Prefácio

Na Licenciatura de Engenharia Informática e de Computação, da Faculdade de Engenharia da Universidade do Porto *(FEUP)*, a utilização do *Scheme* como primeira linguagem no ensino da programação surgiu, no ano lectivo 1994/95, precisamente no ano em que este curso se iniciou. Esta linguagem já tinha sido introduzida na FEUP, no ano lectivo 1991/92, na Licenciatura de Gestão Industrial, mas desconhece-se o seu uso noutras Escolas do País, com a excepção do Instituto Superior Técnico, da Universidade Técnica de Lisboa, na Licenciatura de Informática, a partir do ano lectivo 1998/99. Nos Estados Unidos da América, o *Scheme* começou a ser utilizado no início da década de 80 (universidades de MIT, Yale e Indiana), onde tem conquistado inúmeros adeptos nas melhores universidades e, não é de estranhar que, em 1995[1], já ocupasse a quarta posição entre as linguagens que serviam de suporte ao ensino da introdução à programação, com uma quota de 11%, atrás do *Pascal* (35%), *Ada* (17%), e *C/C++* (17%), mas com um crescimento superior a qualquer uma destas. Em 1998, 250 estabelecimentos de ensino superior em todo o mundo utilizavam *Scheme*, 100 deles em cursos de introdução à programação. Nos Estados Unidos e na mesma data, aqueles números eram, respectivamente, 140 e 47[2]. O entusiasmo por esta linguagem atinge também os alunos, que se organizam em grupos, como acontece com os *Schememonster's Friends* da Universidade de Helsínquia, que se propõem desenvolver projectos de média/grande complexidade com o *Scheme*, como se pode ver em:

http://www.niksula.cs.hut.fi/~candolin/scheme/

O *Scheme*, apesar de ser uma linguagem poderosa, caracteriza-se por ter uma sintaxe simples, com poucas regras, que não exige nem muito tempo *(do aluno)* nem muito espaço *(de manuais)*, o que já não se poderá dizer da generalidade das restantes linguagens. Assim, o esforço e atenção podem incidir, sobretudo, sobre a *programação*. Programar significa encontrar ideias para resolver problemas, pô-las em movimento, com implementações elegantes, legíveis e, tanto quanto possível, eficientes, em tempo *(de cálculo)* e em espaço *(de memória*). Incentivar os alunos a criar ideias é o grande objectivo e, neste sentido, o *Scheme* integra-se de uma forma excepcional neste plano. O facto de não oferecer funcionalidades para certas tarefas, como sejam algumas das estruturas habituais de controlo *(como os ciclos)*, ou a introdução tardia do conceito de afectação *(dados mutáveis)*, o que é visto inicialmente pelos alunos como graves limitações da linguagem, traduz-se num bem precioso, pois este tipo de limitações acaba por retirar ao aluno a hipótese de recorrer à *programação remendada*, induzindo a necessidade de soluções estruturadas, legíveis e propícias a futuras alterações.
O *Scheme* é uma linguagem funcional derivada do *LISP*, mas muito mais simples do que esta. É uma linguagem praticamente desconhecida dos alunos que entram na universidade, alguns já com largos conhecimentos de *linguagens*, mas poucos com bons conhecimentos de *programação*. A

---

[1] Findler, Flanagan, Flatt, Krishnamurthi, and Felleisen
DrScheme: a pedagogic programming environment for Scheme
Proc. 1997 Symposium on Programming Languages: Implementations, Logics, and Programs

[2] Segundo compilação de Ed Martin, Schemers Inc.
(emartin@schemers.com)

programação envolve procedimentos e dados[3]. Aos procedimentos são associadas tarefas específicas e a cada um deles é ligado um nome. Depois, estas tarefas serão postas em movimento, chamando pelos nomes desses procedimentos. Trata-se da *Abstracção Procedimental* que evita o conhecimento dos pormenores dos procedimentos. Para cada um destes bastará conhecer o seu nome e o que faz, mas não será necessário saber como funciona. Um procedimento comporta-se como um *bloco* ou *caixa-preta* que sabemos utilizar, sabemos para que serve, mas não temos que conhecer o seu interior. A *Abstracção Procedimental* virá em primeiro lugar, seguida pela *Abstracção de Dados*. Pegar em dados simples e compor, a partir deles, dados que se adeqúem à representação de cada situação real é uma tarefa fundamental da programação, e que não poderá ser menosprezada.

É importante referir a necessidade de fazer uma aprendizagem da programação PROGRAMANDO. Um computador e um interpretador de *Scheme* são ferramentas excelentes não só para testar os exemplos que vão sendo apresentados, mas também para ajudar a encontrar soluções alternativas para esses exemplos, e para apoiar na criação de ideias novas utilizadas na implementação dos muitos exercícios que vão sendo propostos. Alguns destes exercícios apresentam-se como problemas de média complexidade e por isso podem exigir várias horas ou até dias de trabalho. É um esforço grande, mas que se recomenda e os resultados serão certamente muito gratificantes. *E não esquecer que se aprende muito mais quando as soluções não funcionam à primeira... nem à segunda...*

Os exemplos foram testados numa versão comercializada do *Scheme*, o *EdScheme*, da *Schemers Inc.*, mas existem versões gratuitas e de boa qualidade com origem universitária, nomeadamente, o *MIT Scheme*, e o *DrScheme*, esta última disponível para plataformas *Unix* e *Windows*. Não só com o objectivo de encontrar software, aconselha-se a exploração dos seguintes endereços:

http://www.cs.indiana.edu/scheme-repository/home.html *(The Internet Scheme Repository)*
http://www-swiss.ai.mit.edu/scheme-home.html
http://www.yahoo.com/Computers_and_Internet/Programming_Languages/Scheme/ *(Yahoo links)*

Bibliografia

Este livro surge da experiência de leccionação, com o *Scheme*, na disciplina *Introdução à Programação I*, da Licenciatura de Engenharia Informática e de Computação, da Faculdade de Engenharia da Universidade do Porto *(FEUP)*, iniciada no ano lectivo 1994/95. Tratou-se de uma experiência apaixonante que parece querer continuar a sê-lo. Neste percurso, alunos e docentes foram acompanhados, entre outros, por dois livros excepcionais, de inquestionável valor e que merecem, para além de uma leitura atenta de todos os que se interessam pela *programação*, uma referência que também pretende ser uma homenagem singela aos seus autores:

H. Abelson, G. J. Sussman, J. Sussman
Structure and Interpretation of Computer Programs
McGraw-Hill Book Company, 1985, second edition, 1996

George Springer and Daniel P. Friedman
Scheme and the Art of Programming
McGraw-Hill, Sixth printing, 1993

No percurso que se segue, os vários intervenientes, alunos e docentes, poderão ainda contar com o livro que agora se coloca ao julgamento do leitor, e que não pretende substituir os acabados de referir. As suas características mais relevantes são as seguintes:

[3] Veremos noutro capítulo que, em Scheme, os procedimentos podem também ser tratados como dados, característica eventualmente perturbadora, mas com um grande potencial.

⇒ adopção de *lambda* na definição de procedimentos
Este facto permite que os alunos se familiarizem, desde o início, com a funcionalidade associada ao procedimento primitivo *lambda*, facilitando ainda a entrada no capítulo que considera os procedimentos como *objectos de 1ª classe* e no que introduz a *programação orientada por objectos* em *Scheme*;

⇒ cobertura de um espectro largo de assuntos com recursos limitados de tempo e espaço
Apesar de se cobrirem assuntos que vão desde uma abordagem inicial à linguagem *Scheme* até uma *introdução à programação orientada por objectos*, passando pela *recursividade*, por várias *abstracções*, por um pequeno conjunto de *procedimentos gráficos* e pelos *ficheiros*, tudo isto se pode conter no âmbito de uma disciplina semestral e em cerca de duas centenas de páginas. São ainda apresentados, em todos os capítulos, numerosos exemplos, exercícios e pequenos projectos;

⇒ capítulo final com enunciados de exercícios e projectos
Os exercícios e projectos de fim de capítulo ficam muito associados às matérias nele contidas, coarctando, em parte, a imaginação de quem os tenta resolver. Com um capítulo final de enunciados não existe, à partida, uma associação entre os exercícios e projectos e qualquer tema específico, e assim a margem de actuação será muito maior e, consequentemente, muito mais rica;

⇒ resumo com os principais procedimentos do *Scheme*
Um resumo com os principais procedimentos do Scheme, ilustrado com exemplos, é disponibilizados sob a forma de anexo *(Anexo A)*[4];

⇒ pequeno conjunto de procedimentos gráficos
É disponibilizado um pequeno, simples e flexível conjunto de procedimentos gráficos, em duas versões, uma desenvolvida sobre as primitivas gráficas do *EdScheme* e outra do *DrScheme*, que funcionam numa janela gráfica definida pelo programador *(Anexo B)*. Evita-se assim um dispêndio exagerado de tempo no estudo de um conjunto normalmente extenso e complexo das primitivas gráficas que acompanham o *Scheme* que cada um utiliza. Se este não coincidir com o *EdScheme* ou com o *DrScheme*, recomenda-se, como exercício, a introdução das adaptações necessárias às primitivas gráficas do *Anexo B*.

⇒ soluções de exercícios e projectos
Para alguns dos exercícios e projectos propostos apresenta-se uma solução no *Anexo C*. Convidam-se os interessados a enviar[5] soluções para os exercícios não incluídos neste anexo ou soluções alternativas de melhor qualidade do que as apresentadas. O material recebido, depois de analisado, será disponibilizado com referência aos autores, no endereço:

http://www.fe.up.pt/~fnf/leic/ip1/AnexoSolu.pdf

Panorâmica dos vários capítulos do livro

O texto está estruturado em 7 capítulos e 3 anexos e pretende-se, neste momento, dar uma panorâmica dos temas tratados em cada um deles.

O *Capítulo 1* faz uma breve introdução à linguagem *Scheme*, considerando os *números*, *expressões*, *símbolos*, *definição de procedimentos* e *estruturas de selecção*. Para além deste capítulo, a sintaxe do *Scheme* resumir-se-á, fundamentalmente, a notas pontuais ao longo de outros capítulos e a referências a um dos anexo *(Anexo A)*.

O *Capítulo 2* considera o conceito da *recursividade* e a definição de *procedimento recursivo*, procedimento que se chama a si próprio. É através deste tipo de procedimento que se

[4] Para uma descrição mais aprofundada do Scheme, consultar William Clinger and Jonathn Rees, *Revised (4) Report on the Algorithmic Language Scheme*, disponível em *http://www.cs.indiana.edu/Scheme-repository/R4RS/r4rs-toc.html*
[5] fnf@fe.up.pt (Subject: Programar com Scheme)

ultrapassa, no *Scheme*, a ausência dos ciclos que existem na generalidade das outras linguagens. Mas a *recursividade* é muito mais rica do que isto, pois habilitar-nos-á a encontrar ideias para resolver problemas, alguns de grande complexidade, e que de outra maneira não seriam fáceis de resolver. As chamadas dos procedimentos geram *processos* no computador. A noção de *processo*, seja ele *recursivo* ou *iterativo*, é também considerada, sendo ainda analisada a forma como consome recursos computacionais, *tempo* de CPU e *espaço* de memória. Estes recursos consumidos são medidos através da *Ordem de crescimento*. O capítulo termina mostrando que os procedimentos podem ser definidos como *blocos* ou *caixas-pretas*, sendo ainda apresentadas as vantagens e desvantagens que daí resultam.

O *Capítulo 3* mostra como, a partir de dados simples, se compõem os dados que se adequam à representação dos problemas que se pretendem resolver. Por exemplo, num problema de características gráficas a 2 dimensões *(2D)*, a criação da entidade *ponto* é perfeitamente justificável, sendo constituída por dois valores numéricos que representam as suas coordenadas *x* e *y*, no plano. Para este caso, será necessário agrupar dois valores numéricos criando objectos do tipo *ponto-2D* constituídos, por exemplo, por um *par* ou por uma *lista de dois elementos*. Para compor os dados utilizam-se procedimentos *construtores* e o acesso aos elementos dos dados compostos é realizado com os procedimentos *selectores*. Com os dados compostos e os respectivos construtores e selectores surge a *Abstracção de dados*, cuja potencialidade irá ser posta em relevo quando se apresentar a noção de *Barreira de abstracção*. A abstracção *conjunto* será desenvolvida e estudada com alguma profundidade. As primitivas gráficas, que se encontram no *Anexo B*, começam a ser utilizadas em exemplos e exercícios deste capítulo. A complexidade dos problemas poderá exigir uma análise que identifique as suas principais tarefas e destas, as que ainda forem consideradas complexas, também serão sujeitas a uma análise idêntica. E o processo repete-se, enquanto a complexidade de alguma tarefa o exigir. Trata-se da abordagem *de-cima para-baixo*, que decompõe o problema em problemas cada vez mais simples. Ainda no *Capítulo 3*, tentar-se-á mostrar que esta abordagem e a *abstracção de dados*, esta inspirada numa abordagem *de-baixo-para-cima*, constituem vias complementares e não alternativas.

Os *objectos de 1ª classe* são, normalmente, os operandos *(números, booleanos, pares e símbolos)*, entidades que podem ser processadas e, por isso, reconhecidas como objectos *passivos*. Os procedimentos são objectos *activos*, pois associam-se-lhes actividades de processamento. As propriedades dos objectos que os caracterizam como os objectos de 1ª classe são: 1- Ligam-se a símbolos; 2- são passados como argumentos de procedimentos; 3- são devolvidos como valores de retorno de procedimentos; 4- são elementos de estruturas compostas de dados. O *Capítulo 4* considera o tema *Procedimentos como objectos de 1ª classe*, centrando a atenção na demonstração de que os *procedimentos podem ser o valor de retorno de outros procedimentos*. Outro aspecto interessante, também considerado, é a definição de *procedimentos com um número não fixo de argumentos*.

O *Capítulo 5* mostra que, para além dos *construtores* e *selectores*, também existem *modificadores*. Começam por surgir as chamadas *listas mutáveis* que estarão na base da implementação das abstracções *Filas-de-espera* e *Tabelas*. Os *Vectores* e as *Cadeias de caractares*[6] são abstracções de dados mutáveis directamente disponibilizadas pelo *Scheme*, e mostra-se o que trazem de novo, em relação às restantes abstracções. Perante várias abstracções, as fornecidas pela linguagem e as criadas posteriormente, poderá não ser fácil decidir sobre qual será a melhor em cada situação ou se não será necessário criar uma nova. Ao finalizar o capítulo, introduz-se o tema *ficheiros*, devido à necessidade de guardar em disco dados de uma sessão para outra sessão de trabalho.

O grande interesse da programação *orientada por objectos* reside, em grande parte, no reaproveitamento de código que este paradigma de programação oferece e na modularidade que apresenta, pois cada objecto surge como se fosse um módulo com os seus procedimentos e dados, perfeitamente encapsulados. O *Scheme* não é, propriamente, uma linguagem vocacionada para a programação orientada por objectos, característica que é mais notória em linguagens como o *Java*, *C++*, *Smalltalk* ou *Simula*, mas permite, com relativa facilidade, introduzir os

[6] string

conceitos deste importante paradigma de programação. O *Capítulo 6* constitui uma abordagem simples à *programação orientada por objectos* em *Scheme*. São intuitivamente introduzidos os conceitos de *Objecto*, *Mensagem*, *Método*, *Classe*, *Herança*. Vários exemplos de *classes* são apresentados e desenvolvidos, nomeadamente, *Classe-fila*, *Classe-pilha* e *Classe-lista-circular*.
O *Capítulo 7* não é mais do que um conjunto de enunciados de exercícios e projectos, poucos com pistas de solução, mas de forma a colocar o leitor perante problemas de programação completamente desligados deste ou daquele capítulo. Quando se coloca um exercício no final de um capítulo, uma primeira pista de solução fica implicitamente estabelecida. Na vida real, os problemas de programação não aparecem rotulados ou associados a capítulos de qualquer livro... Perante um problema, torna-se necessário encontrar uma ideia para o resolver, vaga no início, mas sucessivamente mais refinada. Só à medida que as várias tarefas e sub-tarefas do problema vão sendo identificadas, é que se clarificam as associações com os capítulos e matérias tratadas em cada um deles. Surge a forma de representar os dados do problema, acompanhada de um certo conjunto de construtores, selectores e até modificadores, ou seja, surge a *abstracção de dados*. E é sobre a abstracção idealizada, com as tarefas e sub-tarefas identificadas que se refina a construção da solução.

O *Anexo A* apresenta um resumo com os principais procedimentos do Scheme, ilustrados com exemplos. Um conjunto de procedimentos gráficos surge *no Anexo B*, que ainda contém a implementação dos mesmos para *EdScheme* e *DrScheme*. Finalmente, no *Anexo C* encontram-se soluções para alguns dos exercícios propostos.

Projectos propostos

A aprendizagem da programação deverá ser sistematicamente praticada no computador, o que justifica a introdução no texto de numerosos exemplos, exercícios e projectos de pequena/média dimensão. No caso particular dos projectos, procurou-se que fossem particularmente atractivos, pois exigem algum tempo e esforço e também muita imaginação. É um desafio que se coloca e que transmitirá, a quem o vencer, um elevado grau de autoconfiança e gosto pela programação.
No início dos trabalhos de um curso que siga as matérias incluídas no livro, os projectos deverão ser brevemente apresentados e, alguns deles, demonstrados[7], para mostrar aos alunos o que poderão fazer com a programação e com o *Scheme*. No início de cada capítulo, um dos seus projectos deverá ser demonstrado com algum pormenor e, no final, analisado e desenvolvido, pelo menos, no que se refere às partes mais relevantes. Os alunos deverão ser desafiados a desenvolver as suas próprias versões do projecto e até a criar variantes do mesmo ou a definir projectos novos. Este tipo de atitude estimula o gosto dos alunos pelo estudo e, particularmente, pela programação.
Dada a importância desta matéria, apresenta-se agora uma breve panorâmica dos projectos incluídos no texto. O Capítulo 1 é o único que não oferece qualquer projecto, mas apenas pequenos exercícios, devido ao ainda muito limitado domínio da programação e do próprio *Scheme*. No Capítulo 2 é introduzido o projecto *Caminho*, um labirinto simplificado, onde se simula um tabuleiro matricial, em que algumas das células contêm obstáculos, e é necessário procurar uma saída. No Capítulo 3 surgem três propostas de projecto: um primeiro com elevado nível de dificuldade, designado por *Colorir Mapas*, relacionado com o problema de colorir um mapa de países com um número limitado de cores, evitando que a mesma cor seja utilizada para colorir países com fronteiras adjacentes; um segundo projecto, o *Jogo das minas*, simula um terreno de minas, em variadíssimas situações, desde as minas muito ricas às muito pobres. De acordo com uma forma original de se deslocar no terreno, um mineiro vai visitando as minas,

[7] Recorda-se que a maioria dos projectos têm uma solução no *Anexo C*

recolhendo (nas ricas) ou perdendo (nas pobres) riqueza; o terceiro projecto do Capítulo 3, *Lançamento-de-projécteis*, reveste a forma de um jogo electrónico com interface gráfica, onde se procura alcançar, com um projéctil, um objecto colocado aleatoriamente no espaço, para testar a pontaria de quem joga. Trata-se de um projecto com visualização gráfica animada. O *CODEC* é o projecto introduzido no Capítulo 4 que trata, de uma forma muito simplificada, o problema da codificação e descodificação de mensagens. O Capítulo 5 surge com dois projectos: *Adivinha-palavras* é um jogo que desafia o utilizador a adivinhar palavras, fornecendo o comprimento da palavra a adivinhar e informando se as letras que vão sendo indicadas pelo jogador fazem ou não parte da palavra e, em caso afirmativa, onde se posicionam dentro da palavra; o segundo projecto do Capítulo 5 permite o controlo, através do teclado, de um helicóptero imaginário. O Capítulo 6 também apresenta dois projectos, ambos com implementações orientadas por objectos: *Dados* é um jogo electrónico de dados, relativamente simples, e *Vidas* é também um jogo electrónico, mas bastante mais complexo, que simula o percurso de uma criatura num tabuleiro, designado por *o-Vidas*, que muda de trajectória quando embate nas paredes e em criaturas distribuídas aleatoriamente no tabuleiro. Algumas dessas criaturas são *simpáticas*, e contemplam *o-Vidas* com prendas, outras são *antipáticas*, e retiram-lhe alguns haveres, finalmente outras são *neutras* e retiram-lhe metade dos haveres amealhados até essa altura. É claro que se o montante de haveres for negativo, *o-Vidas* só terá a ganhar com isso..., pois retira-lhe metade da dívida. A visualização gráfica de *o-Vidas*, obtida com as reflexões nas paredes e nas outras criaturas, é, simplesmente, divertida... São vários os projectos incluídos no Capítulo 7, mas apenas se distinguem dois: *Caminho-mais-curto*, uma espécie de labirinto em que os percursos são caracterizados pelo respectivo comprimento e procura-se o caminho mais curto entre um ponto à escolha e uma saída; *Labirinto*, relacionado com uma das várias técnicas de resolução de labirintos.

Material de apoio à leccionação

Para facilitar a apresentação das matérias expostas, foi disponibilizado um conjunto de acetatos no endereço:

http://www.fe.up.pt/~fnf/leic/ip1/acetatos.ppt

Fernando Nunes Ferreira
Setembro, 1999