

Apêndice A: Como compilar um programa C

J. L. Rangel

Introdução.

Este apêndice tem como finalidade mostrar como pode ser feita a compilação de um programa C. Tipicamente, um programa C é dividido em vários arquivos, que são usados para formar um único arquivo executável, e a forma como estes arquivos devem ser compilados e ligados pode parecer inicialmente confusa.

No caso mais simples, temos uma situação semelhante à dos exemplos dos primeiros capítulos deste curso, em que todo o código escrito pelo programador pode ser salvo em apenas um arquivo de código fonte. Como vimos, mesmo nestes exemplos pequenos, o programa faz uso de uma biblioteca da linguagem C, a biblioteca padrão de entrada-saída.

Entretanto, programas maiores precisam ser divididos em vários arquivos, mas esta divisão não pode ser feita de uma maneira qualquer. Um programa grande é dividido em “módulos”, organizados de maneira bem definida, onde cada módulo tem uma finalidade específica. A forma de decomposição mais comum é a divisão em um programa principal, e várias bibliotecas, cada uma das quais reúne um certo número de funções com finalidades semelhantes.

Vamos mostrar aqui, através de um exemplo pequeno, algumas maneiras de compilar um programa C dividido em módulos. O programa exemplo faz uso de três bibliotecas, a biblioteca de entrada/saída padrão, `stdio`, e duas bibliotecas definidas pelo usuário, `show` e `bibl`.

A.1. O programa exemplo

O programa está dividido em cinco arquivos:

- `bibl.h` e `bibl.c`, que formam a biblioteca “`bibl`”, com as funções `soma` e `prod`
- `show.h` e `show.c`, que formam a biblioteca “`show`”, com as funções `showStr` e `showInt`
- `main.c`, onde pode ser encontrada a função `main`

A divisão das bibliotecas em dois arquivos é feita de uma maneira que usaremos freqüentemente: o arquivo `.h` contém os cabeçalhos (*headers*) das funções, cuja declaração é completada no arquivo `.c` correspondente.

Quando uma chamada de função em C é compilada, o compilador procura verificar que os tipos dos argumentos da função estão corretos, e que o resultado da função está sendo usado corretamente. Para esta verificação, o compilador usa a informação contida no cabeçalho da função, onde estão especificados o número dos parâmetros, seus tipos e o tipo do resultado da função. Esta informação é exatamente o conteúdo dos arquivos `.h`, que são “incluídos” em outros arquivos, para permitir a verificação das chamadas de cada função.

Cada arquivo tem na primeira linha um comentário para identificação. O conteúdo dos cinco arquivos é o seguinte:

```
/* EDA: bibl.h */

/* soma dois inteiros */
int soma(int x, int y);

/* multiplica dois inteiros */
int prod(int x, int y);
```

```
/* EDA: bibl.c */

#include "bibl.h"

int soma(int x, int y) {
    return x+y;
}

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

```
/* EDA: show.h */

/* exibe uma cadeia de símbolos */
void showStr(char *s);

/* exibe um numero inteiro */
void showInt(int x);
```

```
/* EDA: show.c */

#include <stdio.h>
#include "show.h"

void showStr(char *s) {
    printf("%s",s);
}

void showInt(int x) {
    printf("%d",x);
}
```

```

/* EDA: main.c */

#include "show.h"
#include "bibl.h"

int main(void) {
    int a=2;
    int b=3;
    int c;
    showStr("O primeiro inteiro e' ");
    showInt(a);
    showStr("\n");
    showStr("O segundo inteiro e' ");
    showInt(b);
    showStr("\n");
    c=soma(a,b);
    showStr("A soma dos dois inteiros e' ");
    showInt(c);
    showStr("\n");
    c=prod(a,b);
    showStr("O produto dos dois inteiros e' ");
    showInt(c);
    showStr("\n");
    return 0;
}

```

A.2. Compilando e ligando o programa à mão

Vamos mostrar os comandos necessários para a compilação, considerando que o compilador a ser utilizado é o *GNU C compiler*, em sua versão para DOS, `gcc.exe`. Estes comandos podem ser adaptados para a versão Linux do compilador, ou para outros compiladores. Informações adicionais podem ser encontradas na documentação correspondente. RTFM¹.

Veja na última seção deste apêndice como alguns compiladores de C (adequados para a disciplina Estruturas de Dados) podem ser obtidos a custo baixo ou nulo.

Ordem de compilação e de ligação. As operações de compilação e de ligação dos arquivos para construir um arquivo executável devem ser realizadas numa ordem que respeite a “dependência” entre os diversos arquivos. Isto quer dizer, simplesmente, que as entradas de uma operação devem estar prontas antes da operação ser realizada. Dizemos que as saídas “dependem” das entradas. Considerando nosso exemplo, os arquivos que nos interessam são

- os cinco arquivos fonte `bibl.h`, `bibl.c`, `show.h`, `show.c`, `main.c`, criados usando algum editor apropriado.
- os arquivos objeto resultantes da compilação de `bibl.c`, `show.c` e `main.c`: `bibl.o`, `show.o` e `main.o`.

¹ RTFM = “read the fine manual”, ou seja, “leia o maravilhoso manual”. Esta sigla é muito usada na resposta a pedidos de socorro por *mail*.

- o arquivo executável resultante da ligação dos arquivos objeto `bibl.o`, `show.o` e `main.o`: `prog.exe`

Para edição, no caso do DOS, poderia ser usado o `edit` ou qualquer editor que não acrescente formatação ao texto.

Não há necessidade de compilar separadamente os arquivos `bibl.h` e `show.h`, porque estes arquivos são “incluídos” em outros arquivos durante a compilação. Isto quer dizer, por exemplo, que compilar `bibl.c`, que inclui `bibl.h` significa compilar

```
/* EDA: bibl.c */

/* EDA: bibl.h */

/* soma dois inteiros */
int soma(int x, int y);

/* multiplica dois inteiros */
int prod(int x, int y);

int soma(int x, int y) {
    return x+y;
}

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

com a linha que “inclui” `bibl.h` substituída pelas várias linhas do arquivo incluído. (Linhas em branco e comentários são ignorados pelo compilador.)

Os arquivos objeto são obtidos pela compilação dos arquivos fonte correspondentes, pelos comandos

```
gcc -c bibl.c
gcc -c show.c
gcc -c main.c
```

Estes comandos geram os arquivos `bibl.o`, `show.o` e `main.o` correspondentes. Os arquivos objeto, por sua vez, são ligados para formar o executável `prog.exe` por

```
gcc -o prog main.o bibl.o show.o
```

O programa resultante da compilação e ligação, `prog.exe`, pode ser chamado por `prog`

tendo como resposta

```
0 primeiro inteiro e' 2
0 segundo inteiro e' 3
A soma dos dois e' 5
0 produto dos dois inteiros e' 6
```

Se desejado, para evitar escrever todos os comandos, o processo de compilação/ligação pode ser facilitado. Podemos construir um arquivo “*batch*” (*lote*), ou seja, um arquivo de texto com vários comandos do sistema operacional, um em cada linha. No nosso caso, o sistema operacional é o DOS, e os arquivos *batch* têm nomes terminados em `.bat`, digamos, `compila.bat`.

```
rem EDA: compila.bat: compila e liga prog
gcc -c bibl.c
gcc -c show.c
gcc -c main.c
gcc -o prog main.o bibl.o show.o
```

A primeira linha do arquivo é um comentário (*remark*). Assim, o comando `compila.bat`

ou, simplesmente
`compila`

seria suficiente para executar todos os comandos de compilação e ligação, gerando o executável `prog.exe`. Esta solução automatiza o processo, mas tem uma desvantagem: se por alguma razão um dos arquivos fonte for alterado, torna-se necessário repetir todo o processo.

Suponha, por exemplo, que um erro foi descoberto, e, para corrigi-lo, foi necessário apenas alterar `bibl.c`. As ações que precisam ser tomadas para gerar um novo `prog.exe` são

```
gcc -c bibl.c
gcc -o prog main.o bibl.o show.o
```

sendo as outras duas desnecessárias. Se a correção fosse feita em `bibl.h`, como este arquivo é incluído em `bibl.c` e `main.c`, três ações seriam necessárias

```
gcc -c bibl.c
gcc -c main.c
gcc -o prog main.o bibl.o show.o
```

Naturalmente, a preocupação de evitar a execução de ações desnecessárias só faz sentido porque os programas encontrados na prática costumam ser muito maiores que o nosso pequeno exemplo.

Para evitar as re-compilações desnecessárias, foi introduzida a ferramenta `make`, que será discutida na próxima seção.

A.3. Compilando e ligando o programa usando a ferramenta `make`

Para automatizar o processo de tratar apenas os arquivos que forem necessários é preciso identificar os arquivos “alterados”, e gerar novamente apenas os arquivos que dependem de arquivos alterados. Para decidir que arquivos devem ser tratados, podemos examinar a informação sobre data e hora da última alteração dos diversos arquivos, que é anotada pelo sistema operacional. Por exemplo, se `bibl.c` foi alterado depois da sua última compilação, o arquivo terá uma data/hora mais recente que `bibl.o`. e, uma vez que “`bibl.o` *depende de* `bibl.c`”, a re-compilação é necessária.

Esta informação é passada para o programa `make`, que “faz” um arquivo executável, com o mínimo de operações necessárias. A informação para `make` fica num arquivo, que normalmente recebe o nome `makefile`. (RTFM!)

Para o nosso exemplo, devemos passar para o programa `make` as seguintes informações:

1. Para gerar o arquivo executável prog, precisamos de (versões atualizadas de) main.o, bibl.o e show.o. Se prog não existir, ou se estiver desatualizado, gere um usando o comando fornecido.

```
prog: main.o bibl.o show.o
gcc -o prog main.o bibl.o show.o
```

2. Para gerar o arquivo objeto bibl.o, precisamos de (versões atualizadas de) bibl.c e de bibl.h. Se bibl.o não existir, ou se estiver desatualizado, gere um usando o comando fornecido.

```
bibl.o: bibl.c bibl.h
gcc -c bibl.c
```

3. Para gerar o arquivo objeto show.o, precisamos de (versões atualizadas de) show.c e de show.h. Se show.o não existir, ou se estiver desatualizado, gere um usando o comando fornecido.

```
show.o: show.c show.h
gcc -c show.c
```

4. Para gerar o arquivo objeto main.o, precisamos de (versões atualizadas de) main.c, bibl.h e de show.h. Se main.o não existir, ou se estiver desatualizado, gere um usando o comando fornecido.

```
main.o: main.c show.h bibl.h
gcc -c main.c
```

O elemento básico do arquivo makefile é uma combinação

```
arquivo: dependências
comando
```

em que aparece o nome de um arquivo, acompanhado de sua lista de dependências, para que se possa determinar quando o arquivo deve ser atualizado, e aparece também o comando que faz a atualização, se for necessária. (Note que o comando é precedido do caracter de tabulação horizontal, “Tab”.) Nosso makefile é composto por quatro destas combinações:

```
# EDA: makefile para prog
prog: main.o bibl.o show.o
    gcc -o prog main.o bibl.o show.o
main.o: main.c show.h bibl.h
    gcc -c main.c
show.o: show.c show.h
    gcc -c show.c
bibl.o: bibl.c bibl.h
    gcc -c bibl.c
```

(A primeira linha, iniciada por # é um comentário usado para identificar o arquivo.)

Algumas destas combinações podem ser simplificadas, porque algumas dependências e alguns comandos são considerados óbvios pelo programa make:

```
# EDA: makefile para prog
prog: main.o bibl.o show.o
    gcc -o prog main.o bibl.o show.o
main.o: show.h bibl.h
show.o: show.h
bibl.o: bibl.h
```

ficando implícito que um arquivo `x.o` sempre depende de `x.c`, e pode ser obtido através da compilação de `x.c`.

Para a maioria dos casos que encontraremos em ED, uma *makefile* correta pode ser obtida através da adaptação da *makefile* deste exemplo. Para casos mais complicados, será necessário consultar a documentação do seu compilador ou da ferramenta *make*.

A.4. Compilando e ligando o programa usando um ambiente integrado de desenvolvimento (IDE)

Um ambiente integrado de desenvolvimento é uma ferramenta que combina um editor, um compilador, um ligador, e algumas facilidades para executar e para depurar programas. Neste tipo de ambiente, a *makefile* é substituída por um projeto (*project*), um conceito um pouco diferente, mas que tem finalidade semelhante. Na maioria dos casos, só é necessário criar o projeto, com um nome apropriado, e incluir nele os arquivos que precisam ser compilados diretamente, como `bibl.c`, `show.c` e `main.c` no nosso exemplo. O próprio ambiente se encarrega de verificar as inclusões de arquivos, as dependências entre eles, e de forma semelhante ao *make*, só executa as operações necessárias para a construção de um executável.

Para construir e executar o programa de nosso exemplo, num ambiente integrado, teríamos os seguintes passos:

1. usaríamos as funções de edição para preparar (e salvar) os cinco arquivos fonte.
2. criaríamos um projeto (por exemplo com o nome `prog`) incluindo nele os arquivos `bibl.c`, `show.c` e `main.c`.
3. com o projeto `prog` aberto, usaríamos o comando executar, “*run*”. Este comando dispararia as ações de compilação e de ligação necessárias, emitindo as mensagens de erro correspondentes. No caso de inexistência de erros, o executável correspondente seria gerado e executado. Erros, se existirem, serão sinalizados.

Em alguns ambientes, o arquivo do projeto é um arquivo texto, que deve ser preparado diretamente no editor, com os nomes dos arquivos que devem fazer parte do projeto, mas na maioria das vezes o arquivo do projeto não pode ser editado diretamente, e deve ser criado usando comandos disponíveis no ambiente com essa finalidade.

A maioria dos ambientes tem também facilidades para depuração (*debugging*), que permitem a execução de um programa sob monitoração contínua, passo a passo. Dessa forma, podemos saber a cada instante o próximo comando que vai ser executado, observar os valores das variáveis, e identificar eventuais problemas na execução.

A.5. Compiladores C de baixo custo.

Na disciplina de Estruturas de Dados, a linguagem de programação usada é C, como padronizada internacionalmente pela ISO, a partir de 1990. No caso da disciplina de Estruturas de Dados, o padrão deve ser seguido em todos os momentos. Isto significa, em particular que, em todos os trabalhos da disciplina, será considerado um erro o uso de facilidades não permitidas pelo padrão.

Em princípio, assim, o melhor compilador para uso na disciplina seria um compilador que aceitasse a linguagem C padrão, sem nenhuma construção ou biblioteca adicional.

Como o foco da disciplina é nas técnicas de programação básicas e nas estruturas de dados usadas para na implementação dessas técnicas, não precisamos de um compilador com bibliotecas extremamente especializadas, ou de um compilador que também aceite programas na linguagem C++. Por exemplo, entre os compiladores mais completos atualmente disponíveis são, encontramos compiladores das linguagens C e C++ para Windows, que, além de permitir a programação na linguagem C, permitem a programação orientada a objetos na linguagem C++, e oferecem facilidades para manipulação de janelas e outros elementos de interfaces gráficas, interagindo diretamente com o sistema operacional Windows. Sem contestar de nenhuma maneira sua utilidade no contexto para o qual foram desenvolvidos, alertamos que o uso com todas estas facilidades adicionais pode ser complicado para o iniciante em programação.

Embora seja possível usar qualquer compilador de C para tratar os programas simples que vamos escrever nesta disciplina, um compilador que oferece facilidades adicionais pode ser mais difícil de usar, principalmente para iniciantes na programação de C. Por exemplo, se um compilador aceita programas em C++ um aluno pode utilizar em um dos trabalhos da disciplina alguma facilidade de C++ não disponível no C padrão, sem que o aluno seja advertido deste fato, e o trabalho do aluno terá uma nota ruim, uma vez que será recusado por um compilador de C padrão.

Para facilitar a obediência ao padrão, os manuais contêm, para cada facilidade oferecida, informação sobre sua aderência ao padrão.

Fazemos a seguir alguns comentários sobre compiladores C que podem ser usados na disciplina de ED.

Turbo C, da Borland.

O compilador mais simples de se obter é certamente o Turbo C 2.01, uma versão bastante antiga de compilador C da Borland, que pode ser encontrado no “Museu da Borland”, <http://community.borland.com/museum>. O *software* completo para instalação ocupa três disquetes de 1.44M.

Trata-se de uma versão para DOS, que oferece um ambiente integrado com as facilidades mencionadas na seção anterior. Entretanto, principalmente por não permitir o uso de um *mouse*, é considerado difícil de usar. Seu conceito de projeto é extremamente simples, sendo apenas um arquivo de texto com os nomes dos arquivos que devem ser compilados diretamente.

Entretanto, não se trata de um compilador de C padrão. Em particular, vários dos programas apresentados como exemplos durante o curso não são compilados pelo Turbo C. Por esta razão, este compilador deve ser usado com cuidado, recomendando-se a verificação de todos os trabalhos da disciplina em um compilador padrão de C, antes de sua submissão.

Gnu C Compiler, ou gcc.

Este compilador acompanha as diversas distribuições do sistema operacional Linux. Ainda que não esteja disponível um ambiente integrado, o processo de desenvolvimento é facilitado pelo uso do X-Windows (janelas para Linux) e do editor

Emacs, que tem facilidades para tratamento de programas em C, e que acompanha as distribuições de Linux.

Embora o sistema operacional Linux (juntamente com o compilador gcc) esteja disponível gratuitamente na Internet, normalmente é preferível comprar um CD com o sistema operacional, e evitar as longas horas necessárias para transferir o *software* e a documentação pela rede. Além disso, normalmente os CDs trazem *software* adicional que facilita muito a instalação do Linux. Estes CDs podem também ser comprados em pacotes que incluem também manuais de instalação e de uso, ou junto com livros e revistas. Já estão disponíveis distribuições em português.

O sistema operacional Linux pode ser instalado em uma máquina juntamente com outro sistema operacional como o Windows, escolhendo-se ao ligar a máquina qual o sistema desejado (“*dual boot*”). Ao contrário do que acontece com o Windows, é possível instalar Linux em uma máquina relativamente pequena, como um 486.

O *home site* do Linux é <ftp.sunsite.unc.edu/pub/Linux>, mas pode ser mais fácil obter informação dos sites-espelho (*mirrors*). Informação específica sobre o gcc pode ser obtida também em <http://www.gnu.org>.

Para alunos que pretendem seguir o curso de Engenharia de Computação, a familiarização com o sistema Linux é recomendada, tão cedo quanto possível.

Gnu C Compiler para DOS, ou gpp.

Este compilador é o resultado do transporte do gcc do Linux para DOS. Tem as mesmas facilidades do gcc, inclusive o editor Emacs, transportado para DOS. Para quem preferir, há também um ambiente integrado RHIDE, semelhante ao ambiente do Turbo C, da Borland, mas que tem a vantagem de permitir o uso do *mouse*.

Este compilador, com o ambiente RHIDE e o editor Emacs, pode ser obtido no site de DJ Delorie, em <http://www.delorie.com>. Não há necessidade de baixar os códigos fonte, bastando baixar os “binários”, ou seja, o código executável. A escolha entre RHIDE e Emacs é uma questão de preferência, mas acredito que o esforço para aprender a usar o RHIDE é menor.

A Fig. 1 mostra a interface gráfica, com a janela do projeto (*prog.gpr*) aberta. Como se pode ver, os arquivos incluídos no projeto são *bibl.c*, *show.c* e *main.c*. Note, na janela de *main.c*, que os diversos elementos que compõem o programa estão indicados em cores diferentes, para facilitar a verificação.

Para executar o programa definido por um projeto, basta abrir a janela do projeto, e executar Run | Run, ou seja, selecionar Run no menu Run. Outros comandos permitem compilar separadamente cada arquivo fonte, para procurar erros em cada um separadamente. Muitos comandos têm abreviações (ou atalhos) para simplificar seu uso. (Por exemplo, para executar um programa, podemos usar também a “tecla de atalho” *control-F9*.)

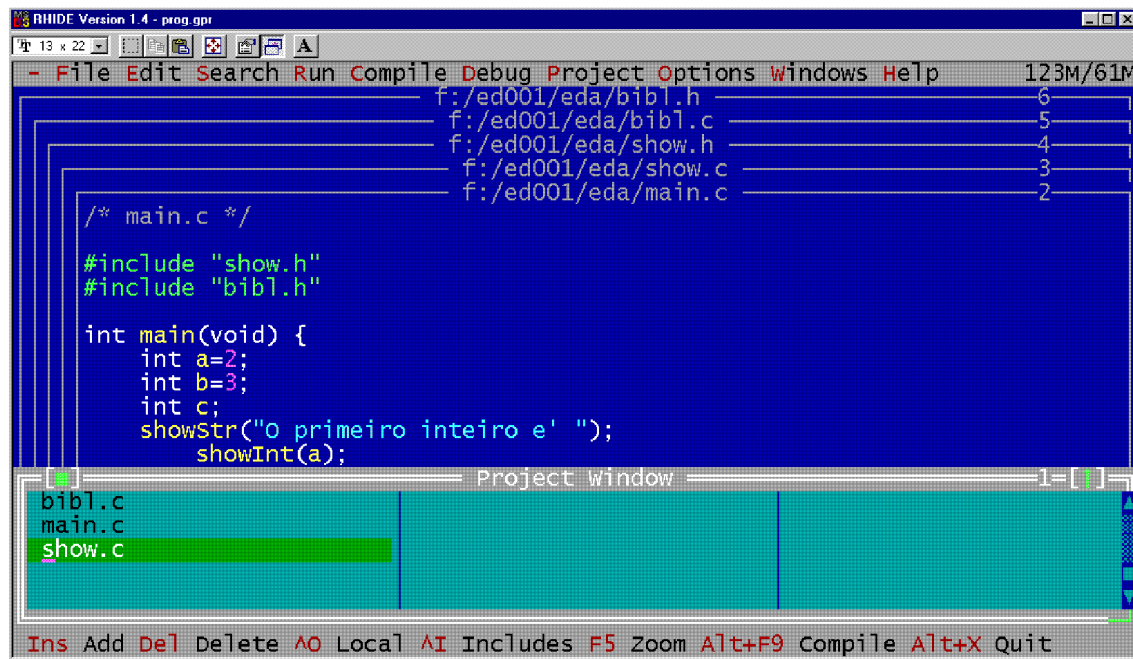


Fig. 1 – Janela do RHIDE com o projeto prog aberto

Para facilitar a organização dos arquivos no disco, recomendamos a criação de um diretório específico para cada projeto, em que ficam todos os arquivos associados: o arquivo do projeto (prog.gpr, no nosso exemplo), os arquivos fontes correspondentes e os arquivos acrescentados pelo processo de compilação.

Borland C/C++ 5.5

Este compilador foi recentemente liberado pela Borland/Inprise, <http://community.borland.com>, e é um bom compilador de C/C++ para DOS/Windows. Entretanto a liberação não incluiu a IDE correspondente, o que dificulta um pouco sua utilização.

Microsoft Visual C/C++ 6 (Introductory Edition)

Este compilador acompanha o livro “C++ How to program (Third Edition)”, de H. M. Deitel e P. J. Deitel, Prentice-Hall 2001. Como observado anteriormente, o uso de um compilador de C++ exige cuidados especiais.

A.6. Freeware, shareware, software aberto, copyrights.

Há várias maneiras de se obter um *software*, além da mais tradicional, de comprar em uma loja uma caixa com disquetes ou CDs e manuais. Por exemplo, hoje é possível, usar um cartão de crédito para comprar um programa pela Internet, “baixando” em seguida os arquivos do programa e de sua documentação.

O extremo oposto ao software comprado é o *freeware*, que é encontrado, gratuito, na Internet ou em CDs de revistas, fornecido sem nenhuma obrigação do autor ou do usuário. (Mesmo assim, a boa educação manda registrar o *software*, quando isso é solicitado pelo autor.)

Um meio termo interessante é o *shareware*, que é baixado gratuitamente, mas apenas para experiência, por um prazo determinado. O usuário se compromete a

desinstalar o software ou a registrá-lo, pagando o preço pedido, até o término do período de experiência. Em alguns casos, a versão baixada gratuitamente é uma versão limitada, sem algumas funções, sem documentação, ou que só funciona até uma certa data. Feito o registro, o usuário recebe a versão completa.

Em alguns casos, há mais de uma versão do mesmo *software*. Por exemplo, a versão completa (*full*) deve ser comprada, mas uma versão simplificada (*lite*) é gratuita.

Software aberto como o Linux e o gcc têm regras um pouco diferentes. O *software* pode ser obtido gratuitamente, com sua documentação, *acompanhado do código fonte*. Este *software* pode então ser modificado pelos usuários. Na realidade, espera-se que os usuários façam modificações e extensões, porque esta é a principal maneira pela qual este *software* é construído. Tanto o *software* como sua documentação podem ter centenas de autores, nenhum dos quais foi (diretamente) recompensado por isto, exceto pela sua inclusão como autor ou colaborador e a forma de sua participação. Já existe muito *software* aberto disponível, e sua qualidade costuma ser muito boa, porque a política do *software* aberto permite críticas bem detalhadas. Leia a respeito na página da Free Software Foundation (<http://www.fsf.org/>).

Qualquer que seja a forma pela qual um *software* é obtido, o usuário deve verificar com cuidado as condições que permitem sua utilização, e tomar cuidado para não violar os direitos dos autores, o *copyright*, ou, como às vezes é chamado no caso do *open software*, o “*copyleft*”. As restrições, se existirem, serão mostradas quando o usuário se registra para baixar o *software*, ou por ocasião da instalação. (No caso de *software* que acompanha uma revista ou um livro, as condições de uso do *software* podem estar no texto da publicação.) Normalmente, essas condições restringem a distribuição do *software*, o seu uso comercial, e se referem à obrigação de sempre indicar a procedência do *software*.

Uma última recomendação: *software* de procedência duvidosa pode estar incompleto, pode ser composto de partes incompatíveis, ou pode ter sido alterado maliciosamente, por exemplo pela adição de algum vírus de computador. Na dúvida, procure só usar *software* original.