A mesma velha questão: como ensinar Programação?

Pedro Guerreiro
Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
P-2825 Monte de Caparica, Portugal
email: pg@di.fct.unl.pt
telefone: +351 1 294 8536
fax: +351 1 294 8541

Resumo

Durante os últimos tempos, a questão de como ensinar a Programação tem estado um tanto adormecida. É como se o assunto tivesse ficado arrumado quando a programação estruturada foi descoberta e o Pascal inventado. No entanto, as ideias prevalecentes sobre a programação evoluíram, o panorama das linguagens de programação é muito diferente do de há vinte anos, os estudantes que hoje aparecem nas universidades pertencem a outra geração. Devemos nós continuar a usar os mesmo métodos, que têm resultados garantidos pela experiência, ou arriscar novas atitudes? Se arriscarmos, se quisermos usar hoje a programação orientada pelos objectos com a naturalidade com que antes abraçámos a programação estruturada, que linguagem devemos escolher? O Smalltalk, tão puro mas tão estranho, o Eiffel, tão bem construído mas tão difícil de obter, o C++ tão mal-visto mas tão popular?

Se nenhuma destas linguagens apresenta vantagens decisivas, o C++ tem a seu favor o ser a mais requisitada. Contra si tem o ser a mais complicada. No entanto, se o ensino for feito cuidadosamente, aprender C++ pode ser uma experiência muito interessante e muito enriquecedora.

1 Introdução

A programação como a praticamos começou com o primeiro compilador de FORTRAN, há quase quarenta anos. Aparentemente, no início, ensinar Programação não era uma questão que preocupasse demasiado os praticantes. O que era preciso era programas que funcionassem nas máquinas existentes, com os compiladores disponíveis. Nesses tempos, o método era o "code it now, fix it later", e não havia muito para aprender em Programação além de dominar a linguagem usada e as peculiaridades do computador onde trabalhávamos.

A situação só mudou com o advento da programação estruturada, no final dos anos 60, sobretudo por influência de Dijkstra, para quem ensinar a programar era essencialmente ensinar a pensar [Dijkstra 72A]. Trata-se de uma ligação muito ousada, mas, se quisermos evitar uma discussão filosófica que não é a nossa, podemos aceitar que ensinar a programar é, pelo menos, ensinar a pensar acerca dos programas.

A linguagem da programação estruturada foi, e é, o Pascal. Foi através do Pascal que as ideias da programação estruturada entraram nos hábitos dos programadores, a ponto de hoje toda a gente as aceitar pacificamente e naturalmente. E, se é costume que as linguagens conservem os sectores de desenvolvimento para que foram criadas, mesmo depois de ter passado o auge da sua popularidade (o FORTRAN continua muito usado em aplicações científicas, o COBOL em aplicações comerciais, o Ada em aplicações militares e de telecomunicações), é de esperar que o Pascal se mantenha como o veículo principal do ensino da Programação, mesmo que outras linguagens pudessem em tese ser mais adequadas para isso.

No entanto, a programação estruturada clássica, aquela que inspirou o Pascal, já não é o paradigma único, talvez nem sequer o dominante, e a função do Pascal fica prejudicada. Com efeito, actualmente programar é programar com objectos, assim como há vinte anos programar era

programar estruturadamente. Ora o Pascal, na sua forma ISO, não serve para a programação orientada pelos objectos [Jensen, Wirth 88].

Para mais, a programação estruturada considerava a programação como uma actividade individual, o que é compreensível, tendo em conta a época em que surgiu. Hoje no entanto, se encararmos mais abrangentemente a Programação não apenas como a actividade de escrever programas, mas como a de desenvolver software, vemos que ela tem que ser compreendida sobretudo como uma actividade de equipa, inserida na disciplina de Engenharia de Software. E, mais uma vez, o Pascal não é adequado para o trabalho em equipa.

Surge então a dupla questão: quando a nossa missão é ensinar Programação a futuros engenheiros de software, que devemos ensinar, e que linguagem devemos usar?

É essa questão que vamos tratar a seguir. Começamos por uma breve revisão do ensino da programação estruturada (Secção 2), e da utilização do Pascal nesse contexto (Secção 3). Depois analisamos as necessidades do ensino da programação orientada pelos objectos (Secção 4) e o modo como tentamos dar-lhes resposta no curso de Programação que actualmente leccionamos (Secção 5). Como a linguagem que usamos, o C++, não é geralmente considerada boa para ensinar Programação, discutimos os prós e os contras da nossa escolha (Secção 6). Finalmente, explicamos alguns dos aspectos técnicos da nossa abordagem (Secção 7).

2 Programação estruturada

Todos os programadores sabem o que é a programação estruturada, ou se não sabem, pelo menos estão convencidos que é isso que praticam. Descontando as confusões acerca dos gotos, programação estruturada é essencialmente o mesmo que programação por refinamentos sucessivos (*stepwise refinement*) [Wirth 71]. A ideia é que para programar se faz como Descartes mandou: "dividir cada um dos problemas em tantas partes quantas as necessárias para melhor os resolver." Em Pascal, a decomposição é funcional, ou procedimental, quer dizer, ao dividir o problema procuramos funções e procedimentos. A natureza monolítica do Pascal era adequada para esta abordagem. Uma linguagem em que as funções e procedimentos pudessem ser agrupados em módulos traria uma outra dimensão de complexidade ("como dividir um sistema em módulos?" [Parnas 72]) indesejável quando a preocupação era sobretudo a resolução do problema, sem envolver considerações de engenharia de software.

É verdade que, de abstracção em abstracção, muita gente advoga que ao ensinar Programação, devemos mesmo abstrair-nos da linguagem e concentrar-nos nos aspectos de resolução de problemas. A linguagem de programação torna-se assim um pormenor secundário. Quem pensa deste modo, usa descontraidamente uma pseudo-linguagem informal para exprimir os algoritmos que vão aparecendo. É esta atitude que detectamos quando nos currículos de cursos de programação de algumas universidades encontramos, depois da enumeração dos tópicos a tratar (funções, vectores, ficheiros, etc.), vem a indicação subsidiária de que a *linguagem-veículo* é a linguagem X ou Y. A ideia é que se poderia mudar de linguagem mantendo a essência do curso. Pensamos que esta atitude é errada, pois para além de tolerar um certo desleixo conceptual e notacional, passa ao lado de uma série de questões fundamentais que não têm existência real fora das linguagens: como é o mecanismo de passagem de parâmetros? que acontece quando o índice de um vector está fora dos limites? como se detecta o fim de um ficheiro? como se avalia uma expressão lógica? Embora estas questões possam ser descritas em abstracto, elas só fazem sentido se estivermos a pensar nalguma linguagem. Se não dermos importância a isto logo na origem, os estudantes subvalorizarão pontos fundamentais da programação, e adoptarão indesejáveis hábitos de falta de rigor.

3 Programação com Pascal

O Pascal tem sido a linguagem favorita para ensinar Programação, se não desde que foi inventada, pelo menos desde que se tornou largamente disponível nos pequenos computadores com o sistema USCD Pascal. É verdade que recentemente certas experiências têm sido feitas no sentido

de substituir o Pascal por outras linguagens mais modernas, por exemplo, o Modula 2, o Smalltalk, o ML. A primeira tem a vantagem de permitir a programação com tipos abstractos, a segunda a orientação pelos objectos, a terceira uma maior clareza na abstracção funcional. Não é claro que os ganhos tenham sido enormes.

Actualmente, muitos dos alunos que chegam à universidade em Portugal para cursos de informática ou outros de engenharia já conhecem o Pascal. Aparentemente, é esta a linguagem mais usada no ensino secundário, nas escolas que têm Informática. Na realidade, verifica-se que, na quase totalidade dos casos, o conhecimento é bastante superficial: não só os estudantes ignoram aspectos importantes da linguagem, como a utilização que dela fazem é rudimentar: por exemplo, é frequente encontrar longas sequências de if-then-elses, quase sempre as variáveis aparecem todas declaradas globalmente, os procedimentos e funções são ad-hoc, há ciclos imbricados com uma profundidade excessiva, etc. Além disso, desconhecem quase completamente os principais algoritmos e estruturas de dados. E por fim, não só geralmente não têm nenhum método, como adoptam alegremente a perigosa filosofia de que "o que é preciso é que funcione".

Seja como for, os estudantes novatos pensam que, de Pascal, e por conseguinte (nas suas cabeças…) de Programação, já sabem quase tudo. Por isso, muitos encaram com uma certa má-vontade, ou mesmo com sobranceria, o curso inicial de Programação com Pascal.

Ainda por cima, nos cursos de Informática é frequente que a disciplina inicial de Programação surja logo no primeiro ano, ocupando um semestre. O nosso curso na licenciatura de engenharia informática na Universidade Nova de Lisboa era uma excepção, pois a Programação aparecia como disciplina anual do segundo ano. As razões para esta escolha são antigas e nunca foram desmentidas: no primeiro ano, os alunos recém-chegados à universidade têm sobretudo disciplinas de carácter propedêutico (Matemática, Física, História de Ciência, etc.) do mesmo género das que conhecem do ensino secundário, e isso ajuda a suavizar a transição. As disciplinas da especialidade (Sistemas de Operação, Bases de Dados, Análise de Sistemas, etc.) só surgem a partir do terceiro ano. Assim ficava o segundo ano todo para fazer a aprendizagem universitária da Programação. É verdade que muitos alunos reclamavam, porque estavam ansiosos por aprender as matérias específicas do curso que escolheram, mas descontando essas reclamações, estamos convencidos que o rendimento do curso no segundo ano era muito melhor do que poderia ser caso ocorresse no primeiro. Se bem que não haja medidas objectivas, esta convicção reforça-se de cada vez que directamente ou indirectamente participamos no ensino da Programação a alunos que são do primeiro ano da universidade. Apesar disto tudo, recentemente a Universidade Nova de Lisboa decidiu antecipar a disciplina de Programação para o primeiro ano, a partir do ano lectivo 1996-1997. Veremos quais são os resultados.

Neste curso de Programação, adoptamos desde logo a atitude de que programar é resolver problemas usando uma linguagem de programação. Quer dizer, a resolução de problemas e a utilização esclarecida, rigorosa e competente de uma linguagem de programação são as duas faces das mesma moeda. Para mais, tratando-se de estudantes de engenharia informática, todo o ensino é desenvolvido no contexto da engenharia de software. Isto é, os alunos são desde logo avisados que não basta que o programa funcione: também é preciso que seja compreensível, eficiente, modificável e fiável [Ross, Goodenough, Irvine 1975].

A linguagem escolhida tem sido o Pascal. As razões desta escolha não são muito profundas: o Pascal existe, existe a experiência de ensinar a Programação com o Pascal, o Pascal é realmente uma boa linguagem para ensinar a programação, que nos permite concentrar-nos em cada aspecto da matéria, sem distracções, e por uma ordem que é eficaz pedagogicamente.

O nosso curso tem os seguintes capítulos:

1. Resolução de problemas de programação.
2. Programação com funções e procedimentos.
3. Processamento de vectores e matrizes.
4. Processamento de ficheiros sequenciais.

5. Técnicas de processamento de texto.
6. Estruturas de dados recursivas.
7. Elementos de cálculo numérico.

O curso é guiado, portanto, pelo tipo de problemas a resolver. O facto de não aparecer nenhum capítulo sobre a linguagem não significa que os aspectos linguísticos sejam subvalorizados. Pelo contrário: a ideia é que não vale a pena, por exemplo, explicar em abstracto o que é um procedimento ou o que é a instrução for. É preferível apreciar a necessidade desses conceitos relacionados com problemas. Por exemplo, os primeiros procedimentos que surgem são procedimentos sem parâmetros, resultantes da decomposição do problema em subproblemas:

```
procedure ReadData;
begin
  ...
end;

procedure ComputeResults;
begin
  ...
end;

procedure DisplayResults;
begin
  ...
end;
```

Não vale a pena, portanto, insistir na passagem de parâmetros quando o que se quer é apenas encapsular um conjunto de instruções. Os parâmetros surgem depois, naturalmente, no contexto de funções, que inicialmente têm um sabor matemático:

```
function Min(x: Integer; y: Integer): Integer;
begin
  ...
end;
```

Os parâmetros no procedimentos servem normalmente, não para parametrizar listas de instruções, mas no contexto do processamento de estruturas de dados, inicialmente vectores:

```
procedure Sort(var theArray: IntegerArray; numberOfElements: Integer);
begin
  ...
end;
```

Quer dizer, como o ensino é guiado pelos problemas, a linguagem aparece conforme as necessidades. Se, inversamente, nos baseássemos na linguagem, então teríamos um capítulo sobre procedimentos, e nele percorreríamos todas as variantes (sem parâmetros, com parâmetros, com variáveis globais, sem variáveis globais, com parâmetros e com variáveis globais, etc.) e ficaria talvez mais difícil ver claramente que algumas destas variantes são indesejáveis, ainda que a linguagem as coloque todas em igualdade.

Um outro exemplo é o do instrução for. Numa abordagem guiada pela linguagem haveria um capítulo sobre instruções iterativas, e nesse capítulo, provavelmente depois de estudar a instrução while e a instrução repeat, dir-se-ia que a instrução for se usa quando se sabe à partida quantas vezes se vai repetir o passo do ciclo. Nem sempre é assim, claro, mas a simplificação até é razoável, num primeiro contacto. Depois, exemplificar-se-ia a instrução com um troço de programa para escrever 100 vezes uma frase qualquer:

```
for i:=1 to 100 do
  Writeln("O Pascal é que é bom.");
```

É duvidoso que isto deixe perceber como é que se pode usar a instrução for para resolver problemas. Inversamente, numa abordagem guiada pelos problemas, a instrução for aparece liga-

da ao processamento de vectores, quando o problema exige que se percorra todo o vector: um bom exemplo é o problema de achar o máximo de um vector, para o qual não faltam situações reais, que podem servir de pretexto.

O inconveniente desta abordagem guiada pelos problemas é que à medida que o nosso próprio estilo de programação evolui, à medida que a influência da programação de tipos abstractos de dados e da programação orientada pelos objectos se reforça, o Pascal vai reagindo cada vez com mais dificuldades. Comparativamente, se nos baseássemos na linguagem, essas interferências não seriam sentidas: um procedimento é sempre um procedimento, uma instrução for é sempre uma instrução for.

Concretamente, no capítulo sobre processamento de texto queremos ter uma tabela de cadeias de caracteres, com operações para inserir uma palavra na tabela, outra para ver se a palavra existe, etc. Ao mesmo tempo, queremos separar claramente a interface da implementação, disciplinar o acesso a recursos que consideramos privativos, e deixar as coisas suficientemente abstractas para poder mudar a implementação sem grandes sobressaltos. Nessa altura constatamos que o Pascal não nos dá margem de manobra para nos exprimirmos como desejamos.

O mesmo acontece quando se estudam as estruturas de dados recursivas: listas e árvores. Se bem que a programação seja perfeitamente fazível, a organização que o Pascal proporciona deixa muito a desejar e vai ao invés dos objectivos de compreensibiliade, modificabilidade e fiabilidade que queremos que os estudantes persigam nos seus programas. Quer dizer, a partir de certa altura, o Pascal deixa de reagir convenientemente às nossas próprias necessidades de programação. De certo modo damos por nós numa posição ridiculamente semelhante à dos que há 20 anos praticavam programação estruturada em FORTRAN, simulando ciclos while à base de rendilhados de instruções IF e GOTO.

Que fazer então? Já que as nossas dificuldades têm a ver com a transição para a programação de tipos abstractos e para a programação orientada pelos objectos, devemos nós finalmente abandonar o Pascal clássico e adoptar uma linguagem decididamente orientada pelos objectos?

4 A Programação Orientada pelos Objectos

Há vinte anos, o facto de não se saber bem o que era a programação estruturada não impediu que ela se tornasse o principal paradigma de programação, aceite e adoptado em todo o lado. Agora parece que o que todos querem é programação orientada pelos objectos, mesmo se também não é claro o que isso seja, nem tão-pouco onde nos vai levar.

Também não será num curso introdutório de Programação que essa discussão deve ser feita: provavelmente os estudantes não compreenderiam o alcance das nossas angústias de professores numa encruzilhada. O que há a fazer, se continuarmos a adoptar uma abordagem guiada pelos problemas, é adoptar um estilo orientado pelos objectos, com a mesma naturalidade que antes fazíamos os refinamentos sucessivos. Mas então, que linguagem devemos usar? O Pascal ISO só dá até certo ponto, como vimos. Adoptar uma versão não normalizada de Object Pascal perturbaria a portabilidade (frequentemente, mesmo na aprendizagem, é preciso mudar de plataforma, e é fundamental que isso se faça sem atrapalhações).

A necessidade de uma linguagem disponível na maior parte das plataformas, e que realmente inspire confiança, reduz muito a nossa escolha. À partida, as linguagens possíveis são o Smalltalk, o Eiffel, o C++ e o Oberon. Mais recentemente, juntaram-se a este grupo o Java e o Ada95.

Não será, portanto, por falta de linguagem que não tornamos o nosso ensino orientado pelos objectos. No entanto, a escolha é difícil pois nenhuma das linguagens se distingue como sendo particularmente apropriada para o ensino, como era o caso do Pascal em relação à programação estruturada.

O Smalltalk é uma óptima linguagem (as outras também…) mas um tanto estranha, para quem tem uma cultura de programação formada nas linguagens da família Algol. Se bem que existam sistemas Smalltalk em muitas plataformas, não é claro para um não iniciado se eles são

todos compatíveis entre si. Além disso, a programação com Smalltalk parece tão diferente da programação habitual em C ou COBOL, que não fica claro que se ganhe muito com aprender Smalltalk, para além de eventualmente alguns bons hábitos.

O Eiffel ambiciona ser o Pascal dos objectos, isto é, a linguagem de eleição para o ensino da programação orientada pelos objectos. É uma linguagem que tem preocupações de Engenharia de Software, o que é uma vantagem. Mas é relativamente pouco acessível, e ainda não se livrou de uma certa fama de pouca confiabilidade.

O C++ gaba-se de ser linguagem mais usada na programação orientada pelos objectos, mas quase todos concordam que é complicada demais para ensinar. Tem a vantagem de ser popular, de haver muitos livros de texto para C++, e muitas boas ferramentas para a programação C++, disponíveis em todas as plataformas. O esforço para aprender C++ não se perde, pois é uma linguagem muito requisitada nas empresas.

O Oberon mantém-se ainda no sector das curiosidades. É uma linguagem simples, eficaz e directa, com um pequeno manual de referência, disponível gratuitamente para muitas plataformas, mas que não conseguiu ainda o estatuto de uma linguagem séria.

O Java é uma invenção ainda muito recente. Apesar de todo o alarido que tem havido à sua volta, a linguagem ainda não está suficientemente difundida, e as suas ferramentas inspiram menos confiança que as do C++, por exemplo. É certamente cedo demais para a adoptar no ensino da programação.

O Ada95 traz as boas credenciais do Ada original (Ada83), e junta-lhe a orientação pelos objectos. O Ada original era uma linguagem muito extensa e difícil, ainda que bem construída. Ensinar Ada83 já era complicado, ensinar Programação com Ada95 não deve ser mais simples. Tem a vantagem de, tal como o Oberon, dispor de implementações gratuitas para várias plataformas.

O panorama é animador: há muito por onde escolher e nenhum dos candidatos apresenta vantagens decisivas. Ainda bem! Podemos assim escapar à monotonia e à estagnação que surge quando todos estão de acordo.

5 O novo curso de Programação

Ensinar várias linguagens num curto espaço de tempo costuma dar mau resultado. Quando finalmente os estudantes começam a apreciar uma linguagem, já têm que a pôr de lado para começar com a próxima. Se isto se passa no contexto da aprendizagem da Programação, a confusão pode ser grande. Por isso, ao ensinar Programação, o ideal seria ter uma linguagem única que nos acompanhasse de uma ponta à outra. (Claro que os estudantes deverão depois estudar mais linguagens noutros cursos, mas isso é outro assunto.)

Infelizmente, ao querer praticar a programação orientada pelos objectos no nosso curso de Programação, não encontramos uma linguagem que nos satisfaça e que nos tranquilize. O que nós gostaríamos de ter era uma linguagem que nos permitisse ensinar a resolução de problemas de programação, começando pelas coisas simples, e continuando gradualmente até à programação orientada pelos objectos, sempre com as preocupações da engenharia de software em mente. Ora, em nossa opinião, nenhuma das linguagens enumeradas há pouco dá para cobrir todo este espectro.

Mas se uma não cobre, talvez duas em conjunto cubram. É verdade que isto vai contra a ideia de uma linguagem única, mas pelo menos temos a consolação que, tratando-se de um curso anual, há tempo para fazer a transição.

O solução que está a ser experimentada neste momento consiste em dividir o curso em duas partes semestrais. Na primeira, em que temos que começar do zero, mantemos a abordagem tradicional, usando Pascal, refinamentos sucessivos, etc. Isto dá-nos oportunidade de explicar aqueles mecanismos próprios das linguagens sem grandes distracções: variáveis, tipos, funções, procedimentos, passagem de parâmetros, vectores, organização da memória. No entanto, não leva-

mos a resolução de problemas muito longe, para não entrar na zona em que o Pascal começa a dar-se mal. Assim, dos seis capítulos indicados atrás, só mantemos os quatro primeiros: os três últimos — técnicas de processamento de texto, estruturas de dados recursivas, cálculo numérico — ficam para a segunda parte, ainda que num contexto diferente.

A linguagem que seleccionámos para a segunda parte foi o C++. É uma escolha surpreendente, face aos argumentos que temos vindo a desenvolver: o C++ é de facto uma linguagem muito complicada, nada própria, à partida, para ensinar Programação. Que razões militam a favor do C++, então?

6 C++ educativo

O C++ é uma linguagem popular. A popularidade só por si não é qualidade claro, mas é verdade que há cada vez mais gente a querer saber programar em C++. E também é verdade que geralmente os programadores gostam do C++. Não se percebe bem por que é que gostam, mas gostam. Se calhar gostam pelas razões erradas. Se calhar gostam porque a linguagem é complicada e dá luta, se calhar gostam porque os programas ficam difíceis de modificar (e isso é uma garantia adicional de emprego), se calhar gostam porque a linguagem nos lembra constantemente que estamos a trabalhar com computadores de plástico e silício e não com mágicas máquinas abstractas de estruturas e conceitos, se calhar gostam porque os espíritos bem-pensantes acham que o C++ é uma vergonha e devia ser proibido.

Por outro lado, mesmo se é complicado e difícil, o C++ inspira uma certa confiança. É uma linguagem já bastante estável, provém do C, que é uma linguagem de referência, existe em quase todas as plataformas, muitos fabricantes de software fornecem compiladores de C++ e ferramentas de programação associadas. O esforço necessário para aprender a programar em C++ não se vai perder, com certeza. Pelo contrário: acrescentar ao currículo uma linha com "programa em C++" é algo que cada vez mais programadores ambicionam.

Tentar ensinar Programação com uma linguagem difícil demais pode parecer uma perda de tempo. Talvez fosse melhor esperar que surja, se não tiver surgido já, uma linguagem mais razoável. Ou então não, ficamos com o C++, tal como ele é, e esforçamo-nos por fazer do vício virtude, tirando partido construtivamente mesmo das suas dificuldades.

Sendo o C++ difícil e complicado, para o dominar é preciso disciplina. Se o enfrentarmos de qualquer maneira, rapidamente ficaremos soterrados por uma avalancha de conceitos que se podem combinar de imensas maneiras, que por vezes entram em conflito uns com os outros, que têm justificações duvidosas, que estão lá para assegurar estranhas compatibilidades. Com tantas maneiras de exprimir ideias parecidas, logo nos esquecemos por que optámos por uma e não por outra. Com tanto por onde escolher, hesitamos tempo demais perante cada escolha, e frequentemente ficamos com a impressão de que se calhar afinal não escolhemos bem. A linguagem, em vez de nos servir para ajudar a resolver o problema, torna-se um obstáculo suplementar.

É claro que a linguagem é neutra em relação à nossa insegurança como programadores. Tudo o que ela permite é válido, em algum sentido, e até pode ser útil. Não obstante, por vezes conceitos importantes em programação aparecem na linguagem lado a lado com outros que são meramente acessórios. Como é que o principiante pode distinguir o trigo do joio? Bom, cingindo-se a uma disciplina de programação, que se baseie nos mecanismos sãos da linguagem, para com eles exprimir modelos claros do universo do problema, deixando de parte, ou para mais tarde, as construções mais fantasiosas.

A disciplina é uma das qualidades fundamentais de um programador, seja em que linguagem for. Um programador indisciplinado, mesmo que genial, é de pouca utilidade para uma equipa de desenvolvimento de software. Ora o C++ é uma linguagem boa para cultivar a disciplina na programação. Com linguagens mais simples, talvez seja possível levar a bom termo alguns pequenos projectos sem tomar consciência da necessidade de disciplina. Com C++, isso é praticamente impossível. Com C++, os estudantes percebem rapidamente que, ou se tornam disciplinados, ou não vão a lado nenhum.

Ao programar em C++, frequentemente seremos confrontados com as nossas próprias limitações como programadores. Mesmo adoptando espontaneamente uma disciplina, muitas vezes cometeremos erros, não propriamente por termos compreendido mal o problema, mas porque pensávamos que já conhecíamos bem a linguagem. Isto é uma situação geral em Programação, mas fica mais gritante quando a linguagem é complexa. É verdade que os erros mais graves são mesmo os que vêm de não termos compreendido o problema. O excesso de confiança é mau conselheiro em programação. Por outro lado, errar, reconhecer o erro, corrigi-lo, faz bem à alma.

A linguagem C++, por ser complexa, ensina os programadores a ser humildes, a respeitar as suas limitações, a reconhecer as dos outros, a nunca partir do princípio que programar é fácil. A humildade, como Dijkstra observou há já muito tempo, é outra das qualidades fundamentais de um programador [Dijkstra 72B]. Ao programar com C++ estaremos permanentemente a recordar-nos que afinal sabemos ainda muito pouco.

Deixar um problema de programação bem resolvido é uma tarefa estafante. É preciso ultrapassar tantos erros, uns aparatosos outros muito discretos, é preciso rever e tornar a rever decisões que pensávamos serem definitivas, é preciso voltar a estudar pontos da linguagem que julgávamos já dominar completamente, é preciso atender a inúmeros pormenores, é preciso afinar o código interminavelmente, é preciso deixar o programa em condições de ser continuado por outras pessoas. E frequentemente isto tudo tem que ser feito sob a pressão de prazos a cumprir. É normal que surjam momentos de desânimo, em que apetece deixar as coisas como estão, e pronto. Mas não pode ser. Temos que perseverar, temos mesmo fazer as coisas como deve ser. Com efeito, a perseverança é talvez a terceira qualidade fundamental de um programador.

Ora o C++ é bom para educar a perseverança. A linguagem é difícil, só chegaremos ao fim se formos corajosos, perseverantes. Caso contrário ficaremos com uma luzes, mas não programaremos verdadeiramente em C++.

Em resumo, o C++ é complicado? É sim. Em vez de nos lamentarmos disso, vamos usá-lo disciplinadamente, humildemente, perseverantemente. No final, os estudantes serão melhores programadores, não só porque conhecem uma nova linguagem mas também porque serão mais disciplinados, mais humildes, mais perseverantes. Assim seja!

7 Utilização do C++

Uma utilização desregrada do C++, tal como uma utilização desregrada de qualquer outra linguagem, nunca conduz a bons resultados. Mesmo na programação tradicional em Pascal há regras que se sobrepõem às da linguagem, a mais conhecida das quais é: “não usar gotos”. Mas há outras, ou devia haver:

- Não usar efeitos laterais em funções.
- Não declarar globalmente o que pode ser declarado localmente.
- Evitar if-then-else muito imbricados.
- Não usar if-then (sem else) dentro de if-then-else.
- Etc.

Com o C++, há muito mais regras como estas. A maior parte tem a ver com eliminar da programação todos aqueles aspectos da linguagem que não têm a ver com a resolução de problemas mas sim com a optimização do código ou com aspectos de conveniência pontual: funções em-linha, membros protegidos, funções amigas, argumentos por defeito, etc. Eis algumas dessas regras:

1. Cada classe tem uma parte pública e uma parte privada (não se usa a parte protegida).
2. O ficheiro .h contém a declaração da classe e o ficheiro .cpp a definição de todas as funções.

3. Todas as classes têm um ou vários construtores explícitos e um destrutor explícito, mesmo se vazio (excepto classes abstractas).
4. Cada função normais (isto é, todas excepto os construtores e o destrutor) são ou selectores (retornam um aspecto do estado do objecto, sem o modificar) ou modificadores (mudam o estado do objecto, sem retornar nada).
5. A inicialização dos membros nos construtores faz-se sempre que possível na lista de inicialização dos membros e não no corpo do construtor.
6. Todas as funções são virtuais.
7. Só se definem operadores com significados consagrados: afectação (=), indexação ([]), e comparação (==, !=, <=, < , >=, >).
8. Não se usa funções em-linha (nem se fala nisso).
9. Não se usa funções amigas (nem se fala nisso).
10. Não se usa funções com argumentos por defeito (nem se fala nisso).
11. Só se usa a variante posfixa dos operadores ++ e --.
12. Os argumentos de tipos simples (int, long, apontador) são passados por valor, e aqueles cujo tipo é uma classe são passados por referência const.
13. Os resultados de uma função ou são de tipos simples ou são passados por referência const.
14. Não se usa resquícios desnecessários do C, como struct, union (nem se fala nisso).
15. Instancia-se classes genéricas através de typedef.
16. A herança é sempre pública.
17. Todas as estruturas de dados são implementadas através de classes.
18. As classes de biblioteca são enriquecidas por derivação.
19. Não se usa abreviaturas nos nomes das classes nem nos nomes das funções públicas.
20. Usa-se sistematicamente o mesmo nome (em Inglês) para funções análogas em classes diferentes.

Nem todas estas regras têm que ser enunciadas explicitamente. Algumas correspondem apenas a partes que são omitidas.

Esta abordagem, se pode ser pedagogicamente justificada, depara-se com alguns obstáculos. O primeiro é que o compilador não garante nada disto. Por exemplo, se um estudante, por lapso ou por rebeldia, decidir omitir a palavra reservada virtual na declaração de uma função, o compilador não se queixa. Mais tarde, o funcionamento polimórfico pode não corresponder ao que terá sido descrito. De qualquer forma, nunca podemos confiar no compilador para garantir que a disciplina preconizada é respeitada.

A dificuldade seguinte é que os livros sobre C++ normalmente não seguem esta abordagem. O de Stroustrup, por exemplo, pretende ser um livro de referência (e contém o manual de referência do C++), e não tem grandes preocupações metodológicas ou pedagógicas [Stroustrup 91]. O de Lippman, sendo um livro de iniciação ao C++, segue a abordagem de passar em revista cada um dos conceitos exaustivamente, e considera todas as variantes, mesmo aquelas que não nos interessam [Lippman 91] .

A terceira dificuldade é de outra natureza: se estamos a praticar um C++ um tanto depurado, os estudantes, quando caírem na realidade e confrontarem o que aprenderam com programas C++ de outras origens, podem sentir-se desprotegidos ou mesmo enganados, ao verificar que os programas a sério comportam muitas coisas que no curso foram desvalorizadas.

Finalmente, é normal que os estudantes pensem que ao aprender C++ vão aprender também C, por feito lateral. Ora as técnicas de programação que aquelas regras induzem não têm nada a ver com as que são próprias do C. Basta observar que em C++ usam-se muito menos apontadores, e por isso toda aquela aritmética de apontadores, tão típica no C, é completamente omitida. Uma outra abordagem à programação com C++, que começasse pela aprendizagem do C permitiria, por hipótese, ganhar as duas linguagens pelo preço de uma. Talvez isso possa ser verdade, do ponto de vista das linguagens em si, mas dadas as diferenças de objectivos entre as duas, du-

vido que a componente de programação orientada pelos objectos em C++ pudesse acabar por receber a atenção necessária.

Conclusão

Mesmo quando estamos contentes com o que temos, com os métodos que usamos, é nossa obrigação de professores universitários, tentar melhorar, perceber o que está para vir, experimentar novas abordagens aos velhos problemas, enriquecer os nossos cursos com ideias modernas e interessantes. Neste momento de viragem nas técnicas de programação, com a afirmação cada vez mais evidente da programação orientada pelos objectos, temos pela frente a necessidade de decidir como fazer.

Se bem que nestes assuntos de programação, frequentemente os campos estejam muito marcados, compete-nos ter uma atitude aberta, exploratória, e simultaneamente não aceitar dogmaticamente aquilo que dizem os que fazem mais barulho. Aliás, penso que devemos mesmo transmitir também isto aos alunos, no sentido de os ajudar a desenvolver o seu espírito crítico, e de serem capazes, mais cedo ou mais tarde, de escolher as linguagens e as ferramentas que melhor resposta podem dar às suas necessidades.

A questão de como ensinar a programação é uma velha questão, que não tem uma resposta definitiva. Se é verdade que não há vantagem em embarcar levianamente em modas passageiras, também não devemos entrincheirar-nos naquilo que conhecemos bem, só porque é mais seguro e já deu provas.

O C++ não é certamente a linguagem ideal, e muito menos a linguagem ideal para ensinar a Programação. De certo modo, os defeitos do C++ não estão no que lhe falta, mas na abundância do que oferece. Aquilo que procuramos numa linguagem moderna, encontramos no C++: classes, herança, polimorfismo, classes genéricas, separação entre interface e implementação, modularização, portabilidade, bibliotecas. Infelizmente, dirão alguns, isso vem no meio de um grande número de conceitos estranhos que tornam muito difícil escrever programas fiáveis, compreendê-los, modificá-los.

Mas programar bem é difícil, e há-de ser sempre difícil. É verdade que dispensaríamos de bom grado as complicações adicionais de uma linguagem com coisas demais. Mas se formos disciplinados, agora como antes, veremos que essas complicações podem em grande parte ser evitadas.

Pois bem, a nossa missão, hoje com a programação orientada pelos objectos como antes com a programação estruturada, é descobrir o melhor caminho para ajudar os nossos estudantes a tornarem-se bons programadores. E esse caminho faz-se com as linguagens existentes, e não na miragem de uma perfeição inatingível.

Referências

[Dahl, Dijkstra, Hoare 72] O-J Dahl, E. W. Dijkstra, C. A. R. Hoare, *Structured Programming*, Academic Press, London, 1972.

[Dijkstra 72A] E. W. Dijkstra, *Notes on Srucuried Programming* in [Dahl, Dijkstra, Hoare 72]

[Dijkstra 72B] E. W. Dijkstra, The Humble Programmer ,*Comm. of the ACM*, Vol 15, 10, pp. 859-866, Outubro 1972.

[Jensen, Wirth 88] K. Jensen e N. Wirth, *Pascal User Manual and Report*, Springer, 1988.

[Lippman 91] S. B. Lippman, *C++ Primer*, Addison-Wesley, Reading Massachusetts, 1991.

[Parnas 72] D.L. Parnas. On the Criteria to be Used in Decomposing Systems into Modules. Comm. of the ACM, vol 15, 12, pp. 1053-1058, Dezembro, 1972.

[Ross, Goodenough, Irvine 1975], D. Ross, J. Goodenough e C. Irvine, Software Engineering: Processes, Principles, and Goals, *IEEE Computer*, Maio 1975.

[Stroustrup 91] B. Stroustrup, *The C++ Programming Language*, Addison-Wesley, Reading Massachusetts, 1991.

[Wirth 71] N Wirth, Program Development by Software Refinement, *Comm. of the ACM*, Vol 14, 4, pp. 221-227, Abril 1971.