quem não conhece a linguagem a distinção de conceitos entre uma e outra forma de expressar um mesmo conceito, numa mesma linguagem, mas em interfaces DOS[2] e Windows.

```
Algoritmo
    leia a
    imprima a+1
Em Turbo-Pascal (para interface caractere)
    var a : integer;
    begin
      Read(a);
      WriteLn(a+1);
    end;
Em Delphi (para interface visual)
    var a : integer;
    begin
      a := StrToInt(Edit1.Text);
      Label1.Text := IntToStr(a+1);
    end;
```

Nota-se que no código escrito para o Turbo-Pascal a variável a recebe diretamente o valor digitado pelo usuário e a linguagem envia o resultado para a posição corrente do cursor numa tela. Já no código em Delphi um objeto de texto recebe um valor e este é posteriormente transferido para um outro objeto, desta vez um para saída na forma de texto. Esses dois objetos devem estar posicionados em algum local de uma janela. Mesmo num programa bastante simples, sem consistência alguma, percebe-se facilmente a distinção existente entre o mesmo código escrito numa mesma linguagem, simplesmente causado pela interface distinta. Mesmo que a lógica usada para uma e outra solução seja idêntica, a forma de se trabalhar com a variável "a" exige uma capacidade de abstração maior na implementação Pascal feita pelo Delphi do que a feita em Turbo Pascal.

Por outro lado há a questão de se programar usando os conceitos da orientação a objetos, que ampliaria o escopo desse artigo a objetivos muito além do presente. Todavia, é um ponto a ser refletido com maior profundidade.

[2] DOS – Disk Operation System é a forma como os antigos sistemas operacionais em interface caractere são conhecidos. Foram largamente utilizados antes da adoção em larga escala da interface visual, pioneiramente lançada pela Apple, idealizada por Steven Jobs, seu fundador.

8. ENDENTAÇÃO OU "FIM-DO-SE"?

Nesse ponto cabe também a discussão da importância da estrutura "se~fim-do-se", não usada nesse texto. Não o foi, pois concordamos integralmente com autores como Salvetti e Barbosa (2001), que não utilizam essa estrutura. Para melhor nos fazermos entender, vamos apresentar duas soluções do mesmo problema, uma usando e outra não, a referida técnica.

Primeiramente será repetida a estrutura anteriormente apresentada destinada a apresentar o maior, entre três números distintos:

```
leia a, b, c
se a > b então
   se a > c então
      imprima a
   senão
     imprima c
senão
   se b > c então
      imprima b
   senão
     imprima c
```

Em seguida, o mesmo programa, agora com a inclusão da instrução fim-do-se

```
leia a, b, c
se a > b então
   se a > c então
      imprima a
   senão
     imprima c
   fim-do-se
senão
   se b > c então
      imprima b
   senão
     imprima c
  fim-do-se
fim-do-se
```

A instrução fim-do-se, que causa arrepios em programadores COBOL, C, Pascal e Java, se originou de uma característica, quase um erro, na montagem do interpretador do velho banco de dados (sic) dBase II. Esse produto híbrido, algo entre um banco de dados relacional e um gerenciador de arquivos, com uma linguagem interpretada de programação, fechava

as estruturas de tomada de decisão (e também as ainda não comentadas, de repetição) de forma explícita.

Duas linguagens de programação foram criadas para gerar programas executáveis capazes de trabalharem com os arquivos de dados gerados pelo produto dBase III, sucessor imediato do dBase II: o Clipper e o Fox Base. Essas linguagens não demoraram a se consolidar e rapidamente angariaram legiões de fãs espalhados pelo mundo. Inevitavelmente começou-se a escrever livros suportando essa linguagem e logo passamos a ter pseudo-códigos, ou seja, programas em português estruturado, escritos para esse ambiente.

Tal situação agravou-se ainda mais, quando uma das evoluções do BASIC acabou por adotar a mesma sistemática de terminação de suas instruções de tomada de decisão (if). Assim, graças as linguagens padrão xBase, como passaram a ser conhecidos os dialetos Clipper, Fox Base e inúmeros outros surgidos na esteira desses produtos e ao BASIC, consolidou-se a esdrúxula notação "se ~ fim-do-se". Por que tal notação é tão difícil de ser aprendida? Valendo-se novamente do inatismo chomskyano, talvez por não ser natural.

Alguém imaginaria uma criança permanecendo séria se seu pai lhe dissesse algo como "Juquinha, se você não guardar seus brinquedos ir apanhar, fim do se" ou se o chefe de alguém dissesse algo como "Paula, se tivermos dinheiro em caixa então pague o José, senão emita um cheque, e fim-do-se"...

O abandono desse tipo de notação, que não prescinde da endentação, permite ao estudante de uma linguagem de programação passar a criar sua estratégia da mesma forma que o faz para seus problemas habituais. Ou seja, tem sua mente aberta a criação de uma solução, e não escrava de uma técnica anti-natural.

9. ESTRUTURAS DE REPETIÇÃO

Contagens, somatórias, séries e toda espécie de problema que exija a inclusão de ao menos uma estrutura de repetição, exigem um grau de abstração do estudante distinto daquele usado até a chegada a esse ponto. Aqui começa fazer sentido o dito que diz ser o computador uma máquina "rápida, precisa, mas burra". A construção das estruturas "para~enquanto~faça~repita" não é algo natural. É fato de que qualquer um de nós diz no seu dia-a-dia "Se chover então pegue o guarda-chuva" ou "Se tenho 40 reais então compro o DVD senão compro o CD do show", mas raramente alguém dirá "Para i de 1 até 50 minutos faça o trabalho de lógica". Essa distinção exige toda sorte de técnicas para que o professor busque ensinar esses conceitos aos seus alunos, todavia alguns estudantes rapidamente entendem o conceito de repetição e passam quase que automaticamente a executar essas estruturas, enquanto que outros apenas com muito esforço começam a desenvolver suas primeiras tentativas em construir suas soluções.

Habitualmente, a série de Fibonacci[3] , de grande valia para que os alunos entendam o conceito de contagem, somatória e série, mas para isso é necessário que esses alunos tenham feito longas séries de exemplos e exercícios de contagens, séries e somatórias.

O algoritmo a seguir imprime os números de Fibonacci, que visto solucionado parece pouco complexo. Na matemática, os números de Fibonacci são uma sequência definida como recursiva pela fórmula abaixo:

{
0, se n = 0
1, se n = 1
f(n-1) + f(n -2), se n > 1
}

O algoritmo a seguir representa essa famosa sequência.

```
leia n
a = 0
b = 1
para i de 1 at, n - 2 faça
  c = a + b
  imprima c
  a = b
  b = c
```

Caso o professor apresente a solução diretamente aos alunos, grande parte deles senão sua totalidade não apresentará dúvidas. Todavia, basta inverter a situação, indicando um enunciado como "Dada a série 0,1,1, 2, 3, 5, 8, 13, 21, 34, ... imprima seus 'n' primeiros termos.", deixando espaço para os alunos, rapidamente alguns deles começarão a perceber detalhes que os levarão a resolver a questão. É certo que sem o domínio da técnica em se construir estruturas de repetição, pouco provável que se consiga criar um algoritmo que resolva o problema, mas também não, condição suficiente para que se resolva a questão simplesmente dominar a técnica de uso da instrução "para".

De fato, excetuadas às vezes em que alunos que já conheciam o problema de antemão o resolvem, no geral a solução da questão tende a ser dada por algum aluno que conhece a técnica construtiva da estrutura de repetição, mas não necessariamente que domine ou tenha fluência em programação.

Por outro lado, mesmo depois de simulações e de dominada a técnica de construção das estruturas de repetição, muitos alunos não conseguem resolver esse problema por mais que estudem e se esforcem, simplesmente decorando a solução.

[3] Esta sequência foi descrita primeiramente por Leonardo de Pisa, também conhecido como Fibonacci (1200 DC), para descrever o crescimento de uma população de coelhos. Os números descrevem o número de casais em uma população de coelhos depois de "n" meses.

Eliminados aqueles casos onde o professor acredita que o aluno esteja de fato estudando, mas isso não sendo verdade e somados aqueles em que o aluno parece esquecer a solução da questão, devido a alguma tensão ou outro problema, restam ainda muitos casos em que essa situação se verifica, por mais exercícios e exemplos que ao aluno sejam solicitados ou apresentados. Seria falta de uma aptidão específica? Seria ausência da pitada de arte?

Os números primos, além de todas as aplicações e usos em v rios ramos do conhecimento, também são um importante instrumento para identificação desse quê de talento que estamos a tratar.

Habitualmente, quando se formula esse problema numa sala de aula, a solução acaba surgindo não de um aluno, mas de um grupo deles. É até interessante, para finalidades didáticas que assim seja, pois esse tipo de problema fomenta muita discussão e desenvolvimento colaborativo.

Em geral, obtém-se uma solução aproximada em acordo com o algoritmo apresentado a seguir:

```
leia n
primo = Verdade
para i de 2 at, n-1 faça
  se n resto i = 0 então
     primo = Falso
se primo então
   imprima n ", primo"
senão
  imprima n "não , primo"
```

Cabe ao professor ter muita paciência, pois a solução em certas salas emerge em menos de cinco minutos enquanto que em outras não antes de trinta minutos. Todavia, qualquer que senha a turma a solução encontrada não difere muito do algoritmo acima.

Incitar os alunos a buscarem otimizações relativas ao intervalo de pesquisa, com colocações como "não há divisor de um número que supere sua própria metade!", "mais que isso, se um número não for primo, encontraremos algum divisor dele entre 2 e sua raiz quadrada!" leva ao surgimento de novas idéias. Contudo, não raro algum aluno descobre que, inútil persistir testando números após identificado ao menos um divisor do mesmo. Logo, outro aluno indica que realizar testes por números pares , inútil, pois qualquer número par é eliminado diretamente pelo dois. Temos então o seguinte algoritmo:

```
leia n
primo = Verdade
i = 2
enquanto i <= raizquadrada(n) e primo faça
```

```
    se n resto i = 0 então
       primo = Falso
    se i = 2 então
       i = i + 1
    senão
      i = i + 2
  se primo então
     imprima n ", primo"
  senão
    imprima n "não , primo"
```

Mesmo que o primeiro algoritmo j não seja óbvio a todos os alunos, em geral depois de uma ou outra explicação, todos parecem entendê-lo. Já sua otimização não apresenta a mesma característica, mesmo que cuidadosamente se siga o critério de nunca fazer mais de uma otimização por vez. Novamente parece existir uma barreira invisível a separar grupos de alunos. O esforço de alguns despidos de talento contrasta com o pouco empenho de outros que possuem mais talento, mas que em razão dessa falta de denodo acabam por não compreender todos os detalhes de uma solução como a versão otimizada. Não obstante, estes últimos acabam entendendo a segunda solução, enquanto que os primeiros não têm a mesma sorte.

Nessa fase dos cursos, há uma seleção natural com vários alunos desistindo do curso. Não h uma relação que possa ser comprovada, até porque no primeiro semestre de qualquer curso muitos alunos vão desistindo ao longo do semestre pelos mais variados motivos, mas as estruturas de repetição parecem abrir um fosso entre dois perfis de alunos: aqueles que concluirão o curso, daqueles que irão desistir. É certo que toda uma série de fatores faz com que muitos capacitados a concluírem um curso ligado ao desenvolvimento de sistemas ficam pelo caminho motivados por toda sorte de problemas, contudo , bastante raro algum aluno que não consegue compreender estruturas de repetição permanecer no curso de sistemas.

Para as escolas e para os professores esse não chega a ser um problema importante, pois podemos realizar exercícios a exaustão sobre esse tema, sem descontentar aqueles que têm talento, até porque estarão exercendo-o, além de capacitar quase que todos os demais alunos no domínio da técnica de construção de estruturas de repetição.

Naturalmente, o mero domínio dessa técnica não capacitar ninguém a ser desenvolvedor de sistemas, quando muito tornará o aluno capaz de sair-se bem numa avaliação ou mesmo num teste numa empresa, mesmo que dificilmente venha a se tornar um desenvolvedor notável.

10. ESTRUTURAS DE DADOS, AS MATRIZES

Em geral, as faculdades separam a disciplina Estrutura de Dados da disciplina Lógica de Programação. Faz sentido, pois no exíguo espaço de um semestre não é provável que um aluno normal seja capaz de dominar tantas técnicas simultaneamente. Todavia, o intervalo para férias se torna um poderoso desafio aos professores de lógica, pois para que um aluno entenda os conceitos dos vetores, como também são conhecidas as estruturas de dados unidimensionais, ele deve dominar todo conjunto de conceitos vistos em estrutura de dados.

Embora tenhamos usado diferentes técnicas e grupamento de exercícios ao longo dos anos, esse tópico tende a ser um divisor de águas, mostrando aos professores o quanto somos impotentes para levar alguém, sem a devida capacidade, a conseguir de fato dominar os conceitos de vetores, matrizes e das estruturas de dados.

De maneira até irritantemente previsível, aqueles alunos que dominaram as técnicas ligadas as estruturas de repetição, lidam com os vetores com muita naturalidade. Em alguns deles torna-se possível perceber uma vocação mais acentuada para a área de dados do que para programação, mas uns e outros tendem a não terem dificuldades na construção de estruturas de programação que envolvam estruturas de dados.

Por outro lado, aqueles alunos que já havia sofrido bastante com as estruturas de repetição, aqui tendem a encontrar um obstáculo, em geral, final. O conceito de vetores exige que o grau de abstração seja muito mais amplo do que o exigido nas tomadas de decisão e nas estruturas de repetição. A matemática, em geral, prepara matrizes mais para cálculo de determinantes ou para se inverter matrizes, o que já leva aqueles estudantes que vivem "de mal" com a matemática a se assustarem. Não deviam, pois a matem tica e a lógica de programação são primas afastadas, no que tange ao conjunto de técnicas que utilizam, mas o fato de não se calcular a matriz cofatora, não ser de grande valia, pois a mera ordenação de um vetor apresenta tantos algoritmos, que só especialistas no tema conhecem.

Ziviani (1999, p 69) afirma que "as técnicas de ordenação permitem apresentar um conjunto amplo de algoritmos distintos para resolver uma mesma tarefa.". De fato, esse auto apresenta v rios m,todos como m,todo da Seleção, da Inserção, de Shell e o Quicksort, descrevendo e explicando cada um deles. É interessante ressaltar que v rios estudantes, a partir da apresentação de um desses m,todos ou dos também conhecidos Buble ou Shake descubram em publicações ou pela internet, o que atualmente é bem comum, vários outros algoritmos e rapidamente comecem a discutir desempenho e outros detalhes técnicos. Em geral, tendem a superar a melhor expectativa dos seus professores.

Por outro lado, aqueles estudantes que com muito esforço haviam vencido o obstáculo imposto pelas estruturas de repetição, entram num estado de frustração absoluta, pois veem alguns colegas que meses antes tinham as mesmas dificuldades que eles, tão fluentes em algo que esses alunos parecem não terem a menor chance de um dia virem a compreender.

Uma maneira interessante de se fazer a transição entre variáveis convencionais e vetores é o conjunto de exemplos relativo a construção de um programa que receba cinco números e os exiba na ordem oposta a que foram fornecidos. As três soluções mais comuns aparecem descritas a seguir:

Solução 1

```
leia a, b, c, d, e
imprima e, d, c, b, a
```

Solução 2

```
leia a1, a2, a3, a4, a5
imprima a5, a4, a3, a2, a1
```

Solução 3

```
para i de 1 at, 5 fa‡a
  leia a[i]
para i de 5 at, 1 passo -1 fa‡a
  imprima a[i]
```

Basta agora enunciar "e se em vez de cinco, não soubermos quantos números teremos?", para invariavelmente algum aluno responde bastar lermos "n" e o substituirmos pelo cinco da terceira solução, mas outros tentam de várias maneiras solucionar a questão tentando de alguma forma alterar a primeira solução. Ora, numa fase dessas num curso de desenvolvimento de sistemas, isso poder parecer ao professor menos experiente pouco caso ou falta de atenção do aluno. Ledo engano. Em se observando alunos ao longo dos anos, acaba se percebendo que justamente aqueles alunos que estudam muito e que tiveram grande dificuldade em dominar cada tópico anteriormente visto, são geralmente os mesmos que apresentam maiores dificuldades na percepção que a primeira solução não pode ser invocada para resolver o novo problema. Além disso, parecem não perceber o elo que a segunda solução é entre as outras soluções.

Por outro lado, ocorre justamente o inverso em relação aos estudantes que haviam conseguido entender as estruturas de repetição rapidamente. Estes juntam rapidamente aos seus conhecimentos aqueles relativos às estruturas de dados, quase como se fosse algo familiar e que lhes havia sido suprimido. É interessante observar que muitos alunos, ao ouvirem de outros colegas que sabem programar que há uma forma distinta de solucionar deteminado problema usando matrizes, o que é especialmente verdadeiro no caso do algoritmo de Fibonacci, que consta na Wikipédia em solução fazendo uso de vetores. Muitos alunos acabam por aprenderem sozinhos matrizes como que a comprovarem a frase não tão agradável aos professores, atribuída ao polêmico Chomsky, a quem os professores mais atrapalham que ajudam...

11. CONCLUSÃO

Muitas das indagações desse texto refletem a longa experiência ministrando lógica de programação em sala de aula. Certamente seriam necessárias extensas tabulações estatísticas para se afirmar que Lógica de Programação tem um quê de arte ou que exige algum talento inicial. Sabemos, entretanto, que qualquer que fosse o resultado de um estudo assim, sempre estaria envolto em discussões, pois qualquer que fosse a amostra de estudantes escolhida, sempre essa amostra poderia ser contestada.

Sabedores que somos da necessidade que nosso país tem em capacitar muito mais gente para desenvolver sistemas e vendo a dificuldade que nossas empresas têm em transformar em profissionais aqueles estudantes que nossas escolar formam e colocam no mercado de trabalho, muitos dos quais embora sejam opções para um mercado carente de profissionais continuem sem colocação na área, tentamos nesse artigo apresentar um ponto a mais a ser discutido para quem deseja, como nós, capacitar estudantes para o mercado de desenvolvimento de sistemas.

Nossa conclusão, contudo, é otimista. Embora seja verdade que um talento acima da m,dia permita a remoção de obstáculos imensos sem máximo esforço, não deixa também de ser verdade que muito esforço compensa certa falta de talento. Todavia, a questão da vocação não pode ser sumariamente desconsiderada.

REFERÊNCIAS

BUNGE, M. *El problema mente-cerebro: Un enfoque psicobiologico.* Madrid: Tecnos, 1988

FORBELLONE, A.L.V.; EBERSPÄCHER, H.F. *A Construção de Algoritmos e Estruturas de Dados*. São Paulo: Pearson Prentice Hall, 2005.

GANE, Chris, SARSON; Trish. *Análise Estruturada de Sistemas.* Rio de Janeiro: Livros Técnicos, 1983.

MANZANO, J.A.N.G.; OLIVEIRA, J.F. *Algoritmos - Lógica para Desenvolvimento de Programação.* São Paulo: Érica, 1996.

PIAGET, Jean . *A psicogênese dos conhecimentos e a sua significação epistemológica.* em M. Piattelli-Palmarini (Org.), Teorias da linguagem, teorias da aprendizagem: debate de Jean Piaget e Noam Chomsky com outros autores

PIATELLI-PALMARINI, Massimo. *Teorias da linguagem, teorias da aprendizagem: o debate entre Jean Piaget e Noan Chomsky*. São Paulo: Cultrix, 1983.

SALVETTI, D.D.; BARBOSA, L.M. *Algoritmos.* São Paulo: Makron Books, 2001.

ZIVIANI, N. *Projeto de Algoritmos com implementações em Pascal e C.* São Paulo: Pioneira, 1999.

Turbo-Pascal e Delphi são marcas da Borland Inc.

Windows é marca registrada da Microsoft Inc.