# Lista de Figuras

# Sumário

Introdução.

Toda manhã, na Africa, uma corça se levanta e sabe
terá que ser mais rápida que o mais rápido dos leões,
ou será morta.
Toda manhã um leão se levanta e sabe
terá que superar a mais lenta das corças,
ou morrerá de fome.
Não importa se você é leão ou corça
quando o sol se levantar,
é melhor sair correndo.
- autor desconhecido

## Como usar este livro.

Para começar, sugerimos que não leia, agora, esta introdução. Leia primeiro o resto do livro, depois a introdução, porque, lhe confessamos, primeiro escrevemos o livro, depois a introdução. Mas se você quiser insistir, faça uma leitura rápida e depois volte para ler com mais cuidado. Você vai ver que, então, vale a pena.

Este livro tem dez capítulos em que lhe apresentamos as técnicas básicas para programar na linguagem $\mathcal{C}$, mas é preciso enfatizar, *você vai apenas se iniciar na linguagem com este livro.*

O livro está divido em duas partes. Na primeira, vamos apresentar-lhe $\mathcal{C}$ em português por duas razões:

- para vencer a dificuldade psicológica com o Inglês, creamos um arquivo que traduz os comandos da linguagem $\mathcal{C}$ para o Português, de modo que você se inicie sem a dificuldade linguística;

- para mostrar-lhe que com $\mathcal{C}$ podemos facilmente construir outra linguagem, neste caso é "$\mathcal{C}$ em português". Isto mostra o poder da linguagem.

Mas você não deve se enganar com o apoio *linguístico* e nem queremos induzí-lo no erro de que *é possivel viver sem o inglês.* Por esta mesma razão vamos manter os programas traduzidos junto com os programas naturais em $\mathcal{C}$ de modo que você vá aos poucos se habituando com as palavras da linguagem em Inglês. Na segunda parte usaremos apenas os comandos em inglês.

O conteúdo das duas partes do livro, em linha gerais é o seguinte:

1. A primeira parte, constituida dos cinco primeiros capítulos, deverá deixá-lo escrevendo pequenos programas em $\mathcal{C}$ e todos os exemplos serão em "português" com a tradução ao lado. Depois você tomará sua decisão, se

quiser continuar escrevendo seus programas em Inglês, como é habitual, ou continuar a escrevê-los em português.

Observe que os programas em "português" rodam da mesma forma como os programas em "inglês", eles não são de mentira.

2. Na segunda parte vamos avançar em profundidade em algumas direções. A partir dai só apresentaremos pedaços de programas, porque os programas inteiros você pode conseguí-los num disco ou via Internet, veja como fazer na bibliografia, ou envie *e-mail* para

   `tarcisio@e-math.ams.org`

   solicitando os programas. Você pode usar este enderço para consultas rápidas, mas, por favor, não espere que lhe possamos dar um curso particular via internet.

   Se você quiser continuar programando em "português" esta será uma opção sua e até lá você terá aprendido como fazer.

O método que vamos usar no livro se assemelha àquele que usaram quando você era pequeno, para aprender a sua lingua materna. Mostraram-lhe várias vezes o mesmo objeto, *cadeira*, e lhe disseram: "*cadeira*", aí você aprendeu a diferença "lógica" entre uma *cadeira* e uma *mesa*.

Vamos lhe mostrar pequenos programas, pedir que você os rode num computador em que o $\mathcal{C}$ esteja instalado[1]. Depois iremos comentar os programas e lhe indicar como você os pode alterar, e assim por diante.

Parte do método consiste do estilo com que os capítulos foram escritos: há superposição entre eles, quer dizer, o mesmo assunto, o mesmo conceito, aparece várias vezes, aumentando de intensidade nos capítulos de "número maior". É o método de explicação lógica da diferença entre *cadeira* e *mesa*, de tanto falar, termina ficando claro o que as coisas são. Você será inclusive convidado a pular para os capítulos mais avançados com frequência e, não tenha dúvida em fazê-lo, se achar que está bem lá na frente, não precisa voltar atrás...

### Como rodar o $\mathcal{C}$ ?

Se você puder escolher, use `LinuX` e a linguagem gratuita $\mathcal{C}$ que vem com este sistema operacional. Se você tiver ainda alguma chance de escolher, mesmo tendo que trabalhar dentro do Windows, (poucas chances de escolha...), sugerimos

- use o $\mathcal{C}$ da fundação `GNU`, procure em `www.gnu.org` ou envie e-mail para o autor. Veja abaixo instruções mais detalhadas,

[1] se você tiver acesso a um computador rodando `LinuX`, então o $\mathcal{C}$ estará com toda certeza instalado

- dê preferência ao $\mathcal{C}$ da Borland, `BC` que todos que o analisam consideram melhor que o Microsoft C,
- se você não tiver mesmo nenhuma outra opção, use o que tiver à mão, mas aprenda o $\mathcal{C}$.
- Se você usa `Linux` e *por alguma razão complicada* precisar de usar `Borland C`, você pode fazê-lo sob `dosemu`. Foi assim que as versões do programas para `BC` foram testadas por mim.

Este livro foi escrito em cima do `gcc`, Gnu C Compiler (o Compilador C da Fundação para Software Livre, [2] Free Software Foundation, FSF). A FSF criou também uma versão do gcc para rodar em DOS e em Windows de modo que se você não tiver comprado um pacote de $\mathcal{C}$, você pode obter um, gratuito, diretamente da FSF no endereço `http://www.gnu.org` procure "What we provide"e você vai ser direcionado para os diversos programas que feitos sob o patrocínio desta fundação, em algum lugar você vai encontrar `djdev` que é o nome do `gcc` para DOS/Windows. .

## O que é $\mathcal{C}$ ?

Há muitos mitos envolvendo a linguagem C, entre eles destacamos:

- **É uma linguagem difícil.** Você verá que é fácil iniciar o seu aprendizado em C. Mas seria uma mentira dizer-lhe que o conteúdo deste livro é suficiente para que se torne um exímio programador. Prometemos que ao término deste livro você poderá estar fazendo alguns programas interessantes e com muita vontade de continuar. Na bibliografia você vai encontrar dicas de como fazer isto.
- **É uma linguagem perigosa, você pode estragar o computador.** Não é *perigosa*, mas é *poderosa*. Se pode dizer, *sem perigo de erro maior*, que tudo que roda hoje nos computadores foi feito em $\mathcal{C}$ ou foi feito com alguma *ferramenta*, uma outra linguagem, que foi feita em C.
- E **de fato**, você pode, com facilidade, travar o computador com um programa errado. Você também pode deixar o sistema operacional confuso gravando algum dado impróprio em local indevido de memória, mas na pior das hipóteses a solução para o problema vai consistir em desligar a máquina e depois ter cuidado com o programa que causou esta confusão.

  Na absoluta pior das hipóteses, você pode ter que instalar tudo de novo, se o seu programa houver se intrometido por cima do sistema operacional

[2] GNU é uma sigla que representa a FSF e também o nome de um tipo de antílope, "large African antelope having a head with horns like an ox and a long tufted tail", copiado do meu dicionário gratuito produzido pelo Lab. de Ciências Cognitivas de Princeton.

gravado no disco... mas para fazer isto só um programa bem avançado e muito mal intencionado.

Claro, logo aqui no começo podemos dizer `quem` pode causar tais transtornos para que você aprenda a manipular com cuidado o vilão: são as variáveis *do tipo* `ponteiro` porque elas fazem referência aos endereços das variávei na memória `RAM`[3]. Consequentemente, se você mandar escrever dados muito grandes em uma variável de tamanho pequeno, haverá uma invasão em áreas de memória e pode ser difícil de predizer as consequências desta invasão.

É isto que se chama de *endereçamento indireto* . A maioria das linguagens de programação não usa este recurso, nelas você pode apenas fazer endereçamento direto, usando o próprio nome da variável. Veja o seguinte exemplo:

**Exemplo: 1** *Endereçamento indireto e indireto*

`numero` = *23; // endereçamento direto*

`&numero` ← *23;// endereçamento indireto*

*A primeira* atribuição *é a comum nas linguagens de programação, foi atribuido o valor 23 diretamente à variável* `numero`*. Na segunda linha estamos dizendo que o valor 23 seja "associado"ao endereço de número.*

*Não é assim que se faz em* $\mathcal{C}$*, veja o programa*

`endereco_indireto.c,`

*mas não se preocupe com entendê-lo completamente agora. Estamos lhe dizendo que olhe o programa apenas para contrabalançar as duas linhas acima que não são reais em programação, apenas uma tentativa de transmitir-lhe o que significa* endereçamento indireto*. No programa*

`endereco_indireto.c`

*você pode ver como é que realmente se faz.*

Em $\mathcal{C}$ também fazemos atribuições diretas de valores nas variávei, mas, além disto, $\mathcal{C}$ pode acessar, quando você usar ponteiros, a memória de forma absoluta, e aí se encontra o risco de que você mande escrever por cima de alguma parte do sistema operacional, por exemplo...e neste caso, com certeza a máquina vai parar. Desligando-a e novamente ligando, uma versão nova do sistema operacional vai ser instalada a partir do disco e tudo voltará ao normal. Este é o grande dano que obviamente deve ser evitado e por isto *primeiro entenda ponteiros antes de usá-los*.

Mas aqui você irá aprender o que é um ponteiro, vai aprender a compreender o que pode estar acontecendo e dominar os poderes da linguagem. Você sabe que pode levar um choque elétrico pegando de mal jeito nos fios, mas nem por isso você prefere viver no escuro...

[3]Random Access Memory, é a memória que você adquire a mais ou a menos para sua máquina e na qual os programas tem direito a fazer registros.

**Observação: 1** *A função* `scanf()` *e o direcionador "*`&`*".*

*O símbolo "&" se chama algumas vezes "redirecionador"de memória, porque ele associa endereço e memória. Algumas funções da linguagem C fazem atribuição de dados via endereço, é o caso da função* `scanf()` *e poristo ela representa um problema, com frequência. Ela exige, um "direcionador de registro",* `&` *na frente de algumas variáveis. Sua omissão fará com o compilador o advirta do erro e se você não levar a sério a advertência pode acontecer que com o valor lido seja colocado numa posição de memória difícil de prever. Se o sistema operacional não tiver um bom controle do uso da memória, e este é o caso do "windows", isto pode levar a sobreposição de uma variável do sistema e consequentemente a uma parada cardiáca violenta do mesmo... mas em geral o "ctrl-alt-del"resolve o problema e o "windows"vai lhe brindar o disco com um monte de lixo quando se re-iniciar.*

*Evite de esquecer o "&" antes das variáveis quando usar* `scanf`*. Mas não precisa se assustar, o compilador que você estiver usando dentro, mesmo dentro do "windows", lhe fará uma advertência se você esquecer um "&" na primeira etapa da compilação do programa, tenha apenas o cuidado de levar a sério a advertência e corrija o esquecimento. Há outras funções que, como* `scanf()` *exigem o &. Tome o mesmo cuidado nestes outros casos.*

- Existem outras formas de copiar informações em lugares errados, elas serão identificadas mais adiante, e todas estão ligadas ao uso indevido do endereçamento de memória. Um exemplo comum como utilização de uma variável com tamanho maior do que deveria. Rodando programas em LinuX, o maior problema que isto pode causar consiste em deixar o programa inconsistente e podendo travar indesejavelmente o que pode em geral ser resolvido entrando n'outra "*área de trabalho*" e "matando" o programa mal comportado.

  A prevenção para este problema consiste no uso cuidadoso das variáveis segundo a declaração das mesmas. Claro, é verdade, se espera de um programador da linguagem $\mathcal{C}$ muita atenção no uso de variáveis.

O nosso objetivo consiste em deixá-lo em condições de escolher um dos caminhos seguintes:

- Se aprofundar em $\mathcal{C}$ para construir programas que executem tarefas difíceis com esta linguagem, *mas usando um outro livro*, não este. Na bibliografia você irá encontrar alternativas.

- Escolher uma outra linguagem, vamos lhe sugerir algumas, usando a experiência adquirida aqui. Queremos lhe dizer com esta segunda opção que $\mathcal{C}$ pode ser uma linguagem introdutória antes de você se definir por uma linguagem apropriada para o seu desenvolvimento, que pode ser em $\mathcal{C}$, mas há muitas outras para escolher. Ao final deste livro você deve se encontrar no ponto de fazer esta escolha.

- Iniciar o estudo de $\mathcal{C}$ pelos seus aspectos de linguagem de alto nível, deixando para o final os indicativos de como se aprofundar na linguagem.

E porque $C$ é tão importante, mesmo que finalmente você vá programar em outra linguagem? Algumas das respostas para esta pergunta são as seguintes:

- A primeira é aquela que já mencionamos algumas linhas atrás, *praticamente tudo que roda nos computadores hoje, ou é feito em* $\mathcal{C}$ *ou com alguma ferramenta que foi feita em* $\mathcal{C}$ e, como consequência, por trás de tudo isto sempre podemos encontrar as pegadas desta importante linguagem.

- Em geral, na solução de problemas computacionais se usa $\mathcal{C}$ como *uma linguagem final* para escrever na forma definitiva os algoritmos que estão rodando bem e sem erros e muitas vezes para escrever pequenos pedaços críticos do algoritmo, não o algoritmo todo. Quer dizer que se começa a escrever numa outra linguagem que, por alguma razão, é mais apropriada, e quando se conseguiu montar o *algoritmo*, funcionando, sem erros, se o traduz para $\mathcal{C}$ ou pelo menos parte dele é traduzido para $\mathcal{C}$.

- Outras vezes se escreve em $\mathcal{C}$ *uma outra linguagem de alto nível* na qual se produzem os programas. Neste caso, o que é comum fazer-se é, continuar espandindo esta outra linguagem com novos módulos escritos em $\mathcal{C}$. Esta é, possivelmente, o uso mais comum da linguagem $\mathcal{C}$.

- Seria um erro não mencionar aqui a estensão construida para $\mathcal{C}$ que se chama $\mathcal{C}++$. Esta é uma nova linguagem mas que admite $\mathcal{C}$ como uma *sub-linguagem*, quer dizer que você pode programar exclusivamente em $\mathcal{C}++$ mas você pode misturar as duas de uma forma conveniente. Outro exemplo é `Python` que é uma linguagem um pouco mais nova que $C++$ e que admite também $\mathcal{C}$ como uma linguagem de estensão. Mas aqui teriamos que fazer uma lista com uma dezena de linguagens, ou mais, para as quais isto é verdade.

- No índice remissivo você encontra *uso de* $\mathcal{C}$, remetendo-o para outros pontos no livro onde mostramos pequenos exemplos de uso da linguagem na construção de outras ferramentas. Não espere, obviamente, encontrar nada revolucionário a nível de um livro introdutório, como este...

Por todas estas razões é importante conhecer a linguagem $\mathcal{C}$.

Por outro lado ela é fácil de se aprender e serve como uma primeira linguagem de programação. É este o intuito principal deste livro: apresentar $\mathcal{C}$ como uma primeira linguagem de programação. Por exemplo, *é facílimo escrever programas em Português que rodem em* $C$ e seria um pouco mais difícil de fazer o mesmo em qualquer outra linguagem de programação.

### Observações e outros meios de comunicação.

O texto é completado com observações de dois tipos. Um dos tipos se chama claramente "observação", o outro são as notas de rodapé.

Você deve ler as observações na ordem em que elas aparecerem, mas sem lhes dar muita importância numa primeira leitura.

Para lhe permitir uma busca mais acurada de informações, o livro tem um índice remissivo alfabético, ao final, em que todos os conceitos que surgem nas observações se encontram *indexados*, de forma que você poderá facilmente retornar a eles quando achar necessário. Também se encontram *indexadas* todas as palavras-chave do texto.

Quando falamos usamos encenação para completar o sentido das palavras usadas no discurso: mexemos as mãos, o corpo e alteramos a entonação da voz. Para suprir um pouco deste teatro usaremos uma convenção tipográfica: *texto em itálico* representa material que você deve olhar com cuidado, possivelmente não está definido ainda e estamos usando a concepção intuitiva do termo. Quando usarmos `texto tipográfico` estaremos fazendo referência a um termo técnico já definido anteriormente ou considerado bem conhecido como tal. As palavras da linguagem $\mathcal{C}$ serão escritas `no estilo tipográfico`. Quan-do usarmos letra pequena estamos lhe querendo dizer que o assunto é polêmico e que há muito mais coisa para ser dito do que estamos conseguindo dizer naquele momento. Usamos texto sublinhado para chamar sua atenção de um detalhe que poderia passar desapercebido, tem o mesmo sentido **texto em negrito**.

### Existe alguma técnica para programar bem?

Bom, chamar de *técnica* é um certo exagero, mas o que vamos dizer agora e repetir umas tantas vezes ao longo do livro, pode aos poucos se tornar *numa técnica de programação.*

O segredo consiste em fazerem-se pequenos programas. Costuma-se dizer que um programa nunca deve ser maior do que a tela do micro. É possivel programar assim com as linguagens que temos hoje, porque elas são modularizadas, quer dizer, um programa é um aglomerado de pequenos programas. Você vai aos poucos entender o que queremos dizer, mas torne esta idéia uma *obsessão*: nunca faça um programa que passe em tamanho da tela do micro.

O maior problema de um programador são os erros que teimam em se esconder, como "insetos", no interior dos programas. Os americanos os chamam de `bug`s. Quanto maiores os programas, mais lugar os insetos encontram para se esconder, acredite. Quando o programa fica do tamanho da tela, a gente consegue rapidamente detectar os "insetos" e então não é necessária nenhuma `técnica de dedetização` para consertar programas defeituosos. Mais à frente vou chamar sua atenção dos ambientes de programação com que você poderá trabalhar, eles estão equipados com instrumentos para fazer esta "dedetização"nos programas.

Você pode muito bem viver sem estes "instrumentos" de análise de programas se aprender, desde o início, a programar bem, e, por outro lado, se o seu programa for ruim, nem elas adiantam muito... *é tão difícil consertar um programa mal feito, que é mais fácil re-aprender a programar e fazer outro programa.*