Capítulo 7

Os tipos básicos de dados

Neste capítulo vamos discutir os *tipos de dados* básicos de C. Entre eles existe um de particular importância que deixamos para a última seção, `ponteiros`, porque ele tem o que ver com todos os demais e também porque ele é de uso intenso nos programas em C. A seção sobre `ponteiros` foi escrita a parte e imagino que deve ser lida independentemente das demais, inclusive há "*ponteiros*"[a] de outras partes do livro para ela.
Um agregado como 132345665, para a linguagem $\mathcal{C}$, é semelhante ao agregado $hfgfabccb$. $\mathcal{C}$ manipula tais agregados segundo certas regras incluidas no compilador de uma forma tal que podemos chamar $\mathcal{C}$ de linguagem, porque, como as *linguagens naturais* $\mathcal{C}$ obedece a uma sintaxe e tem alguma semântica de muito baixo nível.

Na medida em que você declarar corretamente os dados, $\mathcal{C}$ irá aplicar a estes dados as regras sintáticas correspondentes e, com igual importância, irá separar espaço de memória do tamanho certo para guardar estes dados. É esta a importância do *tipo de dado* na declaração de uma variável: determinar o espaço que ela vai ocupar na memória e portanto o segmento de memória que será usado.

[a] Este uso da palavra "ponteiro" é técnico, mas nada tem a ver com seu significado dentro da linguagem $\mathcal{C}$. Aqui a palavra ponteiro quer dizer "link", uma ligação entre duas idéias como num texto em *html*...

7.1 Os números em $\mathcal{C}$

Este livro não pretende discutir questões matemáticas, aqui, os números são *tipos de dado* de uma determinada linguagem de programação e isto é diferente do conceito de número em Matemática, embora, sob algum aspecto, os inteiros em $\mathcal{C}$ tenham aspectos que somente se atinge em cursos mais avançados de Matemática, quando se estuda congruência, em Álgebra.

7.1.1 Os números inteiros

Vamos começar com os números inteiros.
A forma de encarar os números, numa linguagem de programação, difere daquela com que um matemático vê este tipo de *objeto*. Há linguagens de programação em que a visão matemática de número chega a ser aproximada, `Python, LISP, calc` por exemplo, relativamente a números inteiros. Em `Python` ou em `LISP` o limite para se escrever um número inteiro fica por conta da memória da máquina.

Em $\mathcal{C}$, os números formam um conjunto finito e usa uma aritmética apropriada para um *conjunto finito de números*, a da congruência módulo p. Este poderia ser um tópico para um livro inteiro, de modo que vamos ter que cortar os horizontes para escrever dentro do escopo deste livro, mas é preciso que o leitor saiba disto. O local onde você pode se expandir a respeito deste assunto é num livro de Álgebra.

Um número inteiro é um objeto que vai ocupar um determinado espaço da memória do computador e ao qual certas regras serão aplicadas. Este é um dos pontos em que a linguagem $\mathcal{C}$ difere de máquina para máquina. Em `LinuX`[1] o maior número inteiro positivo que o `gcc` reconhece é 2147483647.

No `BC`, se você escrever `int` na área do editor (dentro de um programa editado), e colocar o cursor sobre esta palavra, apertanto `Ctr-l F1`[2], você vai receber uma ajuda no contexto, específica, sobre `int` na qual lhe vai ser dito qual é o maior inteiro que o `BC` reconhece.

Como eu uso somente `LinuX`, me vejo na impossibilitade de verificar qual é o maior inteiro reconhecido pela linguagem $\mathcal{C}$ em outros sistemas. Porém o método indicado acima vai lhe mostrar qual é a capacida numérica do `BC` e, certamente, o método funciona em outros sistemas.

Veja o resultado da operação *aritmética para números finitos*

$$2147483647 + 1 = -2147483648.$$

Voce pode verificar isto rodando o programa

```
#include <stdio.h>


 int   main()
   {
       printf("%d", 2147483647+1);
```

[1] Você pode encontrar esta informação em /usr/lib/gcc-lib/i486-linux/2.7.2.3/include/limits.h

[2] se não funcionar, deixe o cursor sobre a palavra e, acionando o `help` escolha `Topic search`

```
    return 0;
}
```

apesar do aviso que o compilador lhe vai dar de que o programa produz um "overflow", (estouro de dados). Você pode ignorar este mensagem, tranquilamente. A mensagem de erro ocorre porque, embora $\mathcal{C}$ saiba bastante Álgebra, não se encontra convencido de que Álgebra é verdadeira... a linguagem calcula corretamente o valor mas a operação está ultrapassando o limite de memória reservado para números inteiros, eis a razão do *overflow* que significa "*está derramando*"...

Um programa que executa tarefa semelhante é `inteiros.c`. Leia o programa, rode-o, volte a lê-lo.

Por exemplo observe que

$$2^{32} = 4294967296 \; ; \; 2^{31} = 2147483648$$

entretanto isto não parece estar ligado ao programa que você acabou de rodar.

Vamos pensar de outra forma. Lembrando a *Análise combinatória*, se você tiver dois *caracteres*, $\{0,1\}$, para *arranjar* com repetição em uma matriz com 32 posições, o número de arranjos (com repetição) seria

$$2^{32} = 4294967296.$$

Agora separe um `bit` para indicar se o número é positivo ou negativo.

Falei de `bit`, entenda que é uma posição na matriz de 32 posições acima.

Separar um `bit` lhe faz perder a possibilidade de usar $\{0,1\}$ uma vez, portanto o conjunto dos `números inteiros` que se podem assim representar é

$$2^{32}/2 = 4294967296/2 = 2^{31} = 2147483648.$$

Ainda tem o zero, e assim a aritmética de inteiros em $\mathcal{C}$ vai de

$$-2147483648 \ldots -1, 0, 1 \ldots 2147483647$$

sendo esta a razão do resultado que você obteve com o programa `inteiros.c`. Quando pedirmos $2147483647 + 1$ $\mathcal{C}$ responderá com -2147483648 e sucessivamente $-2147483648 + 1 = -2147483647 \ldots$

Como um `byte` corresponde a oito `bits` então os inteiros no `gcc` ocupam 4 bytes de memória:4 x $8 = 32$.

Uma outra forma de obter a mesma informação junto com outras informações ligadas aos tipos de dados "escalares" pode ser consultando o manual, ver no índice remissivo "Libc". Veja também o último capítulo.

Se você quiser saber o espaço ocupado na memória por um inteiro, rode o programa `inteiros.c`. e para compreender exatamente o que significa o "tamanho" de um tipo de dado, altere manualmente o número que aparece no programa `inteiros.c`, por exemplo, retirando alguns algarismos[3], e voltando

[3] A versão do programa, no disco, é a mais recente e pode ser diferente da que se encontra no livro.

a rodar o programa. Não tire muitos algarismo porque o programa poderá demorar muito rodando...mas, neste caso, tem o `Ctrl-C` para pará-lo.

Exercícios: 33 *Experiência com inteiros*

1. *Faça um programa que adicione dois inteiros fornecidos pelo teclado.*

   Solução: `inteiros.c`

2. *Altere o programa* `inteiros.c` *para que um número seja lido pelo teclado e seja testado.*

   Solução: `inteiros01.c`

3. *Se você tiver usado números muito grandes ao rodar* `inteiros01.c`*, vai aparecer uma conta* meio estranha*. Tente encontrar uma explicação. Rode* `inteiros02.c` *e não dê muita importância à reclamação do compilador sobre o tamanho das constantes dentro do programa. Neste caso pode ir em frente.*

4. *Introduza em* `inteiros.c` *um teste para verificar se uma soma de dois* inteiros positivos *ainda é positivo na* aritmética finita *do* `gcc`*.*

   Solução: `inteiros03.c`

5. *Altere o programa* `inteiros.c` *para que ele rode a partir de um* inteiro grande *acrescentando mais uma unidade até chegar ao maior inteiro do seu sistema. Ver sugestões em* `inteiros01.c`

6. *Faça um programa que receba dois inteiros e depois lhe pergunte qual a operação aritmética para executar com eles e que ela seja executada.*

7. *Evolução de um programa*

   (a) *Quebre o programa*[4] `quatro_operacoes.c` *em quatro módulos que devem ser chamados a partir de um "menu".*

   (b) *Torne o programa* `quatro_operacoes.c` *numa máquina de calcular permanente na memória.*

   (c) *O programa* `quatro_operacoes01.c` *tem um "lay-out" de baixa qualidade, reforme-o.*

   (d) *O programa* `quatro_operacoes01.c` *não usa memória para guardar as operações executadas, altere isto.*

   (e) *Inclua no programa*

   `quatro_operacoes02.c`

   uma informação de como parar o programa:

   "Ctrl-c"

   naturalmente...com o cursor na `shell` *em que você tiver rodado o programa.*

[4] no diretório do DOS, procure `qua_opr*.c`

(f) *Ofereça um meio mais evoluido para parar o programa*
`quatro_operacoes02.c`

Solução: `quatro_operacoes03.c`.

(g) *Infelizmente o usuário pode ser obrigado a digitar números antes que o programe pare... defeito do* `quatro_operacoes03.c`. *Corrija isto.*

(h) *O programa* `quatro_operacoes04.c` *tem um péssimo "lay-out", corrija isto se ainda não o houver feito.*

(i) *Refaça o programa* `quatro_operacoes04.c` *usando a função* `switch()`.
Solução:`quatro_operacoes05.c`

Sugestões:

1. *Quando os inteiros crescerem além do limite, tornam-se inteiros negativos, um teste para detectar isto pode ser num* < 0.
2. *Uma variável do tipo* `char` *pode receber os itens de um menu. Verifique que se* `char` *é aceito como tido de dados do* `switch`
3. *Máquina de calcular: ver o programa* `quatro_operacoes.c`.
4. *Modularização? analise a suite* `quatro_operacoesXX.c` *Use grep modulo quatro* para encontrar em que programas aparece a palavra chave "modulo" (sem acento).*
5. *Máquina de calcular permanente? Loop infinito... ver*
   `quatro_operacoes01.c`
6. *Use*
   `grep 'assunto'' quatro*.c`
   para encontrar o programa que você deseja, (os nossos programas tem um sistema de palavras-chave para buscas deste tipo):
   *grep 'assunto' *.c | more*

Uma das experiências feitas nos exercícios do bloco anterior foi a do tamanho ocupado por um inteiro. No `gcc` é 32 bits o espaço necessário. Pode ser até mesmo o inteiro 0, ele ocupa o mesmo espaço do `inteiro máximo`. Isto significa que `gcc` "planeja"[5] um espaço adequado para guardar números inteiros. Se o problema que você for resolver não precisar de números tão grandes, escolha um *tipo de dados* mais modesto. Caso contrário você vai gastar memória a toa.

A biblioteca "`limits.h`", que pode ser encontrada no diretório
'/usr/lib/gcc-lib/i386-linux/2.9XX/include/" [6],
contém as definições dos tamanhos de dados. As letras XX devem ser substituídas para bater com a versão atual do `gcc`. Você pode obter a versão digitando

`gcc -v`

no meu caso o resultado é

```
tarcisio@linux:~/tex/c$ gcc -v
Reading specs from /usr/lib/gcc-lib/i386-linux/2.95.2/specs
gcc version 2.95.2 20000220 (Debian GNU/Linux)
```

[5] Na verdade quem planeja é o programador!

[6] Este caminho pode variar de máquina para máquina, com algum programa de busca de arquivos, `mc`, por exemplo, você pode localizar esta biblioteca.

$XX = 5.2$ porque a parte final, 20000220, é a data da distribuição.

Lá você pode encontrar, por exemplo, no caso do `gcc`:

- LONG_MIN é o espaço ocupado por um 'signed long int'. é também o menor número inteiro do tipo "longo" reconhecido pelo `gcc`. Dito de outra forma, é também o menor número deste tipo que você deve usar para não gastar espaço à toa na memória. Ainda repetindo de outra forma, se você precisar apenas de números menores que 32 bits, então não deve escolher este tipo de dados.

  é o mesmo tamanho do tipo de dados `int`.
- LONG_MAX é o valor máximo que pode assumir um "inteiro longo".
- ULONG_MAX é o valor máximo que `gcc` entende para um inteiro sem sinal. Tem a mesma capacidade dos inteiros longos com sinal.
- LONG_LONG_MIN é o menor valor que pode ser alcançado por um inteiro longo com sinal. De outra forma, representa o espaço ocupado por este tipo de dados: 64 bits que é menor do que 2^{64}.

Retomando o que diziamos no início desta seção, os números no computador formam um *conjunto finito*, como não podia deixar de ser. Em algumas linguagens, LISP, Python, é possível manter esta barreira limitada pela memória do computador e até "pensar" que não existem barreiras... por um artifício especial que estas linguagens têm para trabalhar com aritmética. Não é o caso do $\mathcal{C}$ *embora* estas linguagens tenham sido construidas em $\mathcal{C}$.

É, lembre-se da introdução, há coisas que não podemos fazer com $\mathcal{C}$ mas que podemos fazer com *linguagens* ou pacotes que foram feitos em $\mathcal{C}$ ...

Observação: 27 *A conta de dividir*

A conta de dividir em `quatro_operacoesXX.c` *não está errada, apenas não é a divisão habitual. Quando você escrever em* $\mathcal{C}$*, "a/b", estará calculando apenas o quociente da divisão entre os dois inteiros. Assim*

$$3/4 \rightarrow 0$$

porque o quociente na divisão de 3 *por* 4 *é o inteiro 0. Nesta divisão o resto é ignorado. No* `gcc` *há funções para recuperar o resto na divisão inteira de modo que se possa escrever os dados do* algoritmo da divisão euclidiana

$$d_ividendo = d_ivisor * q_uociente + r_esto$$

O programa `numeros02.c`[7] *escreve os dados deste algoritmo.*

Há outras funções que executam tarefas semelhantes, mas com números reais. Leia mais a este respeito na próxima subseção sob o título, funções que analisam números reais.

[7] no diretorio do DOS, procurar `numer*.c`

7.1.2 Os números reais

> Esta seção se dedica aos números não inteiros. Vou estar me referindo, nesta seção, aos números reais e estarei assim super-adjetivando os números de que tratarei aqui, que nada mais são do que números racionais. Em inglês se usa a expressão `float` para se referir ao que chamaremos de reais. Em Português existe a expressão "número com ponto flutuante" que corresponde ao `float` americano. Vou adotar "real" por ser mais curto, mas quero dizer mesmo "*número com ponto flutuante*".
>
> Como eu já disse antes, em $\mathcal{C}$, os números formam um conjunto finito, e êste é também o caso dos números reais. Existe uma "aritmética" apropriada para estes números contida num padrão elaborado e divulgado pela `IEEE`. É bom lembrar que poderiamos escrever 100 páginas sobre este assunto...você tem aqui um resumo.

Há muitos anos atrás havia nas *tabernas* e sobretudo nas lojas de *material de construção*, umas máquinas mecânicas próprias para o cálculo com números reais[8]. As tais máquinas tinham duas teclas com setas para esquerda ou para direita, ao lado do teclado, que permitiam deslizar a escala para direita ou para a esquerda, era o ponto flutuando, e correspondia a multiplicar por 10 ou dividir por 10. Veja na figura as teclas vermelha, à esquerda, com as setas. Apertando as chaves laterais com o polegar e o apontador se limpava a "memória"... e rodando a manivela, à direita, se fazia a multiplicação (soma repetida) ou divisão (subtração repetida) conforme o sentido da rotação. Uma máquina mecânica de multiplicar. (fig. 7.1).

Figura 7.1: Máquina do balcão do comércio, coleção do autor.

Quando você escrever

$$173737823.$$

[8]ponto flutuante...

observe que o ponto não foi um engano, `gcc` irá entender que não se trata de um inteiro e fará diferença entre

$$173737823., 17373782.3, 173737.823, 17373.7823$$

entretanto

$$173737823. = 173737823.0 = 173737823.00$$

Observe que o ponto "flutuou", era o que a máquina mecânica, de que falamos acima, fazia.

O `gcc` usa os seguintes valores para definir a fronteira do *conjunto finito* de números reais:

1. reais com precisão simples
   (a) FLT_MIN 1.17549435E-38F
   (b) FLT_MAX 3.40282347E+38F
   (c) FLT_EPSILON 1.19209290E-07F

   quer dizer com 9 algarismos significativos.

2. reais com precisão dupla
   (a) DBL_MAX 1.7976931348623157E+308
   (b) DBL_MIN 2.2250738585072014E-308
   (c) DBL_EPSILON 2.2204460492503131E-016 portanto com 16 algarismos significativos.

para que você possa ter uma idéia do que esta precisão pode significar, analise o seguinte exemplo, sem levá-lo muito a sério...

Exemplo: 10 *Erro no acesso à estação espacial*

Por favor, não leve a sério este exemplo. Dê-lhe a importância indicada pelo tamanho da letra. Um problema deste tipo se resolve com instrumentos bem mais avançados. Comentários ao final.

Suponha que um programa monitorando o envio de uma nave espacial calcule o ponto de encontro da mesma com à estação espacial internacional *que se encontra em órbita terreste num raio de 326 Km (ou 326.000 m)*[9] *a partir do centro da Terra.*

Simplificando o processo de condução da nave, como se nada mais houvesse entre seu lançamento e sua chegada à estação espacial internacional*, e seu o ponto de encontro estivesse no centro da estação que deverá medir cerca de 120 metros quando estiver toda montada no ano 2005*[10]*, vamos calcular o erro relativamente ao ponto de encontro com a precisão que temos no* `gcc`*. Nossa simplificação vai ao ponto de supor que tudo se encontra estático e portanto que a nave parte em linha reta de encontro à* estação *se dirigindo ao ponto central. Quer dizer que estamos calculando a base de um triângulo isósceles cuja altura seria 326Km. A base deste triângulo é região de erro, a nave deveria chegar ao ponto central onde se encontraria o acesso. Com erro de* $FLT_EPSILON = 1.19209290E-07$ *teriamos* $erro = FLT_EPSILON * 326000 = .03886222854\ m$ *isto é, 3cm de erro.*

Obviamente nada neste exemplo é real, a não ser aproximadamente as dimensões,

[9] os valores não são precisos, o raio terrestre não é examente 6 km, por exemplo

[10] grato ao Prof. Emerson do Dep. de Física da UeVA por estas informações, ver http:www.nasa.gov

- *Não se enviam espaçonaves numa rota perpendicular à superficie da terra, apenas o lançamento é feito numa perpendicular para diminuir o gasto de energia com a saída da gravidade;*
- *As naves seguem rotas em forma de espiral para permitir a entrada em orbita perto do ponto de interesse e para melhor aproveitar a rotação da terra e a força de gravidade dos planetas, da lua ou do sol, por exemplo;*
- *A rota em espiral permite uma aproximação tangencial da órbita da estação espacial;*
- *As naves possuem um controle feito por computador que corrige a rota a cada ciclo do computador, quer dizer a cada milhonésimo de segundo;*
- *O instrumento matemático usado neste tipo de problemas são as* equações diferenciais ordinárias vetoriais *e não* semelhança de triângulos *...*

O exemplo serve apenas para mostrar que o erro $FLT_EPSILON = 1.19209290E-07$ *entre* dois números reais consecutivos *é suficientemente pequeno para resolver um problema da magnitude deste, envio de uma espaçonave para se acoplar com outra, com uma aproximação aceitável. Muito mais exatos ficam os resultados com a precisão dupla.*

Entretanto, para salvar o exemplo, as aproximações feitas a cada milhonésimo de segundo, podem usar "semelhança de trinângulos" e neste caso as alturas dos triângulos serão tão pequenas que a precisão final será de milhonésimos de centímetro (e não os 3cm calculados acima).

Finalmente, para resolver problemas críticos, *computadores são especialmente planejados com quantidade grande de memória e neste momento, sim, a precisão da linguagem C pode ser alterada, se for necessário. Mas, muito mais provável é que se construa uma linguagem apropriada para o problema, veja o que já dissemos na introdução. Leia também a observação em* `real01.c`.

Os programas `realXX.c` são um complemento do presente texto. Eles vão ser objeto dos exercícios seguintes.

Exercícios: 34 *Experiências com os reais*

1. *Leia e rode* `real.c`. *Veja os comentários do compilador e procure entender quais são os erros encontrados. Há comentários sobre os erros no programa corrigido* `real01.c`.

2. *Altere o programa* `real01.c` *para com ele fazer cálculo de áreas de triângulos e quadrados. Corrija os erros em* `real01.c`, *leia os comentários do compilador.*

   Solução: `real04.c`

3. *O programa* `real04.c` *tem diversos defeitos na saída de dados, rode-o, observe os defeitos e corrija o programa. Solução:* `real05.c`

4. *Modularize* `real04.c` *de modo que o usuário possa calcular áreas de triângulos, retângulos ou círculos separadamente. Faça observação sobre a área de quadrados semelhante a que se faz sobre círculos. Sugestão: reutilize o programa* `menu.c`. *Veja também o programa* `vazio.c`.

   Solução: `real06.c`

5. *Corrija os diversos defeitos de lay-out das saídas de dados do programa* `real06.c`.

6. *Crie uma biblioteca, "aritmetica.h", nela coloque as funções de* `real06.c`.

   <u>*Solução:*</u> `real07.c, aritmetica.h`

7. *Deixe o programa* `real07` *no tamanho da tela.*

   <u>*Solução:*</u> `real08.c, aritmetica.h`

8. *O programa* `real08.c` *tem ainda alguns defeitos de apresentação, corrija-os e expanda o programa para que ele execute mais operações. Tente manter a função principal* toda *visível na tela. Saída, crie uma função* `executa()` *na biblioteca* `aritmetica.h` *chmada de* `real09.c`*... e elimine o* `switch` *de* `real08.c`*, claro,* esconda*, se você quiser pensar assim, o* `switch` *em* `aritmetica.h`

9. *Faça um programa, chamado* `Matematica` *que execute algumas operações geométricas e aritméticas. Depois, distribua seu programa na rede, ele pode ser um tutorial.*

Não se pode dizer que fizemos muita matemática nos programas acima, de fato não. Entretanto você pode ver como se podem construir programas mais complicados a partir de programas mais simples.

A suite de programas `realXX.c` foi construida na sequência definida pelos exercícios do bloco anterior. Se você analisar com atenção irá encontrar aqui uma resposta para aquela pergunta que ficou no ar desde o início:

Observação: 28 *Existe alguma técnica para programar bem?*

Uma resposta simples para esta pergunta não existe. Conseguir responder a esta pergunta equivaleria a criar uma receita simples para resolver qualquer problema... obviamente isto é um absurdo.

Entretanto nós lhe mostramos aqui um método que ajuda a começar:

1. *Quebre o problema em pedacinhos e resolva cada um destes pedacinhos;*

2. *Não aceite que um programa fique maior do que a tela do computador... Um programa que ficar todo na tela é fácil de ser analisado. Os insetos se escondem facilmente em programas que ocupam várias telas ou milhões de linhas.*

3. *Se você estiver precisando de técnicas para detetização (debug), certamente seus programas são muito grandes. Quebre-os. Refaça tudo a partir do começo.*

4. *Aprenda a programar de forma simples e refaça os seus programas. Adianta pouco corrigir programas quilométricos...*

5. *Crie funções pequenas que resolvam tarefas interessantes e pequenas. Veja* `menu.c`*,* `vazio.c`*,* `aritmetica.h` *e* `ambiente.h`.

Funções de variável real

Vamos terminar esta seção analisando algumas funções da biblioteca[11] `glib`.Você vai encontrar mais informações consultando `info` do `LinuX`. Use os comandos

- Numa área de trabalho, digite `info`
- Dentro do `info` digite "m" e responda `glib` ou `libc`.

Vamos nos fixar nas funções que usaremos mais a frente e para isto colocaremos aqui indexação para facilitar sua busca de informações imediata. Mas aprenda a consultar `info`, vale a pena.

As funções que escolhemos para descrever aqui, executam tarefas pouco usuais mas de grande importância na solução de diversos problemas. Elas fazem o truncamento de números ou situam um determinado número entre dois inteiros. Fazem também a conversão de `real` em `inteiro`.

Elas estão definidas na biblioteca `math.h`.

Apresentamos as funções com sua sintaxe explicitada, por exemplo, a primeira é

double ceil(double X)

apresentada no formato como apareceria se estivessemos declarando esta função dentro de um programa. É assim que você irá encontrar a informação em `libc`.

- A função: double ceil (double X) calcula o inteiro mais próximo que seja maior do que X. Assim `ceil(X)` será um número do tipo real, mas inteiro, e maior ou igual que X.

  Quer dizer que

  - `ceil(X)` $\geq$ X;
  - `ceil(X)` é um número inteiro do tipo `real`.
  - Por exemplo

  $$ceil(1.5) = 2.0$$

  A palavra "ceil" vem de "ceiling", (teto). Quer dizer que estamos calculando o "teto inteiro de 1.5".

- A função:

  `double floor (double X)`

  calcula o inteiro mais próximo, e abaixo de X:

  - `floor`$(1.5) = 1.0$
  - `floor`$(x) \leq x$;
  - `floor`(x) é um inteiro, do tipo real.

[11] Mais abaixo vamos lhe mostrar como você pode pesquisar as bibliotecas disponibilizadas pelo `BC`.

A palavra “floor” é a palavra inglesa para (piso). 1.0 é o número inteiro (real) que está *logo abaixo* de 1.5. Escrevemos: “número inteiro (real)” porque o resultado de `floor()` é um número real. Isto é importante, ao calcular

$$\text{como o resultado é um real: } floor(3.5)/4 \rightarrow 0.75 \quad (7.1)$$

$$\text{se o resultado fosse inteiro: } floor(3.5)/4 \rightarrow 0 \quad (7.2)$$

São detalhes que esquecemos quando temos que fazer contas...e que se perdem em programas com centenas de linhas.

- A função:

  `double rint (double X)`

  arredonda X para um inteiro *guardando o tipo 'double'*, de acordo com o método de arredondamento definido para $\mathcal{C}$. Veja em `glib` com *info*. 'Floating Point Parameters' os vários tipos de arredondamento existentes. O método *default*[12], entretanto, é “`para o mais próximo inteiro`”. Assim, se no $\mathcal{C}$ que você estiver usando, estiver preservada a especificação de fábrica, `rint(X)` será o inteiro mais próximo de `X`. Será a alternativa otimizada de escolha entre `ceil(x), floor(x)`.

- A função: double modf (double VALOR, double *PARTE-INTEIRA) Esta função quebra o argumento VALOR em duas partes, A, B tal que:

  - $A \in [-1, 1]$;
  - B é inteiro mais próximo de zero;
  - A, B tem o mesmo sinal de VALOR.
  - Guarda o inteiro B em *PARTE-INTEIRA.
  - Devolve A na linha de comando[13].

  Por exemplo, ‘modf (-2.8, &parte_inteira)’ devolve ‘-0.8’ e guarda ‘-2.0’ em ‘parte_inteira’, dois números negativos porque $VALOR = -2.8 < 0$.

É preciso terminar dizendo, estamos longe de ter esgotado as funções definidas em `math.c`. Consulte `libc` com auxílio de `info` para se dar contas disto. Dentro de `info` use o comando “m” e digite `glib` ou `libc`.

7.1.3 Bibliotecas do BC

É através do `help` que você vai descobrir e analisar as biliotecas disponíveis com o `BC`. Abra um programa qualquer e coloque o cursor sobre `floor` e clique no botão `help`. Ao cair o menu, escolha `Topic search` e você vai cair num help sobre a função `floor()`.

[12] a palavra “default” significa “padrão”

[13] Como $\mathcal{C}$ não é uma linguagem interpretada, não tem sentido falar em “valores na linha de comando”... e sim, simplesmente, devolve A.

Observe no canto direito superior `MATH.H`, possivelmente em amarelo. É um indicativo de que esta função está na biblioteca `math.h`. Se você clicar em `MATH.H` o `help` vai levá-lo para ver as funções desta biblioteca. Tudo que dissemos acima sobre `glib` vale aqui para `help` do `BC`

Siga tela abaixo e você vai encontrar um programa-exemplo ilustrando como funciona esta função.

Em `math.h` você vai encontrar as outras funções cuja referência fizemos acima. Aproveite para passar os olhos sobre os nomes das funções e você vai encontrar `ceil()` entre muitas outras. Desça até o final da página e você irá encontrar `List of all header files`.

`Header file` é o que chamo biblioteca. O texto está enfatizado, coloque o cursor sobre ele e dê enter. Você vai encontrar a mesma listagem que existe sob `Linux`. Escolha alguma dessas bibliotecas para fazer uma analise rápida, e pronto, você já sabe onde pode encontrar as informações. Saia e volte quando precisar.

Cast - transformando tipos de dados

Há uma operação em C chamada `cast` que serve para transformação entre dois tipos de dados (de um "maior" para um "menor").

Por exemplo, você pode transformar um número do tipo `real` em outro do tipo `inteiro` "jogando" o `real` "dentro" do `inteiro` e naturalmente perdendo alguma informação: a parte fracionária.

Mas também você pode "jogar" inteiros nos reais, com `rint()`, ver `cast.c`).

Exercícios: 35 *float, ceil(), floor()*

1. *Rode e leia o programa* `floor.c`*.*
2. *Altere* `floor.c` *para calcular* `rint()`*.*

Veja o seguinte exemplo que é bem comum.

Exemplo: 11 *Transformando dados*

Considere um programa que calcule percentuais de dados inteiros, é o caso de uma pesquisa de opnião.

Os dados que vão ser usados são

- *Total de pessoal entrevistadas, um inteiro;*
- *número de pessoas que adota uma certa opinião, outro inteiro*

entretanto vamos querer calcular

$$\frac{opinião}{total}$$

e a conta vai ficar errada, porque, você já viu, no início deste capítulo, que

$$3/4 = 0$$

em C.

A saída é escrever

$$((float)3)/4 = 0.75$$

A expressão, `((tipo) var)` *se chama*

- *em, C,* “`cast`”,
- *em português,* `transformação do tipo de dados`.

Observe que basta fazer $((float)3)$ *porque se um dos “fatores” for tipo* `float` *o resultado será deste tipo.*

Utilidade desta operação? Inteiros ocupam menos espaço de memória que os reais! Como o programa vai receber dados inteiros, é melhor definir as variáveis que vão receber estes dados como `inteiras` ganhando espaço na memória e *tempo de processamento*! No cálculo final se faz a `transformação do tipo de dados`, para conseguir o resultado correto.

Observe que se `var` for do tipo `float` então

$$((int)var) \equiv floor(var)$$

Rode o programa `cast.c` para ver as limitações da transformação de dados e a perda de informações que ela pode acarretar. O seu uso fica restrito a número pequenos.

O programa `cast.c` vai lhe mostrar que a equivalência acima se torna discutível quando os números forem grandes.

7.2 Caracteres e vetores de caracteres.

Os caracteres

A palavra chave da linguagem C, para `caracteres` é `char`. Veja o programa[14]

`quatro_operacoes02.c`

onde a variável `operador` está definida como

`char operador;`

e deve receber um dos seguintes valores

$$+, *, /, -$$

Veja que o método, internamente (dentro do programa), consiste em escrever

$$'+', '*', '/', '-'$$

e não como escrevemos acima, nem

$$"+", "*", "/", -".$$

[14]no diretório do DOS, procure qua_opr*.c

Há uma diferença substâncial entre

$$"+", '+', +$$

- a é um símbolo que pode (ou não) ter um valor associado. Então + é um símbolo associado ao qual se encontra o algoritmo da adição;
- 'a' é um caractere, é um dos 256 caracteres reconhecidos no teclado.
- "a" é um vetor de caracteres, neste caso um vetor de tamanho 1.

Exercícios: 36 *Diferença - caracteres-strings*

1. *Altere o programa* `quatro_operacoes02.c` *usando "x" em vez de 'x' em cada ocorrência dos elementos de*

$$\{'+', '*', '/', '-'\}$$

e analise o resultado.

2. *Altere o programa*[15] `quatro_operacoes02.c` *usando x em vez de 'x' em cada ocorrência dos elementos de*

$$\{'+', '*', '/', '-'\}$$

e analise o resultado.

O resultado da experiência no primeiro exercício foi a seguinte:
warning: comparison between pointer and integer
e a justificativa é:

- "x" é um vetor (ponteiro...), é a diferença anunciada acima:
  "x" é um vetor de caracteres
  'x' é um caractere

  e os vetores são *naturalmente* `ponteiros` em $\mathcal{C}$.
- A variável `operador` foi definida para receber um caractere;
- Caracteres, junto com os números são os dados básicos, (leia: os valores básicos) da linguagem $\mathcal{C}$
- Vetor é uma variável indexada e isto se faz com (ponteiro) em $\mathcal{C}$.

Vamos aprofundar mais esta questão na seção sobre `ponteiro`, que é a próxima, e você, sem nenhum preconceito, pode lê-la agora e depois retornar a esta. Mas faça uma leitura rápida porque a presente discussão é fundamental.

[15] no diretório do DOS, procure `qua_opr*.c`

Observação: 29 *Valores básicos*

Vamos rapidamente insistir na expressão valores básicos *da linguagem. Digamos que* $\mathcal{C}$ *foi projetada para lidar com números e caracteres. Seria pouco dizer isto, ela foi projetada para trabalhar com os símbolos que você pode encontrar no teclado de sua máquina, como por exemplo *, 2,^.*

Seu objetivo seria criar outras linguagens, (originalmente um sistema operacional) que soubesse lidar com estes objetos, os `caracteres`*, de modo a criar expressões que tivessem signficado tanto para a máquina como para o ser humano que a fosse manipular.*

Uma linguagem de baixo nível, se dizia, (tem gente que ainda diz...)

Hoje a linguagem $\mathcal{C}$ *se projetou além deste objetivo estrito e podemos com ela fazer outros tipos de trabalho, (de alto nível...).*

A solução mais prática, para manter a linguagem com sua especificação inicial foi a de criar os vetores para entender "aglomerados de caracteres" como este:

A = "aglomerados de caracteres".

então A é um vetor, quer dizer uma variável formada de uma sucessão de "caracteres", numerados sequencialmente. Voltaremos logo abaixo a esta questão.

O segundo exercício no bloco acima produziu outra reclamação por parte do `gcc`. Agora o compilador se perdeu totalmente... a lista de reclamações passou de uma página porque o erro (*operador* $== *$) confundiu o resto da análise. $*$ agora é o nome de uma função que tem uma sintaxe particular: $a * b$, deve estar entre dois caracteres que `gcc` possa identificar como "números".

A sintaxe é algo extremamente importante para as linguagens de programação. A precisão tem que ser total. Cada símbolo tem que estar colocado exatamente nas condições especificadas. '*' é diferente de $*$, também

$$'1' \neq 1 \ ; \ '2' \neq 2.$$

Temos assim dois grandes tipos de dados em $\mathcal{C}$ com os quais podemos construir <u>todos</u> os outros:

- caracteres: '1', '2', . . . , 'a', 'b', . . . ;
- números que são formados a partir dos caracteres especiais 1,2,..,9,0

Vamos passar a discutir logo os `vetores de caracteres` e ao final faremos comparações e exercícios que terminarão por completar as ideias.

Caracteres especiais

Há vários caracteres especiais, a lista de exercícios seguinte é um tutorial sobre *caracteres.*

Exercícios: 37

1. *Leia, rode e leia o programa* `ascii.c` *e, claro, você nada viu. Leia os comentários no programa.*
2. *Leia, rode e leia agora o programa* `ascii_1.c`*. Este programa imprime um trecho da tabela* `ASCII`*, a partir de um ponto inicial que lhe vai ser solicitado.*
3. *Em* `ascii_1.c` *responda inicialmente com o número 1 quando isto lhe for pedido, e veja que nada será impresso. Veja a partir de quando alguma coisa "útil" é impressa respondendo ao programa outros pontos iniciais.*

4. *O programa* `ascii_2.c` *é uma pequena variante de* `ascii_1.c`. *Veja qual a diferença.*

5. *Leia, rode e leia* `ascii_3.c`. *Este programa usa uma variável,* `pausa` *para dar saltos de páginas formadas de 60 elementos da tabela* `ASCII`. *Mas verifique que a impressão fica mal feita em alguns casos, descubra por que.*

6. *Altere* `ascii_3.c` *para que cada página contenha 80 elementos da tabela.*
   Solução: `ascii_4.c`

7. *Veja em qualquer dos programas* `asciiX.c` *como imprimir caracteres, usando o formatador %c Faça um programa para imprimir*
   ```
   printf(``%c'',7);
   ```
   o ascii 7 que aciona a campinha.
   Solução: `apeteco2()` *em* `ambiente.h`

ASCII é uma sigla que significa *American Standard for Communication and Information Interchange* e foi criado, como o nome o indica, para criar um padrão para comunicações, possivelmente caminhando para se tornar obsoleto, ou de uso restrito ao núcleo interno do processamento de dados (como a definição dos códigos de teclados).

Os primeiros códigos ASCII são "especiais" servem para passar página, linhas, controlam a campainha do sistema, etc... não sendo porisso "visíveis" (possíveis de serem impressos).

Vetores de caracteres

A palavra `string` significa um objeto do tipo

```
``asdfafeqerr rere qerqe weqrqrerer''
```

um conjunto de caracteres, o que pode incluir espaço em branco, enfeixados por *aspas duplas*. Em $\mathcal{C}$ este *objeto* é um `vetor de caracteres`, quer dizer, uma matriz com uma linha e várias colunas.

O primeiro elemento da matriz, 'a' é indexado por zero e assim sucessivamente os demais, por 1,2,etc...

Se dermos um nome ao vetor

```
frase = ``asdfafeqerr rere qerqe weqrqrerer''
```

(e observe que a sintaxe acima não será aceita por $\mathcal{C}$ a não ser na inicialização e definição da variável) então

$$frase[0] =' a', frase[1] =' s', ...$$

A forma de fazer atribuição a um vetor de caracteres, fora da área de inicialização, é usando a função `strcpy()`. A igualdade acima seria:

```
strcpy(frase,``asdfafeqerr rere qerqe weqrqrerer'')
```

depois do que, em algum ponto do ponto posterior do programa, $frase[2]$ faz referência ao caracter 'd'.

Um caracter especial, o nulo, (NULL), '\0', fecha um vetor de caracteres. Assim, no exemplo acima, temos:

`frase[31]='e', frase[32]='r', frase[33]='\0'`

entretanto, na declaração de variáveis se pode ter:

`char frase[80]`

gerando as seguintes respostas:

`sizeof(frase) -> 80; strlen(frase) -> 32`

Veja mais a respeito de `vetores de caracteres` e as funções que os manipulam em `info`, use o comando `m` dando-lhe como resposta `libc`. Você vai encontrar, no índice `Character Handling`, a lista das funções para manipular `strings`.

Exercícios: 38 *Caracteres e vetores de caracteres*

1. *Rode (não leia...) o programa* `carac01.c`. *Ele explica* `caract1.c`. *Acompanhe com* `caract1.c` *aberto em uma janela.*
2. *Leia e rode o programa* `caract1.c` . *Altere* `caract1.c` *para fazer uma busca de um caractere qualquer fornecido pelo teclado.*
3. *Rode o programa* `caract7.c` *para ver o tamanho do espaço ocupado, na memória, por um caractere. Leia o programa também.*
4. *Leia e rode o programa* `caract11.c` , *verifique que o programa não se explica. Mofique-o para que o programa diga ao usuário o que vai ser feito.*
5. *Leia e rode o programa* `caract12.c` , *novamente este programa é executado sem grandes explicações. Inclua as mensagens necessárias.*
6. *Em* `caract12.c` *Troque o valor de* `busca` *para verificar segunda possibilidade do programa.*
7. *O programa* `caract13.c` *tem um erro, mesmo assim merece ser estudado. Leia o programa e descubra o erro.*

   Solução `caract14.c`

7.3 Ponteiros.

Ponteiro é um "super tipo"de variável em $\mathcal{C}$ no sentido de que é um tipo de variável de qualquer outro tipo... tem ponteiro do tipo inteiro, tem ponteiro do tipo real (float) etc...
Mas claro, esta não é a melhor forma de iniciar a discutir este assunto.
As *variáveis do tipo ponteiro* guardam endereços na memória que serão associados às outra variáveis. Os exemplos a seguir vão deixar bem claro o que é isto. O importante nesta introdução, é desmistificar (e ao mesmo tempo mostrar a importância), deste tipo de dados.
É dizer que `ponteiro` *aponta* para um endereço inicial de memória para guardar um tipo de dado: inteiro, float, etc...

Você pode criar uma variável to tipo ponteiro com a seguinte declaração:

```
tipo  outro\_nome {\tt nome}  *{\tt nome}ptr;  // uma variavel do tipo ponteiro.
```

Com esta declaração, você

- `nome`prt Criou uma variável do `tipo ponteiro` para guardar o endereço de um *determinado tipo de variável*, por exemplo `int`. O nome da variável é arbitrário, os programadores tem o hábito de acrecentar "ptr"ao final do nome para facilitar a identificação, no programa, das variáveis do tipo ponteiro.
- criou uma variável "`outro_nome`"do mesmo tipo que a variável do tipo ponteiro. Não é necessário fazer isto, mas com frequência é conveniente, inclusve assim (insistindo):
  `tipo outro_nome, nomeptr;`
- separou na memória espaço suficiente e necessário para associar com a variável `nome`. Por exemplo, se `nome` for do tipo `int` haverá 4 bytes separados a partir de um ponto inicial de memória;
- em algum momento, no código do programa, você deverá incluir o seguinte comando:

  ```
  nomeptr  = &nome;
  ```

  que fará a *associação* entre `nome` e os endereços reservados por `nomeptr`.

Exemplo: 12 *Um tutorial sobre ponteiros*
Os programas

`pont.c, pont1.c,...pont16.c`

representam alguns tutoriais sobre ponteiros. Vamos apresentar alguns deles agora.

1. *Primeira etapa.*

   Leia e rode os programas

   `pont.c, pont1.c, pont2.c`

   nesta ordem. Se possível abra duas áreas de trabalho (shells). Numa rode o programa `pontXX.c` *e na outra edite o programa para que você possa acompanhar o que ele está fazendo.*

2. *Segunda etapa.*

   Vamos comentar o que você viu. Se alguma coisa do presente comentário lhe parecer confuso, *repita a primeira etapa e retorne ao ponto em que a explicação lhe pareceu obscura.*

   A discussão está baseada em cada um dos programas.

- `pont.c` *Criamos duas variáveis de tipo* `int`*. Uma delas é um* ponteiro para um inteiro, `numptr`*. Veja a forma como se declaram ponteiros:*

  *int *numptr;*

  Inicialmente a variável `num` *nada tinha o que ver com* `numptr`*. Ao executar*

  *num = *numptr*

  foi estabelecida uma ligação. Experimente alterar a ordem dos dois comandos:

  *num=*numptr; num=6;*

  e você vai ver que o resultado é o mesmo. Mas o comando

  `num=*numptr;`

  é o que estabelece a ligação entre as duas variáveis. A partir de sua execução, alterações na variável `num` *serão registrada por* `*numptr`*.*

  A variável `numptr` *guarda um endereço de memória que pode ser associado a um inteiro, isto quer dizer, ela guarda o primeiro endereço de um segmento de memória que pode guardar uma variável do tipo inteiro: 4 bytes de memória.*

- `pont1.c` *O programa começa lhe pedindo um valor, mas não se deixe envolver...*

  Observe que o programa cometeu um erro: não indicou que tipo de dado espera, deveria ter dito: "me forneça um valor inteiro para a variável."

  Veja outra forma de associar variável e variável do tipo ponteiro (ou variável com endereço). O efeito é o mesmo que o obtido com a método de `pont.c`*, é apenas outro método.*

  Neste ponto você pode ver o uso dos operadores

  `*, &`

  para acessar

  - `*` *valor associado ao ponteiro;*
  - `&` *endereço do ponteiro.*

- `pont2.c` *Como toda outra variável, podemos declarar um ponteiro inicializado. Isto foi feito neste programa.*

  Você pode ver a sequência de números

  01234...01234

  indicando o índice de cada um dos valores, caracteres, dentro do vetor de caracteres apontado pelo ponteiro.

  Novamente vemos aqui o uso dos operadores

  `*, &`

  para acessar valor ou endereço.

  O programa lhe mostra o tamanho do vetor, e você pode ver uma caracteristica da linguagem C que toma como índice inicial o 0.

Depois o programa percorre um laço usando os índices do vetor de caracteres e imprimindo o valor de cada novo endereço e mostrando o endereço invariável, sempre mostrado, o endereço inicial do segmento de memória em que se encontra guardada o valor "isto é um teste".

O último valor de vetor é o NUL.

Exercícios: 39 *Laboratório com ponteiros Vamos trabalhar com os programas* `pontXX.c`

1. *Experimente imprimir com* `printf()`*, o nome de alguma função e analise o resultado. Por exemplo, altere a função* `main()`*, dentro do programa* `menu.c` *para que ela contenha apenas o comando:*

   printf(executa);

   Se der erro[16]*... complete os parâmetros de* `printf()` *com a formatação adequada!*

2. *Altere o programa* `pont1.c` *incluindo novas variáveis (com os respectivos ponteiros) para que você entenda como está utilizando "ponteiros".*

3. *Melhore* `pont1.c` *incluindo mensagem indicativa de que tipo de dado o programa espera. Não se esqueça de alterar o nome do programa, os "*programas errados*"têm uma função específica de laboratório e não devem ser alterados.*

4. *Altere* `pont1.c` *solicitando outros tipos de dados, como caracteres, número real, etc... Não se esqueça de alterar o nome do programa.*

5. *Leia, rode e leia o programa* `pont2.c`*. Acrescente novas variáveis, rode novamente o programa até que fique claro o uso dos* `operadores &, *`.

Observação: 30 *Identificadores de funções*
Os identificadores das funções são variáveis do tipo ponteiro.

Observação: 31 *Virtude e castigo.*

Se pode dizer que o uso de `ponteiros` *é tanto uma das* dificuldades *básicas da linguagem C, por um lado, e por outro lado, a sua principal* virtude.

Com a capacidade de acessar e manipular os dados diretamente na memória, a linguagem ganha uma rapidez típica da linguagem assembler que faz exclusivamente isto. Linguagens mais evoluidas *dentro da escala da abstração não permitem que o programador acesse diretamente a memória, elas fazem isto por eles. Com isto se ganha em segurança que falta ao C, (a segurança nos programas em C tem que ser conquistada centímetro a centímetro...). Melhor seria dizer, em vez de segurança,* abstração*, porque é possível programar em C com grande segurança*

- *deixando de usar ponteiros e perdendo assim uma das características da linguagem;*
- *aprendendo a dominar os ponteiros e fazendo registro (comentários) do seu uso quando eles se tornarem decisivos.*

---

[16]compre outro livro...

Como já dissemos anteriormente, se você declarar uma variável como inteiro, e nela guardar um número em formato de ponto flutuante, o risco mais elementar que o seu programa pode correr é que o espaço de nomes *reservado ao C pelo sistema operacional fique inútil por superposição de dados, e consequentemente o seu programa deixe de rodar. Um risco maior é que o setor de memória reservado ao C seja estourado e outros programas do sistema venham a ser corrompidos e consequentemente o computador fique* pendurado.

Portanto, todo cuidado é pouco no uso de ponteiros, e por outro lado, se perde uma grande possibilidade da linguagem se eliminarmos o uso dos ponteiros, ao programar em C.

Mas você pode, literalmente, programar em C sem usar ponteiros. Tem autores que sugerem fortemente esta atitude. Eu evito o uso de ponteiros, mas em geral não preciso deles para o meu trabalho.

Observe que numa mesma declaração é possível, com o uso da vírgula, declarar variáveis do tipo ponteiro e do *tipo comum*:

```
int *n,m ;
```

`n` é do tipo ponteiro apontando para um número inteiro, e `m` é do tipo inteiro, guardando o valor de um número inteiro. Mas se trata de um mau hábito, porque as duas declarações, a que está acima, e a que vem logo abaixo

```
int *nptr;
int m;
```

ocupam o mesmo espaço quando o programa for compilado e diferem de uma pequena quantidade *bytes*[17] quando guardadas em disco, entretanto a segunda é mais legível que a primeira e deve ser esta a forma de escrever-se um programa, *da forma mais legível*, para que os humanos consigam lê-los com mais facilidade, uma vez que, para as máquinas, as duas formas são idênticas.

Nós escrevemos programas para que outras pessoas, que trabalham conosco, na mesma equipe, possam ler com mais facilida e assim o trabalho em equipe possa fluir.

As seguintes declarações podem ser encontradas nos programas:

```
/* duas variaveis do tipo ponteiro para guardar enderecos
de memoria onde esta\~ao guardados numeros inteiros. */
int *n,*m;
/* variaveis de tipo inteiro. */
int l,p;
char palavra;
/* variavel de tipo ponteiro apontando para local de memoria onde
se encontra guardada uma variavel de tipo char */
char *frase;
/* variavel de tipo ponteiro apontando para local de memoria onde
se encontra guardada uma variavel de tipo ponto flutuante. */
float *vetor;
```

significando respectivamente que se criaram variáveis de tipos diversos algumas do tipo ponteiro, como está indicado nas observações.

[17]um byte é formado de 8 bits, e um bit é um zero ou um 1

7.3.1 Operações com ponteiros.

Crucial é o processo operatório com ponteiros. Vamos discutir isto em dois momentos:

- acesso às variáveis do tipo ponteiro
- operações aritméticas com ponteiros

Acesso aos ponteiros.

As operações de acesso aos ponteiros são:

- atribuir um valor ao endereço apontado. Isto pode ser feito de duas formas:
  1. Veja `pont1.c`
     ```
     numptr = &num;
     ```
     se houver algum valor atribuido a `num`, o que sempre há, então fica *indiretamente* atribuido um valor ao endereço `&numptr`
  2. Veja `pont.c`
     ```
     num = *numptr
     ```
     Em ambos os caso, mais do que *atribuir* um valor a um endereço o que está sendo feito é *associar* um endereço a uma variável. De agora em diante mudanças de valores em `num` são automaticamente associadas com o endereço `&numptr`.
- explicitar o endereço.

  É feito com o operador `&`.
- ir buscar um valor associado ao endereço apontado.

  É feito com o operador `*`

Operações aritméticas com ponteiros.

Por razões obvias, só há duas operações que se podem fazer com ponteiros: somar ou subtrair um número inteiro.

Ponteiro é endereço, e da mesma forma como não teria sentido somar endereços de casas, não tem sentido somar ponteiros, mas tem sentido somar um número inteiro[18] a um ponteiro, ou subtrair.

Como um endereço é um número inteiro, $\mathcal{C}$ permite somar dois ponteiros. Quando isto é feito, *na verdade* está sendo somado o espaço ocupado por uma variável a um deteminado endereço.

Também a única operação lógica natural com ponteiros é a comparação para determinar a igualdade ou uma desigualdade entre os valores para os quais eles apontam.

---

[18] como teria sentido somar ao endereço de uma casa 10 ou 20 metros para indicar onde começa a próxima casa, o mesmo se passa com os ponteiros e o tamanho do tipo de dado...

Exercícios: 40 *Ponteiros*

1. *Rode e leia* `pont3.c`*, e faça o que o programa pede, e volte a rodar o programa.*

2. *O programa* `pont4.c` *não pode ser compilado. Veja qual o erro e o corrija. Depois de corrigido, rode o programa. Não altere o programa, mude-lhe o nome.*

3. *Altere o valor da variável em* `pont4.c` *e volte a rodar o programa. Altere o valor da variável colocando nela uma frase que tenha sentido, volte a rodar o programa.*

4. *Acrescente a mudança de linha no comando de impressão de* `pont4.c`*, dentro do loop, e volte a rodar o programa.*

5. *Verifique porque* `pont5.c` *não roda, corrija-o e depois rode o programa. A solução se encontra no comentario do programa.*

6. *Traduza para o inglês todos os programas da suite* `pontXX.c`*.*

7.4 Manipulando arquivos em disco

Arquivos são um *objeto* ao qual estão associados cinco *métodos* mais importantes:

- `fopen() abrir`
- `fclose() fechar`
- `fclose(‘‘arquivo’’,’’r’’)` abrir para ler (read ’r’)
- `fclose(‘‘arquivo’’,’’w’’)` abrir para escrever (write ’w’)
- `fclose(‘‘arquivo’’,’’a’’)` abrir para acréscimos (append ’a’)
- `fprintf()` é a irmão de `printf()` para enviar dados para arquivo em disco.
- `fgets()` faz leitura vinda de um arquivo. Observe a mudança na sintaxe.

A função `fopen()` tem um papel especial aqui, veja os detalhes técnicos nos programas

- leitura com `fgets() prog20_X.c`
- escrita com `fprintf()` em `agend4.c`

Há duas formas de usar `fopen()`. Uma simples:

`fopen(nome_do_arquivo, ‘‘w|r|a’’)`

você deve escolher um dos métodos `w,r,a` e uma outra forma mais `algébrica` em que `nome_do_arquivo` é uma variável e contém o nome do arquivo.

Se habitue de escrever sempre a companheira de de `fopen()`, a função `fclose(nome_do_arquivo)` na linha de baixo, assim que usar `fopen()`, para evitar de esquecer.

Não fechar um arquivo pode deixá-lo corrompido.

A sequência de programas `prog20_X.c` mostra a evolução de tentativas para ler dados gravados num arquivo em disco, você deve rodá-los e ler os comentários que explicam porque os programas falharam. Sempre, o programa de maior índice é o programa "mais correto" da suite considerada.

Exercícios: 41 *Uso de arquivos em disco*

1. *O programa* `prog20.c` *está errado, ignore isto inicialmente.* Leia, rode e leia *o programa para entender como ele funciona. Leia os comentários e se precisar, faça pequenas modificações com o objetivo de entender o programa e fazê-lo funcionar (não se esqueça de trocar-lhe o nome).*

2. `prog20.c` *não diz quantas ocorrências foram encontradas da palavra escolhida. Altere isto para ficar sabendo quantas vezes aparece a palavra escolhida, é apenas um erro de portugues...e claro, o resultado errado está também neste ponto!*

   Solução `prog20_7.c`

3. *O programa* `prog20.c` *está fazendo uma estatística errada. Tente descobrir o erro.*

7.5 Matriz, (array)

> Tabela de dados.
> Há dois tipos de dados em $\mathcal{C}$ para tabular dados:
>
> 1. As **matrizes**, (**arrays**);
> 2. As **estruturas** (**structs**).
>
> Neste parágrafo vamos estudar as matrizes, e no seguinte as estruturas.

As matrizes são uma das invenções mais antigas da *matemática moderna*[19]. Veja um exemplo:

$$\mathcal{A} = \begin{bmatrix} 1 & 2 & 3 & -1 \\ 2 & -1 & 3 & 2 \\ 2 & -1 & 3 & 2 \end{bmatrix} \tag{7.3}$$

A matriz $\mathcal{A}$ tem 12 elementos distribuidos em tres linhas e quatro colunas. Cada linha tem quatro colunas, ou vice-versa cada coluna tem tres linhas.

Esta "indecisão" sob a forma de ver a distribuição dos elementos de $\mathcal{A}$ forçou uma convenção chamada "lico" (linha-coluna). Por esta convenção vamos ver uma matriz como formada por linhas e as linhas formadas por colunas. Desta forma o "endereçamento" dos elementos na matriz fica definido:

[19] Se você de fato exigir uma definição de "matemática moderna", que tal dizer que é aquela que estamos usando e construindo hoje...

$$\mathcal{A}_{3\ 4} = 2$$

enquanto que que $\mathcal{A}_{4\ 3}$ não existe *nesta matriz.* A notação acima é matemática. Em $\mathcal{C}$ diremos:

$$\mathcal{A}[3][4] = 2;$$

ou melhor, podemos[20] *atribuir* o valor 2 à posição 3, 4 da matriz $\mathcal{A}$.

$\mathcal{C}$ "entende" as matrizes de modo diferente do que fazem as demais *linguagens de programação.* Compare com os `vetores de caracteres` (`strings`) que também são matrizes.

As matrizes em $\mathcal{C}$ são "endereços", portanto `ponteiros`. A declaração de uma variável do tipo `matriz` se faz assim:

```
float A[30][20];
```

ou mais genericamente:

```
tipo_de_dado identificador[num_lin][num_col];
```

Veja no programa-errado `texto.c` a definição da variável `palavras`, observe como falamos: *a variável* "palavras". O identificador é "palavras" e não "`palavras[80][5]`".

Vamos insistir na observação que fizemos acima, de que as *matrizes em $\mathcal{C}$ são* `ponteiros` e corrigir o erro acima a respeito da atribuição.

Veja as duas linhas seguintes de código para fazer atribuições de valores às matrizes. Suponha que `palavras` represente uma matriz definida assim:

```
int palavras[10][5];
```

Vejamos agora como se fazem atribuições de dados nos elementos das matrizes:

- Forma errada de atribuir valor ao endereco $[i][j]$:

  ```
  fgets(deposito, sizeof(deposito), stdin);
  sscanf(deposito, "%s", &palavras[i][k]);
  ```

- Forma certa de atribuir valor ao endereco $[i][j]$:

  ```
  fgets(deposito, sizeof(deposito), stdin);
  sscanf(deposito, "%s", palavras[i][k]);
  ```

  sem o redirecionador de endereço porque

  ```
  palavras[i][k]
  ```

[20] falso..., veja logo abaixo como é que se fazem atribuições!

já é um endereço (ou um `ponteiro`).

Observação: 32 *Índices em $\mathcal{C}$*

Se fossemos definir índices terminariamos por escrever outro livro[21]*...*

Falando levianamente, digamos que índexação serve para estabelecer a correspondência entre elementos de dois conjuntos, (seria portanto uma espécie de função? resposta: sim). Mas em geral não queremos ver os índices desta forma e sim como uma espécie de "contagem"[22]*.*

Veja a matriz $\mathcal{A}$ da matemática. Dissemos acima que o elemento $3,4$ era o dois:

$$\mathcal{A}_{3\ 2} = 2.$$

Exatamente como os apartamentos de um prédio de 3 andares em que houvesse 4 apartamentos por andar, $3,4$ é o número de um apartamento. A vírgula define uma "sintaxe" separando o indicativo da linha *do* indicativo da coluna.

As matrizes são a estrutura de dados *das tabelas retângulares com mútiplas* entradas *em que todas as entradas são do mesmo tipo:*

1. *apartamentos;*
2. *números*

nos exemplos que demos acima. Na próxima seção veremos tabelas em que os elementos são de formato e tipo diferentes, as `estruturas`.

Vamos terminar esta observação falando de uma peculiaridade da linguagem $\mathcal{C}$. Os índices em $\mathcal{C}$ começam de zero. Quer dizer, quando definirmos

$$float A[4][7]$$

teremos os seguintes endereços disponíveis:

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a0,3 & a_{0,4} & a_{0,5} & a_{0,6} \\ a_{1,0} & a_{1,1} & a_{1,2} & a1,3 & a_{1,4} & a_{1,5} & a_{1,6} \\ a_{2,0} & a_{2,1} & a_{2,2} & a2,3 & a_{2,4} & a_{2,5} & a_{2,6} \end{bmatrix} \quad (7.4)$$

E o programador deve tormar cuidado para otimizar seu uso da memória.

Se não usar os índices a partir de zero, estará deixando memória desocupada, mas "reservada".

A declaração

```
int a[4][7];
```

separa na memória 28 espaços para números reais, portanto 28 x 4 *bytes* = 112 *bytes* vai ocupar a matriz `a` na memória do computador ou no disco.

Existe uma maneira de falar, resumida, que caracteriza o "tamanho" de uma matriz e a disposição de suas linhas e colunas: dizemos que <u>a</u> é matriz 4 x 7, que se lê "4 por 7".

Definição: 1 *Dimensão de uma matriz De forma mais geral, diremos que uma matriz <u>a</u> é n x m, ou "n por m" se ela tiver n linhas e m colunas. Isto é também o que se chama de "dimensão" de uma matriz.*

Exercícios: 42 *Matrizes em $\mathcal{C}$*

[21] que esta observação não o assuste, mas também fique certo de que índice é um assunto complexo

[22] ora, isto volta a ser função...

1. *Escreva um programa criando uma matriz* 3 x 4 *de inteiros. Comentário: você vai precisar de um* `while`.
2. *Quanto de memória ocupa uma matriz* 3 x 4 *de inteiros. Resp 48 bytes.*
3. *Quanto de memória ocupa uma matriz* 3 x 4 *de reais. Resp* 384 *bytes* = 12 x 32 *bytes.*
4. *Use um programa que avalie o uso da memória no computador que você estiver usando e veja qual é maior matriz de números reais que um programa poderia manipular nesta máquina. A resposta não é única, veja uma questão, nesta lista, sobre disquetes.*
5. *Quais são os tamanhos máximos de matriz que podemos guardar num disquete de 1.4kb ?*

   Solução: 1 *fatorando 1440, temos* $1440 = 2 \ \text{x} \ 2 \ \text{x} \ 2 \ \text{x} \ 2 \ \text{x} \ 2 \ \text{x} \ 3 \ \text{x} \ 3 \ \text{x} \ 5$ *que pode gerar os seguintes pares de fatores:*

   $$2 \ \text{x} \ 720; 4 \ \text{x} \ 360; 8 \ \text{x} \ 180; 16 \ \text{x} \ 90; 32 \ \text{x} \ 45$$
   $$96 \ \text{x} \ 15; 288 \ \text{x} \ 5; \ldots$$

   Isto é, todos os pares $x \ \text{x} \ y; x * y = 1440$.

6. *Usando matrizes para textos, que horror!*
   (a) *Analise o programa "errado"*`texto.c, rode-o.`
   (b) `O erro do programa texto.c se encontra demonstrado ao final do mesmo, no último laço. Analise o que está acontecendo e tente a correção.`
   `correção em texto01.c.`
   (c) `atribuição de valor Troque, em texto.c`

```
palavras[1]= "Marcar consulta com um especialista.";
palavras[2]= "Agendar algum exame de laboratório.";
palavras[3]= "Marcar uma consulta de retôrno.";
palavras[4]= "Bater papo à-toa.";
```

   `por`

```
strcpy(palavras[1], "Marcar consulta com um especialista.");
strcpy(palavras[2], "Agendar algum exame de laboratório.");
strcpy(palavras[3], "Marcar uma consulta de retôrno.");
strcpy(palavras[4], "Bater papo à-toa.");
```

   `e compile o programa. Procure assimilar a mensagem de erro que surge, ela lhe diz ''incompatible types in assignment'' significando que a atribuição (assignment) tenta associar tipos de dados`[23] `incompatíveis.`

[23]A mensagem é estúpida, não são os tipos de dados que são incompatíveis...

(d) `Observe que texto.c se compõe de tres etapas bem marcadas:`

  i. `diálogo com o usuário;`

  ii. `uma entrada de dados;`

  iii. `uma saída de dados.`

  `Marque estas etapas com comentários.`

  `solução em texto01.c.`

(e) `Transforme cada uma das etapas do programa texto01.c em uma função, modularizando o programa.`

  `solução em texto02.c.`

7.6 Estrutura, struct.

Dados estruturados, foi o grito de guerra da ciência da computação na década de 70. Claro, hoje continuamos aquela tarefa com um tipo de estruturação mais aprofundada, *orientação a objetos.* Vamos ver aqui como se estrutura a informação, em $\mathcal{C}$ e por que. Mas não chegaresmos a discutir *orientação a objetos*, que é típica de linguagens como $\mathcal{C}++$, `Python`, `Java`, entre outras menos populares.
Uma `estrutura, (struct)`, é uma construção, em $\mathcal{C}$ de um novo tipo de dados formado de campos onde dados de tipo diferentes são guardados. Dissemos *dados de tipos diferentes* porque, se os dados forem do mesmo tipo será melhor usar *vetor* em vez de uma `estrutura`.
Exemplos de `estruturas` são as tabelas dupla ou múltipla entrada, em que cada coluna (ou linha) guarda um tipo de dado.

O tipo de dado `struct, estrutura`, em $\mathcal{C}$ se inspira nas tabelas de dupla ou múltipla entrada, em que cada coluna (ou linha) guarda um tipo de dado:

entrev. \ dados	salário	idade	turno	profissão	especialidade	bairro
José	500,00	27	m,t	professor	literatura	Junco
Maria	600,00	31	t,n	professora	sociologia	Centro
João	1.000,00	45	m,n	professor	matemática	Centro
Francisco	800,00	35	t,n	médico	ginecologia	Junco

que poderia ser uma folha de censo com as respostas dadas por 4 entrevistados. Quer dizer que para este censo se considerou importante registrar os campos

salário, idade, turno, profissão, especialidade,bairro

como característica de cada entrevistado. Um "cidadão", para este censo, se caracteriza pelas propriedades acima. Não interessa aqui discutir se este censo está mal elaborado, e certamente está. Nosso objetivo é considerar um exemplo

de tabela e ver como é construida para exibir o instrumento da `linguagem C` que pode produzir tabelas.

Para que um *entrevistador* exerça sua função, ele deve ser informado sobre quais os valores aceitáveis para cada campo. Por exemplo, no campo "expediente" dois valores devem ser dados, tirados de

m, t, n

m de *manhã*, t de *tarde* ou n de *noite*.

Você encontra a definição e exemplo de `struct` dentro dos programas `estruturaX.c` no disco. No diretório do `BC`, `estruX.c`

Exercícios: 43 *tabelas e censos*

1. *Rode e leia o programa* `cadast.c`
2. *Altere o programa* `cadast.c` *para registrar os dados de uma turma de alunos, com os campos* `nome, nota1,nota2,nota3, média final`.
3. *Crie uma entrada de dados para o programa e uma saída de dados, se não tiver feito ainda.*

Da mesma forma temos, por exemplo, "tempo". Tempo é uma "estrutura"com os seguintes campos:

dia_da_semana(0..6) dia_do_ano(0..365) segundos (0..59) minutos(0..59) hora(0..23) dia_do_mes(1..31) mes(0..11) ano(1900 ...)

Ao lado de cada variável da `estrutura tempo`, e *apenas para informação do usuário*, se encontram os valores que foram planejados. Em geral não é conveniente criar as variáveis com tais restrições do ponto de vista de seus valores, embora isto seja possível e crie mais segurança para o programa (e mais trabalho para o programador que deve criar mecanismos de verificação para os dados fornecidos ou lidos automaticamente).

De maneira análoga poderiamos ter construido a tabela do censo, indicando com brevidade, usando códigos, as informações de cada campo. Para profissão, poderiamos ter usado a codificação da Receita Federal. Para salário poderiamos ter usado inteiros indicando múltiplos do "salário mínimo".

Feita uma "especificação" deste tipo é possível guardar a informação de forma compactada. Assim, no caso do tempo, o conjunto de dígitos

200007011331290346

representa um determinado instante na seqüência do tempo e poderia ser reescrito, em formato de fácil leitura, para humanos:

2000.07.01.13.31.29.03.46

ou usando outro qualquer tipo de separador como

2000/07/01 – 13.31.29 – 03.46

porque definimos uma estrutura para caracterizar o tempo e agora com um programa podemos facilmente passar da expressão apropriada para humanos para a expressão apropriada para cálculos algébricos no computador e vice-versa.

Uma rotina pode então ser construida para fazer uma *álgebra de tempo* para somar, subtrair tempos permitindo que um programa possa calcular prazos decorridos, ou detectar se a data para um deteminado evento chegou.

O `gcc` previu estas duas formas de tratar o tempo, *humano* e *interno para cálculo com as máquinas.* Cabe ao programador construir a tradução adequada.

Da mesma forma um programa de computador pode agir sobre o "censo" para dele extrair informações qualificadas sobre uma determinada região, como salário médio, concentração profissional, etc... através de uma "álgebra" adequada para tratar estes dados.

Observação: 33 *Codificação e leitura humana*

Codificação *é assunto para computadores. Programas devem ser escritos para uma fácil comunicação com humanos que lêm frases, e não códigos.*

As traduções, código $\Longleftrightarrow$ frases *podem ser feitas facilmente com funções em qualquer linguagem de programação, desde que as frases se restrinjam a um conjunto bem definido de palavras. Inclusive estas* frases *podem ficar guardadas em arquivos no computador e serem escolhidas a partir de pedaços digitados pelo usuário, como acontece nos módulos de pesquisa dos programas na Internet. Internamente os programas vão trabalhar com os* códigos.

Depois desta breve apresentação genérica sobre estrutura de dados[24]*, vamos nos especializar em alguns casos. Você pode encontrar um mundo de informações a respeito no* info*, no manual sobre* Libc*, veja no índice remissivo* info *ou* Libc.

Vamos adotar aqui uma terminologia para fazer referência aos dois formatos sob os quais o `gcc` processa o tempo.

- `tempo para leitura`, chamado em inglês de `broken time`, porque ele se apresenta em campos, como se encontra descrito a seguir;

- `tempo para cálculos`, que se apresenta em estado bruto, em segundos, a partir de um momento que foi definido como `epoch`, "a época".

É relativamente fácil passar de um para o outro entre estes dois tipos de sistemas de processamento de informações relativas ao *tempo*, no `ggc`. Se você

[24]há livros inteiros, com mais de 200 páginas sobre o assunto...

tiver tempo no formato bruto em segundos, `tempo para cálculos`, contando desde a `época`, basta sair sub-dividindo em aglomerados sucessivos de anos e o resto em meses, e assim assim sucessivamente até saber qual é a data que aquele número representa.

Por outro lado, um inteiro `bruto` é fácil de ser somado ou subtraido de outro permitindo assim uma *álgebra do tempo* simples. Veja um exemplo:

Exemplo: 13 *Tradução e álgebra de tempo*
Queremos saber qual o tempo que se passou entre
final; 10 de Dezembro de 1998
e
inicio; 15 de Outubro de 1994.

Traduzimos inicio e final para tempo bruto. Nos parece mais fácil se estabelecermos uma "`época`*" particular, por exemplo, 01 de Janeiro de 1994 e calcularmos o número de segundos contidos em 15 de Outubro de 1994:*

$$inicio = 24969600$$

e depois, relativamente á mesma época calcularmos o número de segundo contidos em 10 de Dezembro de 1998:

$$final = 156297600$$

A diferença em segundos é:

$$lapso = final - inicio = 131328000$$

que é o lapso de tempo decorrido em segundos, que agora vamos `quebrar`*, como dizem os americanos, em anos, meses, dias, minutos e segundos. Se for para o cálculo dos juros de uma dívida, iremos desprezar minutos e segundos.*

Para isto criamos novas variáveis, `ano, mes` *para conter o número de segundos em ano comercial e no mes comercial que tem 30 dias, porque esta é regra legal para o cálculo do tempo decorrido. Depois faremos divisões sucessivas:*

$$anos = lapso/ano = 4.164383561643835616 43$$

que diz se terem passado 4 anos, isto

$$4 * ano = 126144000$$

restando portanto

$$resto = lapso - 4 * ano = 5184000$$

e finalmente vamos ver quantos meses se passaram. Para isto definiremos a variavel mes:

$$mes = 30 * 24 * 60 * 60$$

e depois calculamos[25]

$$meses = resto/mes = 2$$

Sendo assim o tempo legal decorrido entre as duas datas de 4 anos e 2 meses.

Exercícios: 44 *Estrutura e álgebra com o tempo*

1. *Escreva uma função que, recebendo duas datas diferentes, calcule o lapso de tempo decorrido entre elas.*
2. *Melhore a função construida usando uma "`época`" definida no programa.*
3. *Faça uma função que calcule os juros devidos para uma certa 'soma' entre dois períodos dados.*
4. *Faça um programa para cadastrar os funcionários de uma empresa registrando os dados:*

   Nome, idade, altura, peso

   cada um destes dados numa nova linha no arquivo.

   Solução: `cadapesX.c`.

7.6.1 O tempo para os humanos lerem

- Para expressar o tempo em forma legível pelos humanos,
- Para fazer cálculos algébricos com o conceito de *tempo*

Neste seção vamos trabalhar com a escrita do tempo para os humanos lerem e deixar os cálculos do tempo para a próxima.

Você pode encontrar mais informações técnicas no manual do `gcc`, veja *Libc* no índice remissivo.

Tipo de dados: struct *tm

Este tipo de dados contém os seguintes campos:

- int tm_sec que representa o número de segundos e varia de 0..59.
- int tm_min Número de minutos até completar uma hora, variação 0..59.
- int tm_hour Número de horas a apartir de *meia noite*, variação 0..23.
- int tm_mday Número dos dias do mes, variação 1..31.
- int tm_mon Numeração dos meses do ano, variação 0..11.
- int tm_year Numeração dos anos a partir de 1900.

[25] observe que se trata de uma multiplicação para não escrever 2592000, que seria a quantidade segundos, porém, indecifrável...

- int tm_wday Numeração dos dias da semana, domingo=0 (Sunday). Variação 0..6
- int tm_yday Numeração dos dias do ano, a partir de 1ºde Janeiro, variando 0..365.
- int tm_isdst Uma etiqueta (flag) para indicar se está em vigor o horário de verão. Se estiver em vigor,
  1. tm_isdst=1, se estiver em vigor;
  2. tm_isdst=0, se não estiver em vigor;
  3. tm_isdst=< 0, se a informação não estiver disponível;
- long int tm_gmtoff

  É o número de segundos que mede o afastamento do horário local (algébrico) do horário-gmt. Por exemplo, deve ser adicionado -4*60*60 para se corrigir o horário de Brasilia relativamente ao GMT. Pertence a biblioteca GNU-C e nao existe em um ambiente ISO C.
- const char *tm_zone Nome da zona de tempo em uso, também inexistente no ISO C.
- Função:

  struct tm* localtime (const time_t *TIME)

  Na variável TIME se encontra o endereço onde está armazenado o *tempo*, a função 'localtime' converte o conteúdo de TIME em sua representação de `tempo para leitura`, relativamente a `zona de tempo` registrada na máquina.

  A função devolve um ponteiro para este valor do tempo que pode ser utilizado para recuperá-lo e fazer as transformações que se deseje, veja o programa `hora.c`. Veja mais detalhes em `Libc`, ver "info" no índice remissivo ao final do livro.

  Leia (ou releia) o exemplo 13, página 162. Veja também os programas da série `horaX.c` em que você pode encontrar dicas de como automatizar os cáculos feitos no exemplo citado.
- Função: struct tm * gmtime (const time_t *TIME)

  Semelhante à função localtime. A diferença é que o tempo se apresenta no formato UTC[26]

[26]esta sigla representa um acordo entre francófonos e anglófonos na divisão do mundo.... os franceses dizem Universel temps coordoné e os americanos dizem Universal Time Coordinated. Ambos aceitam perder um pouquinho da sintaxe nas respectivas línguas,fala-se que os ingleses queriam medir o tempo em *pés*.

- Função: time_t mktime (struct tm *BROKENTIME)

  A função mktime é usada para converter o *tempo fracionado* no *tempo do calendário*. Completa `tm_yday` , e `tm_wday` que estejam faltando a partir dos demais dados, num processo de noramalização. Se não for possível restaurar o tempo para o formato do calendário, o valor de retorno será -1. Esta função também dá valor a variável `tzname` - nome da zona de horário.

7.6.2 O tempo para o computador fazer álgebra

O exemplo 13, página 162 se constitue na motivação desta seção.

Você pode encontrar mais informações técnicas no manual do `gcc`, veja *Libc* no índice remissivo.

Exercícios: 45 *Estruturas*

1. *Verifique que o programa* `texto.c` *usa indevidamente a tipo de dados* `array`*, transforme-o usando* `struct` *que é tipo de dados natural para um tal programa.*

7.7 Formatadores para saída de dados

> Nesta seção final reunimos as informações sobre como formatar os dados basicamente para as funções `printf(), fprintf(), scanf(), sscanf()`.
> Vamos aqui aplicar o assunto *tipo de dados* em dois dos momentos mais críticos de um programa, na saída e na entrada de dados.
>
> Dominando o uso destas funções, vovê estará fazendo um tutorial prático sobre *tipos de dados*.

Nesta seção vamos estudar mais detalhadamente a sintaxe para a conversão da dados necessária para que

`printf(), fprintf(), scanf(), sscanf()`

leia ou escreva os dados de forma agradável, ou no caso de `scanf` consiga interpretar os dados que lhe forem oferecidos.

Apesar da posição deste capítulo no livro cabe aqui uma apresentação explicita destas funções, porque *você pode estar aqui de visita*, vindo das primeiras páginas do livro incentivado pela indexação do assunto.

Vocabulário: 7 `printf(), fprintf(), scanf(), sscanf()`

- `printf()` *Saída de dados. É a função que imprime dados na tela e que nós traduzimos por* `imprima()` *ou* `escreve()`*. É bastante complexa porque admite uma grande quantidade de parâmetros permitindo organizar de forma perfeita a saida de dados na tela ou em papel.*

- fprintf() *Saída dados. É a função que imprime dados em um arquivo,* fprintf.

  O primeiro "f" do nome quer dizer isto. É muito semelhante printf()*, mas com diferenças específicos, você vai ter que lhe dizer em que "arquivo" as* coisas *devem ser* impressas. *Também você vai ter que dizer de onde vêm as coisas. Veja nos programas* agend1.c *exemplos de uso desta função.*

- scanf() *Entrada de dados. É a função ultra-poderosa, ultra-simples e sensível para fazer entrada de dados. Tem uma sintaxe semelhante a de* printf()*. Enquanto você não tiver uma boa experiência evite de usar esta função.*

  Use sscanf() que é mais burocrática, porém mais segura.

- sscanf() *Entrada de dados. É a função muito poderosa para fazer entrada de dados. Tem a mesma capacidade de* scanf() *porém sua sintaxe conduz melhor o programador a evitar os erros. Veja no arquivo* entra_dados.c *exemplo de uso desta função junto com* fgets().

Vamos nos concentrar em printf uma vez que vale o que dissermos também para fprintf e, em alguma medida, também para scanf. Vamos discutir as diferenças.

A figura (fig. 7.2) página 166, mostra um exemplo e fixa a terminologia que vamos usar.

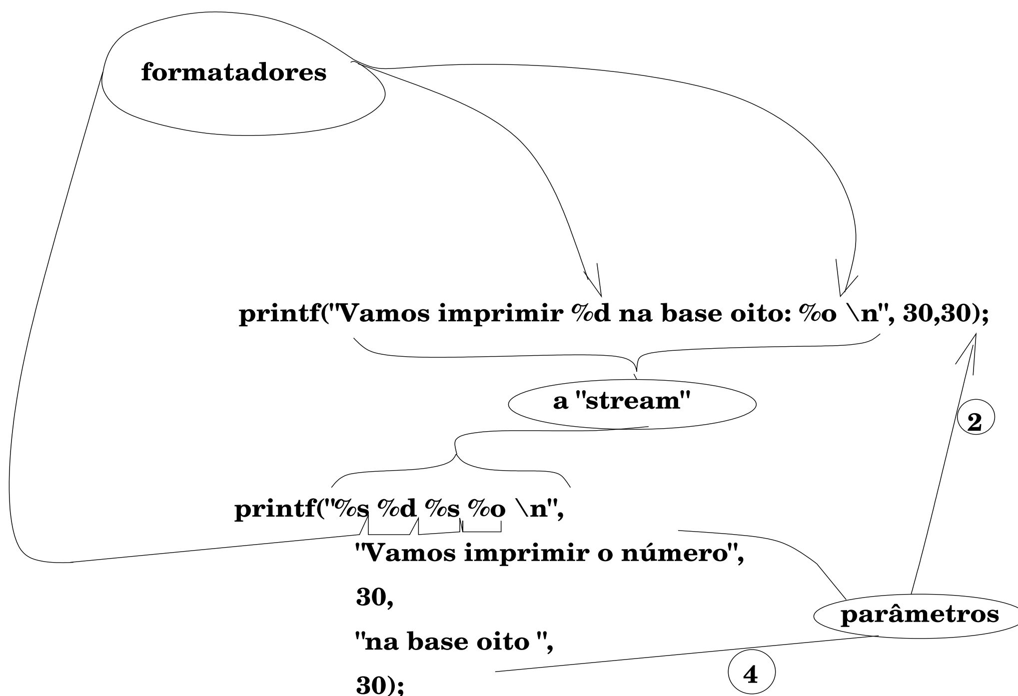

Figura 7.2: duas formas equivalentes para imprimir 30 na base 8

Dois métodos para formatadar os dados:

- Os formatadores de dados imersos num vetor de caracteres.

  O exemplo da (fig. 7.2) mostra que os `formatadores de dados` podem estar distribuidos (imersos) dentro de um vetor de caracteres oferecido a `printf` que então completará este vetor de caracteres "expandindo" os correspondentes formatadores de dados na ordem em que eles forem encontrados. A "`máscara` se chama em ingês, `stream` contém os formatadores sendo seguida por uma lista de parâmetros. Ver `numero01.c` como exemplo deste método.

- Os formatadores de dados numa mascara que tudo define. Uma outra forma de fazer, consiste em, escrever todos os formatadores de dados como um primeiro parâmetro, chamado *modelo*, em inglês `template`, e fornecer, separados por vírgulas, os correspondentes dados. Ver `prog20.c`, ao final, como um exemplo deste método. Leia inclusive a observação (20) ao final.

A formatação de dados da função `printf()` tem a seguinte estrutura:
`%modificadores tamanho [ . precisao] tipo de conversão`
Veja os exemplos: nas figuras (fig. 7.3), (fig. 7.4).

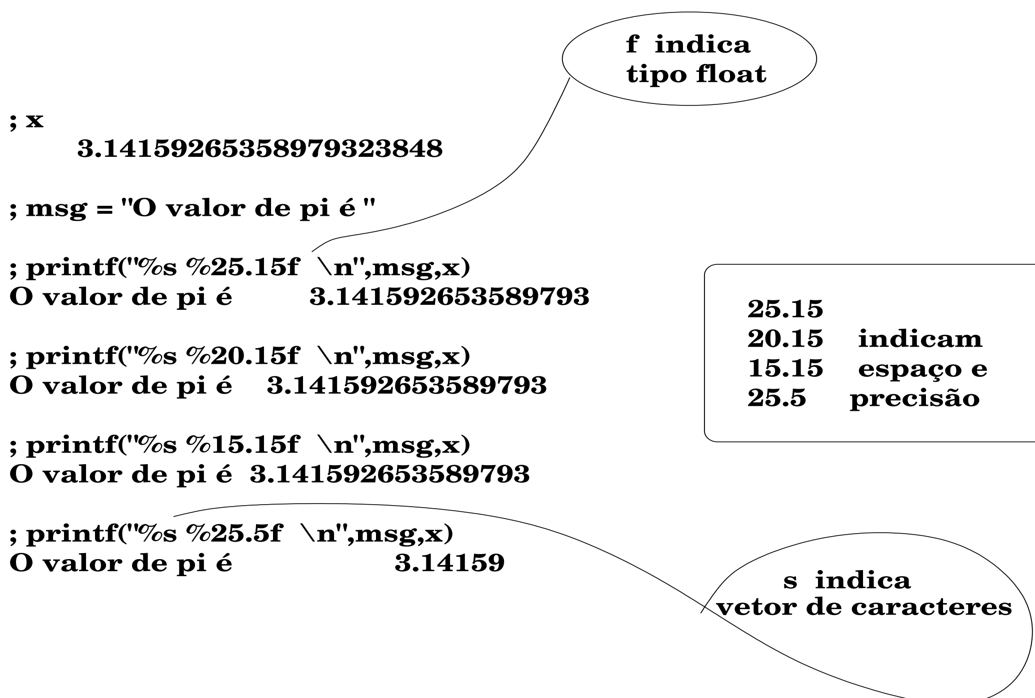

Figura 7.3: Formatação de dados em `printf()`

Leia e rode o programa `format1.c`.

Da mesma forma como diversos tipos de dados podem ser impressos também diversos formatadores, com seus espaçamentos específicos podem ser incluídos

```
# include <stdio.h>
int main()
     {
          float   x=3.141516171788;
          char    msg[50]="O numero pi é aprox. ";
          printf("%s  %25.8f \n", msg,x);
          printf("%s  %25.18f \n", msg,x);
          printf("O número pi é aprox. %5.18lf \n",x);
          printf("O número pi é aprox.  %lf \n",x);
          printf("%s  %5.5e \n", msg,x);
          printf("O número pi é aprox.   %5.5lf \n",x);
          return 0;
     }
```

saída de dados

```
O numero pi é aprox.       3.14151621
O numero pi é aprox.       3.141516208648681641
O número pi é aprox.     3.14151621
O número pi é aprox.  3.141516
O numero pi é aprox.   3.14152e+00
O número pi é aprox.  3.14152
```

Figura 7.4: Uso de `printf()`

num vetor de caracteres, apenas observada a ordem com que eles sejam fornecidos. Rode e analise os dois programas citados e assim como os programas `formatoXX.c`. Veja também `formatadores` no índice remissivo deste livro.

Se você desejar imprimir algum caracter especial, como "%", em `libc` você vai ver como. Em particular "%" será impresso pela linha de comando abaixo:

```
printf(‘‘O percentual do candidato X é %2.2f%%\n’’);
```

Exercícios: 46 *Formatação de dados*

1. *Leia e rode o programa* `format1.c`
2. *Altere o programa* `format1.c` *substituindo* $x = 3.141516171788$ *por* $x = 3.14$ *e analise o resultado.*
3. *Altere o programa* `format1.c` *substituindo %f por %lf e analise o resultado.*

   Solução: `format2.c` *Veja a observação sobre* "aproximação arranjada pelo compilador"
4. *Escreva um programa,* `format3.c` *para imprimir o símbolo % depois de um número real (float).*
5. *Altere o programa que você acabou de fazer para que a frase anunciando a inflação do mes esteja na primeira linha e na segunda linha o índice de inflação.*

   Solução: `format3.c`

6. *Crie variáveis inteiras e use o "coringa" * para usar um formato variável para imprimir espaço e precisão de dados. Sugestão: use um loop.*

   Solução: `format4.c`

7. *Faça um programa que leia, com* `scanf()` *um número inteiro, um número real* `(float)`*, e imprima estes números em duas linhas distintas com a mensagem que indicando o tipo de número.*

   Solução errada em `format5.c`

8. *Conserte o prorama errado* `format5.c`*.*

   Solução: `format6.c`

Observação: 34 *Aproximações arrumadas pelo compilador*

O programa `format2.c` *exibe,* põe em evidência*, erros que o compilador insere nos números. Rode novamente o programa e o leia novamente também para entender o que dissemos.*

Não vamos apresentar uma solução para este problema aqui, ela seria complicada para o nível do texto, mas obviamente que ela existe. Somente queremos alertá-lo para os erros de arredondamento *que o compilador produz, que precisam ficar sob contrôle.*

Exercícios: 47 *Acesso ao disco -* `fprintf()`

1. *Leia o programa* `cadast.c` *e registre a sintaxe da função* `fprintf()`*, semelhanças e dissemelhanças com* `printf()`*.*

2. *Leia e rode o programa* `cadast.c`*, use nomes completos, nome e sobrenome*

3. *Leia o arquivo que você tiver indicado para receber os dados e veja que os dados foram truncados: foi gravado somente o primeiro nome de cada funcionário.*

4. *Troque* `converte_palavra()` *(sscanf()) por* `concatena_palavras()` *(strcat()) e rode novamente o programa. Veja que os nomes foram guardados completamente.*

   Solução: `cadapes3.c`

5. *Estude a suite de programas* `cadapesX.c` *rodando e lendo sucessivamente os programas. Faça alterações nos programas para incluir mais dados no cadastro de funcionários.*

6. *Use o programa* `cadapes3.c` *para construir a sua agenda pessoal de endereços com os campos: nome, endereço, telefone fixo, telefone celular, (outros dados que lhe parecerem importantes).*

   Solução: `agend1.c`

7. *O programa* `agend1.c` *tem um bloco com o comentário "isto é um teste, deve ser apagado". Procure o comentário e veja que a sintaxe de* `fprintf()` *é semelhante a de* `printf()`*. Analise a diferença.*

8. *A função* `sprintf()` *permite a composição automática de vetores de caracteres(dentro de um programa). Ver* `compos_autom.c`*. Altere* `agend.c` *para verificar se o arquivo já existe e usar um nome diferente.*

    ```
    sprintf(deposito, ''%s %d'',NOME_ARQ,extensao)
    ```

    Defina convenientemente `extensao`*, dê lhe um valor adequado.*

    Solução: agend3.c

    O manual do `gcc` *alerta que a função* `sprintf()` *é perigosa, o próximo exercício oferece uma alternativa.*

9. *Refaça a questão anterior usando* `strcat()`

    Solução: agend4.c

Vocabulário: 8 *sprintf(), fprintf()*

Primeiro, deixe-nos lembrá-lo uma fonte de consulta, ajuda, . No sistema `info`*, digite esta palavra numa shell (em LinuX, obviamente), seguida de libc:*

`info libc`

e você tem um manual da linguagem C. Com a barra \ você faz pesquisa de palavras e o `info` *vai abrir no pê da área de trabalho uma janela onde você pode digitar o nome de uma função da linguagem, por exemplo,* `sprintf`*. Tente.*

Este livro teria 1000 páginas se fossemos explicar tudo que existe. E para que repetir, se o que existe é excelente. O nosso objetivo é conduí-lo, pedagogicamente, pelo que já existe.

- `sprintf()`

    ```
    int sprintf (char *S, const char *mascara, ...)
    ```

    Semelhante a `printf()` *mas guarda os dados no vetor de caracteres S. Termina os dados com o caractere NULL. É uma função* `inteira` *devolvendo o número de caracteres colocados em S menos um do NULL. O NULL não é contado. Ver* `agend3.c`*. Não oferece proteção contra gravação de um vetor de dados que supere o tamanho de S. Usar com cuidado.*

- `fprintf()` *É a saída de dados padrão para arquivos em disco. Semelhante a* `printf()` *com a diferença de que o primeiro parâmetro é do tipo* `FILE`*, para onde são dirigidos os dados. Ver* `agend.c`