Como organizar, compilar e depurar um programa C

Este capítulo é um resumo de informações práticas sobre a organização, a compilação, a correção de defeitos (= *debugging*), e a execução um programa em linguagem C. As informações valem para o sistema operacional GNU/Linux. Algo semelhante vale nos sistemas MAC e Windows (mas *não use Windows*, por favor!)

Como organizar um programa C

Todo programa em linguagem C é um conjunto de funções, uma das quais é `main`. A execução do programa consiste na execução de `main`, que tipicamente invoca outras funções do conjunto. Em um programa bem organizado, a função `main` tem caráter apenas *gerencial*: ela cuida da entrada de dados, da saída de resultados, e da chamada de outras funções (que fazem o trabalho realmente importante).

O conjunto de funções que constitui um programa reside em um ou mais arquivos, conhecidos como *arquivos-fonte* (= *source files*). Cada arquivo-fonte é um *módulo* do programa. O nome de cada módulo termina com `".c"`. O módulo *principal* é o que contém a função `main`. Nos exemplos a seguir, vamos supor que o programa reside em dois módulos,

`ppp.c` e `qqq.c`,

sendo `ppp.c` o módulo principal. Um bom programador tem a sensibilidade de espalhar as funções pelos módulos de uma maneira lógica, colocando num mesmo módulo as funções encarregadas de uma mesma parte do problema que o programa deve resolver.

Para cada módulo, exceto o principal, é importante escrever um *arquivo-interface* (= *header file*), contendo as macros, as declarações das eventuais variáveis globais e os protótipos das funções a que o módulo principal pode ter acesso. O nome da interface acompanha o nome do correspondente módulo: se o módulo é `qqq.c`, a interface é `qqq.h`. Assim, se `ppp.c` é o módulo principal, o programa completo consistirá nos arquivos

`ppp.c`, `qqq.c` e `qqq.h`.

Como compilar um programa C

Antes que o programa possa ser executado, ele deve ser *compilado*. A compilação transformará o conjunto de arquivos-fonte em um arquivo *executável*, também conhecido como *binário*.

Preparação

O primeiro passo é simples mas importante: abra um terminal, crie um diretório de trabalho, e vá para esse diretório:

```
~$ mkdir meu-diretorio-de-trabalho
~$ cd meu-diretorio-de-trabalho
~$
```

Coloque todos os arquivos do seu programa no diretório de trabalho. Confira o conteúdo do diretório de trabalho:

```
~$ ls -1
ppp.c
qqq.c
qqq.h
~$
```

Compilação

Para compilar os arquivos do seu programa e transformá-los em um arquivo executável xxx, basta invocar o compilador gcc. Por exemplo,

```
~$ gcc ppp.c qqq.c -o xxx
~$
```

A primeira fase da compilação é um *pré-processamento*, que cuida de todas as linhas de código que começam com "#". A segunda fase, compila os arquivos pré-processados.

O compilador emite uma lista dos erros porventura presentes nos arquivos-fonte. Corrija os erros com auxílio de um editor de texto e compile o programa novamente.

Warnings de compilação

Além de detetar erros que impedem a compilação, o compilador gcc pode emitir advertências (= *warnings*) sobre imperfeições do seu programa. Para forçar o gcc a fazer isso, use a opção `-std=c99` (para escolher o padrão C99 da linguagem C) e a opção adicional `-Wall`:

```
~$ gcc -std=c99 -Wall ppp.c qqq.c -o xxx
```

(Sugiro também usar as opções `-Wextra`, `-Wno-unused-result`, `-Wpedantic` e `-O0`.) Se o programa usa a biblioteca `math`, ou seja, se tem um `#include <math.h>`, acrescente um `-lm` ao final da linha de comando.

Os warnings devem ser levados muito a sério. Corrija as causas das advertências e compile o programa novamente. (Felizmente, a longa lista de opções e nomes de arquivos não precisa ser redigitada: basta usar a tecla ↑.)

O ciclo compilar–corrigir–compilar deve ser repetido até que não haja mais *nenhum warning*. Só depois disso, o programa xxx estará pronto para ser executado.

Como executar um programa compilado

Agora que a compilação foi bem-sucedida, o programa contido no arquivo xxx pode ser executado:

```
~$ ./xxx
```

(O prefixo "`./`" é importante para indicar que se trata do arquivo xxx que está no seu diretório de trabalho e não de um eventual homônimo de xxx que está em algum outro diretório.)

Se a função `main` do seu programa tiver parâmetros, digite os correspondentes argumentos na linha de comando. Por exemplo, se o seu programa tiver três argumentos, digite

```
~$ ./xxx arg1 arg2 arg3
```

Tipicamente, um ou mais desses argumentos são nomes de arquivos de dados que o programa xxx deve processar. Recomendo colocar todos esses arquivos de dados no seu diretório de trabalho antes de executar o programa.

Make e Makefile

Para automatizar um pouco a compilação do seu programa, reduzindo assim a digitação enfadonha de longas linhas de comando, use o utilitário `make`. No caso do exemplo discutido acima, coloque um arquivo `Makefile` muito simples no seu diretório de trabalho e diga

```
~$ make xxx
```

para compilar o programa.

Depuração (debugging) e GDB

"The art of debugging is figuring out
what you really told your program to do
rather than what you thought you told it to do."
— Andrew Singer

"Debugging is twice as hard as writing the code in the first place.
Therefore, if you write the code as cleverly as possible, you are,
by definition, not smart enough to debug it."
— Brian W. Kernighan

Finalmente, seu programa passou pelo compilador sem erros e sem warnings! Infelizmente isso não significa que o programa está livre de *bugs* (= defeitos = erros).

Rode o seu programa com dados de teste. As primeiras tentativas podem terminar abruptamente em um *crash*, como um *segmentation fault* (tentativa de acessar uma posição de memória que está fora dos limites alocados para a execução do seu programa), por exemplo. Para encontrar o bug que causou o crash, examine os arquivos-fonte do programa e os resultados da tentativa de execução. Use seu espírito de detetive. Se encontrar o bug, corrija-o e recompile o programa.

Se a tentativa de encontrar o bug manualmente não tiver sucesso, use o poderoso caça-bugs GDB Debugger. (Antes, é preciso recompilar o programa com a opção `-g` do `gcc`.)

(Veja Julia's drawings: How I got better at debugging.)

Vazamento de memória

Um bug (= defeito = erro) particularmente desagravável é o vazamento de memória (= *memory leak*): o programa funciona muito bem quando a quantidade de dados é pequena mas esgota a memória disponível e termina num crash quando a quantidade de dados é grande. (Isso pode ser constatado, por meio do *system monitor* ou *task manager*, observando o gráfico da quantidade de memória em uso.) O vazamento de memória acontece tipicamente quando o programador esquece de desalocar a memória alocada por `malloc`.

A melhor maneira de descobrir onde está o vazamento é examinar os arquivos-fonte do programa com muita atenção. Se essa análise manual não for suficiente, você pode recorrer ao Valgrind.

Perguntas e respostas

- PERGUNTA: Existem outros compiladores para a linguagem C além do `gcc` da GNU?
  RESPOSTA: Tente o compilador Clang.

Veja *C Programming Language Standard* e *Compiling a C program* em GeeksforGeeks.

Atualizado em 2018-03-19
https://www.ime.usp.br/~pf/algoritmos/