COLETÂNEA DE EXERCÍCIOS E NOTAS DE AULA EM LINGUAGEM DE PROGRAMAÇÃO C

DAVID BUZATTO

IFSP - CÂMPUS SÃO JOÃO DA BOA VISTA

Coletânea de Exercícios e Notas de Aula em Linguagem de Programação C

Prof. Dr. David Buzatto

Instituto Federal de Educação, Ciência e Tecnologia de São Paulo
Câmpus São João da Boa Vista

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Buzatto, David
Coletânea de exercícios e notas de aula em linguagem de programação c [livro eletrônico] / David Buzatto. -- Vargem Grande do Sul, SP : Ed. do Autor, 2023.
PDF.

ISBN 978-65-00-60543-3

1. Algoritmos de computadores 2. Ciência da computação 3. Linguagem de programação (Computadores) 4. Programação (Computadores) I. Título.

23-141788 CDD-005.133

Índices para catálogo sistemático:

1. C : Linguagem de programação : Computadores : Processamento de dados 005.133

Aline Graziele Benitez - Bibliotecária - CRB-1/3129

À Aurora,
luz da minha vida

SUMÁRIO

APRESENTAÇÃO

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand".

Martin Fowler

DREZADO aluno e leitor, seja bem-vindo! Este livro contém diversos Capítulos, organizados de forma a guiá-lo no processo de fixação dos conteúdos aprendidos em aula, nas disciplinsa básicas de programação de computadores, por meio de exercícios e desafios aplicados em linguagens de programação. A ordem dos Capítulos obedece a um caminho lógico que será empregado pelo professor no seu processo de aprendizagem, ou seja, a ordem dos Capítulos segue a ordem cronológica dos tópicos que serão apresentados, ensinados e treinados em laboratório. Note que apesar deste livro ter sido organizado para apoiar no desenvolvimento de disciplinas, nada impede que seja usado como material de apoio para outros cursos, bem como para pessoas que desejam treinar por meio de exercícios suas habilidades com programação básica.

Antes de começarmos, eu gostaria de me apresentar:

Meu nome é David Buzatto e sou Bacharel em Sistemas de Informação pela Fundação de Ensino Octávio Bastos (2007), Mestre em Ciência da Computação pela Universidade Federal de São Carlos (2010) e Doutor em Biotecnologia pela Universidade de Ribeirão Preto (2017). Tenho interesse em construção de compiladores, teoria da computação, análise e projeto de algoritmos, estruturas de dados, algoritmos em bioinformática e desenvolvimento de jogos eletrônicos. Atualmente sou professor efetivo do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP), câmpus

São João da Boa Vista. A melhor forma de contatar é através do e-mail davidbuzatto@ifsp.edu.br.

Para que você possa aproveitar a leitura deste livro de forma plena, vale a pena entender alguns padrões que foram utilizados neste texto. As cinco caixas apresentadas abaixo serão empregadas para mostrar, a você leitor, respectivamente, boas práticas de programação, exemplos, conteúdos complementares para melhorar e aprofundar seu aprendizado, dicas pertinentes ao que está sendo apresentado e, por fim, itens que precisam ser tratados com cuidado ou que podem acarretar em erros comuns de programação.

Boa Prática

As caixas de **Boa Prática** serão usadas para apresentar sugestões de como se deve escrever código-fonte de maneira limpa, organizada e legível, bem como outras técnicas e padrões para auxiliar nesses objetivos.

Exemplo

Pequenos trechos de código serão mostrados nas caixas de **Exemplo**, além de informações relativas aos códigos mostrados.

Saiba Mais

Nas caixas de **Saiba Mais** serão apresentados recursos complementares para você conhecer mais sobre determinado assunto.

Dica

As caixas de **Dica** vão ser usadas com o objetivo de apresentar uma dica ou complementar alguma informação.

Atenção!

Quando for necessário realizar alguma observação mais importante, serão empregadas as caixas de **Atenção!**.

Além disso, diversos exercícios conterão caixas de entrada e/ou saída, em que serão apresentados exemplos de dados de entrada para o problema proposto e de saída que devem ser obtidos após o processamento da entrada fornecida como exemplo.

Note também que este livro foi escrito de forma quase coloquial, com o objetivo de conversar com você, tentando te ensinar, e não com o objetivo de ser um material de

pesquisa.

É de suma importância que você resolva cada um dos exercícios básicos de cada Capítulo, visto que a utilização de uma linguagem de programação, e mais importante ainda, a obtenção de maturidade no desenvolvimento de algoritmos, é ferramenta primordial para o seu sucesso profissional e intelectual na área da Computação.

Um ponto importante sobre os exercícios é que todas as saídas apresentadas ao usuário, quando feitas em modo texto, serão feitas sem usar acentos. Ou seja, se um programa precisar exibir uma palavra acentuada, algo como, por exemplo, “Olá”, você deverá digitar tal palavra sem usar o acento, ficando “Ola”. O principal motivo para essa restrição é que normalmente a página de códigos utilizada para executar o programa em modo texto diverge da codificação utilizada na compilação, criando inconsistências, principalmente ao se usar as linguagens C e C++ em ambiente Windows. Por isso, nas entradas e saídas de todos os enunciados, você perceberá que faltarão acentos nas palavras acentuadas, o que é feito de propósito visto essa “restrição”. Há como contornar tal característica, mas isso envolve alguns detalhes que fogem do escopo desta obra e que lhe forçaria a inserir código no seu programa que não faz parte da solução, nem dos conceitos que devem ser aprendidos. Usando as linguagens Java ou Python não haverá tal problema, entretanto, será mantido o padrão de não usar acentos para todos os exercícios. Essas quatro linguagens de programação foram citadas, pois serão linguagens que, dependendo da disciplina em que essa lista for usada, poderão estar sendo ensinadas/aplicadas. Os programas de exemplo conterão acento normalmente.

Por fim, espero sinceramente que este livro lhe seja útil! Vamos começar?

CAPÍTULO 1

ENTRADA E SAÍDA PADRÃO FORMATADOS

"A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original".

Albert Einstein

NESTE Capítulo serão treinados os conceitos E/S (Entrada e Saída)[1], sendo estes fundamentais para o funcionamento de qualquer programa de computador. Além disso, serão utilizadas variáveis e outros comandos, além de expressões básicas aprendidas em aula. Os trechos de código abaixo contém lembretes das estruturas principais que devem ser usadas para resolver os exercícios propostos. Note que em todos os Capítulos a maioria dos conceitos necessários serão apresentados dentro de trechos de código mostrados no início de cada um. Sendo assim, sempre estude os códigos apresentados no início de cada Capítulo e leia com atenção os comentários inseridos neles.

[1] Em inglês o termo é I/O que vem de *Input* (Entrada) e *Output* (Saída).

1.1 Exemplos em Linguagem C

Saída padrão usando a função printf

```c
/*
 * Arquivo: EntradaSaidaPadraoFormatadosOlaMundo.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>

int main() {

    // comentários de uma linha iniciam com // (duas barras)

    // a função printf direciona o texto inserido (entre aspas duplas)
    // para a saída padrão
    printf( "Ola mundo!!" );

    return 0;

}
```

Caracteres de escape comuns

```c
/*
 * Arquivo: EntradaSaidaPadraoFormatadosSaida.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>

int main() {

    // comentários de múltiplas linhas iniciam com /* e terminam com */

    /* o caractere \ (contra barra) é usado para "escapar" caracteres
     * especiais. Os principais são \n e \t servindo, respectivamente,
     * para pular uma linha do console e continuar a saída de texto
     * no início da próxima linha e para inserir um caractere de
```

```c
     * tabulação.
     */
    printf( "Um texto!\n" );
    printf( "Outro texto, na linha de baixo!\n" );
    printf( "\tEssa linha inicia tabulada!" );

    return 0;

}
```

Declaração de variáveis

```c
/*
 * Arquivo: EntradaSaidaPadraoFormatadosDeclaracaoVariaveis.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>

int main() {

    /* cada variável precisa de um tipo e de um identificador (nome)
     *
     * os tipos que serão usados por enquanto são:
     *    int: número inteiro
     *    char: caractere
     *    float: número decimal/ponto flutuante
     */

    int idade;                // variável inteira chamada idade
    char letra;               // variável para caracteres chamada letra
    float altura;             // variável decimal chamada altura

    int paresDeTenis = 5;     // variável inteira chamada paresDeTenis
                              // inicializada com o valor 5

    char letraInicial = 'D';  // variável para caracteres chamada
                              // letraInicial inicializada com o
                              // valor 'D'

```

```
30      float peso = 90.5;          // variável decimal chamada peso
31                                  // inicializada com o valor 90.5
32                                  // atenção: use . (ponto) como separador
33                                  // decimal!
34
35      return 0;
36
37  }
```

Especificadores de Formato

```
1   /*
2    * Arquivo: EntradaSaidaPadraoFormatadosFormatacao.c
3    * Autor: Prof. Dr. David Buzatto
4    */
5
6   #include <stdio.h>
7   #include <stdlib.h>
8
9   int main() {
10
11      /* a função printf é capaz de interpolar dados dentro
12       * do texto (string) que irá imprimir na saída padrão.
13       *
14       * para isso, é necessário marcar posições dentro da string
15       * usando o caractere % (porcento) e utilizar um especificador
16       * de formato específico para cada tipo de variável.
17       *
18       * os specificadores de formato que serão usados por
19       * enquanto são:
20       *    %d: para variáveis inteiras (int)
21       *    %c: para variáveis de caracteres (char)
22       *    %f: para variáveis decimais (float)
23       *
24       * alguns especificadores possuem opções de formatação.
25       * por exemplo, para fixar a quantidade de casas decimais
26       * que serão exibidas para uma variável float, usa-se:
27       *    %.nf: onde "n" é a quantidade de casas decimais.
28       *    Exemplo: %.2f => o valor da variável float
29       *             será formatado usando duas casas decimais
30       */
```

```c

    float pi = 3.1415;
    float raio = 20.78;
    float circunferencia = 2 * pi * raio;
    float area = pi * raio * raio;

    printf( "O circulo de raio %f tem:\n", raio );
    printf( "\tCircunferencia = %.2f\n", circunferencia );
    printf( "\tArea = %.2f\n", area );

    return 0;

}
```

Operadores aritméticos

```c
/*
 * Arquivo: EntradaSaidaPadraoFormatadosAritmetica.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>

int main() {

    /* existem na linguagem C quatro operadores aritméticos:
     *     +: adição
     *     -: subtração
     *     *: multiplicação
     *     /: divisão
     *
     * esses operadores atuam de forma específica dependendo do
     * tipo numérico sendo operado. isso se nota principalmente
     * em relação ao operador / (divisão) quando atuado em valores
     * inteiros e de ponto flutuante!
     *
     * além desses quatro operadores, ainda existe o operador de
     * resto/módulo que é dado pelo símbolo % (porcento) e que é
     * usado apenas para números inteiros.
     */
```

```
26
27      int numeroInteiro1 = 9;
28      int numeroInteiro2 = 2;
29      float numeroDecimal1 = 9;
30      float numeroDecimal2 = 2;
31
32      // resulta em 4
33      int divisaoInteira = numeroInteiro1 / numeroInteiro2;
34
35      // resulta em 4.5
36      float divisaoDecimal = numeroDecimal1 / numeroDecimal2;
37
38      printf( "Inteiros: %d / %d = %d\n",
39              numeroInteiro1, numeroInteiro2, divisaoInteira );
40      printf( "Decimais: %f / %f = %f\n",
41              numeroDecimal1, numeroDecimal2, divisaoDecimal );
42
43      return 0;
44
45  }
```

Entrada padrão usando a função scanf

```
1   /*
2    * Arquivo: EntradaSaidaPadraoFormatadosEntrada.c
3    * Autor: Prof. Dr. David Buzatto
4    */
5
6   #include <stdio.h>
7   #include <stdlib.h>
8
9   int main() {
10
11      /* para realizar a entrada de dados em um programa
12       * pelo dispositivo de entrada padrão configurado no sistema
13       * operacional (usualmente o teclado) usa-se a função scanf.
14       *
15       * essa função funcionará de forma parecida que a função
16       * printf, entretanto, ao invés de direcionar o valor de uma
17       * variável para a saída padrão, ela direcionará o valor
18       * fornecido através da entrada padrão para uma variável.
```

```c
     *
     * os especificadores de formato utilizados na função printf
     * também são usados na função scanf.
     *
     * *** ATENÇÃO! ***
     * cuidado ao usar %c na função scanf. Preceda o especificador
     * com um caractere de espaço!
     * Por exemplo, scanf( " %c", &suaVariavel );
     */

    char primeiraLetra;
    int idade;
    float peso;
    float altura;
    float imc;

    printf( "Entre com a primeira letra de seu primeiro nome: " );
    scanf( " %c", &primeiraLetra );

    printf( "Entre com sua idade: " );
    scanf( "%d", &idade );

    printf( "Entre com seu peso: " );
    scanf( "%f", &peso );

    printf( "Entre com sua altura: " );
    scanf( "%f", &altura );

    imc = peso / ( altura * altura );

    printf( "%c, seu IMC é: %.2f", primeiraLetra, imc );

    return 0;

}
```

| Boa Prática

Sempre utilize comentários no código fonte com o objetivo de auxiliar você e/ou outro programador que fará a leitura do código posteriormente.

| Boas Práticas

- Declare uma variável por linha!
- Ao nomear variáveis, dê preferência para nomes que identificam corretamente o propósito delas e inicie seus nomes com letras minúsculas;
- Um padrão comum para nomenclatura de variáveis em linguagens de programação mais modernas é chamado de CamelCase. Nesse padrão, caso uma variável seja uma palavra composta ou uma frase, une-se todas as palavras e as iniciamos com letras maísculas. Por exemplo, se quisermos declarar uma variável chamada "altura da pessoa", faríamos algo assim: `alturaDaPessoa`;

| Exemplo

No exemplo apresentado abaixo é mostrada a declaração e inicialização de três variáveis, sendo que a representação gráfica do que acontece na memória principal após a execução desse código é apresentada na Figura 1.1.

Declaração e inicialização de variáveis

```
1 int idade = 38;
2 char letra = 'd';
3 float altura = 1.84;
```

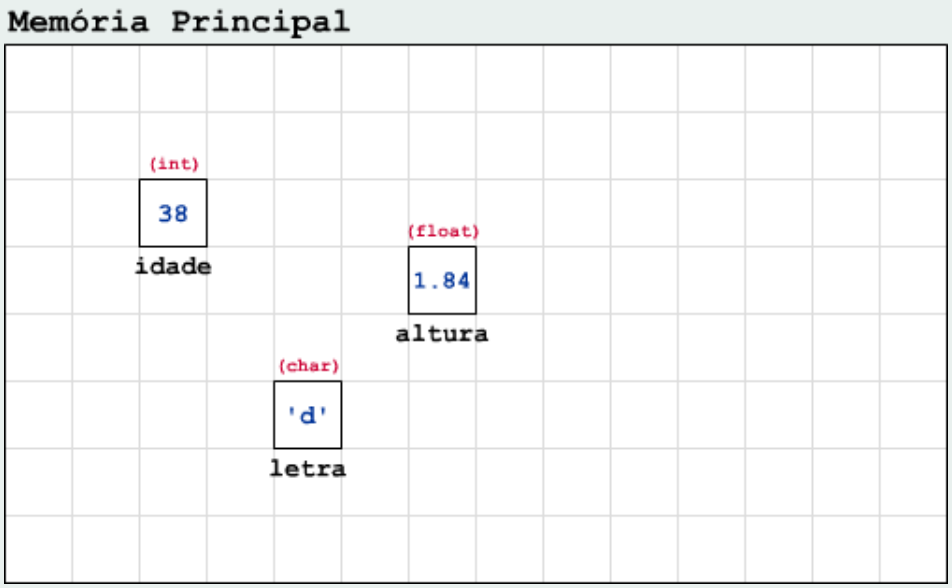


Figura 1.1: Variáveis na memória

Saiba Mais

Uma descrição do padrão CamelCase pode ser encontrada no link **<https://pt.wikipedia.org/wiki/CamelCase>**. Esse padrão será usado praticamente o tempo todo enquanto você programar. Note que algumas linguagens possuem seus próprios padrões de codificação dependendo de suas respectivas comunidades e, além disso, cada empresa que você trabalhar poderá ter um padrão específico.

Atenção!

- Não é permitido iniciar o nome (identificador) de uma variável utilizando números;
- Use somente letras (sem acentos), números e "_" (*underscore*) para nomear uma variável;
- Identificadores de variáveis não podem conter espaços.

Atenção!

Cuidado ao usar o especificador de formato `%c` na função `scanf`. Sempre preceda-o de um caractere de espaço. Por exemplo:

```
1 char variavelChar;
2 ...
3 scanf( " %c", &variavelChar );
```

Dica

Para imprimir um símbolo de porcentagem (%) na saída ao se usar a função `printf`, preceda-o de outro símbolo de porcentagem. Por exemplo:

```
1 printf( "9%%" );     // imprime 9%
```

1.1.1 Operadores Aritméticos e de Atribuição

Na Tabela 1.1 abaixo são apresentados os operadores aritméticos e na Tabela 1.2 os operadores de atribuição da linhagem C. A precedência dos operadores aritmétidos é a mesma da álgebra, podendo também serem agrupados usando parênteses.

Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da Divisão Inteira (módulo)

Tabela 1.1: Operadores aritméticos

 # | Exemplo

Usando o operador aritmético de adição

```
1 int a = 1;
2 int b = 2;
3
4 // a variável r conterá o resultado da soma de a e b
5 int r = a + b;
```

Operador	Significado
=	Atribuição simples
+=	Atribuição composta (adição)
-=	Atribuição composta (subtração)
*=	Atribuição composta (multiplicação)
/=	Atribuição composta (divisão)
%=	Atribuição composta (resto)

Tabela 1.2: Operadores de atribuição e atribuição composta

 # | Exemplo

Usando o operador de atribuição composto para adição

```
1 int a = 1;
2 int b = 1;
3 b += a;  // o mesmo que b = b + a;
4 // idem para os outros operadores de atribuição compostos
```

Saiba Mais

Caso queira ler mais sobre expressões na linguagem C, você pode consultar a documentação/referência "oficial" da linguagem no link **<https://en.cppreference.com/w/c/language/expressions>**. A página principal dessa documentação é acessível através do link **<https://en.cppreference.com/w/c/>**.

Agora que já conhecemos o básico da linguagem de programação C, nós podemos começar a trabalhar nos exercícios. Como já frisado, o desenvolvimento das tarefas é ESSENCIAL para o sucesso no aprendizado de qualquer linguagem de programação. Vamos lá!

1.2 Exercícios

Exercício 1.1: Escreva um programa que imprima a mensagem "Ola Mundo!" quando executado.

Arquivo com a solução: ex1.1.c

Saída

```
Ola Mundo!
```

Atenção!

Note que o primeiro exercício possui somente a caixa de **Saída**, indicando que nesse exercício seu programa deve apenas gerar saída para o usuário. Vários dos próximos exercícios seguirão a mesma ideia.

Exercício 1.2 (DEITEL; DEITEL, 2016): Escreva um programa que imprima o seguinte desenho quando executado.

Arquivo com a solução: ex1.2.c

Saída

```
***
**
*
*******
```

⚠ | Atenção!

Em alguns exercícios será necessário apresentar espaços na saída de modo a "espaçar caracteres". Para que a quantidade de espaços fique evidente em cada linha, nessas saídas eles serão apresentados como asteriscos quase pretos. Esse padrão será usado no decorrer do livro.

Exercício 1.3 (DEITEL; DEITEL, 2016): Escreva um programa que imprima o seguinte desenho quando executado.

Arquivo com a solução: ex1.3.c

Saída

```
########
#******#
#******#
#******#
#******#
########
```

Exercício 1.4 (DEITEL; DEITEL, 2016): Escreva um programa que imprima o seguinte desenho quando executado.

Arquivo com a solução: ex1.4.c

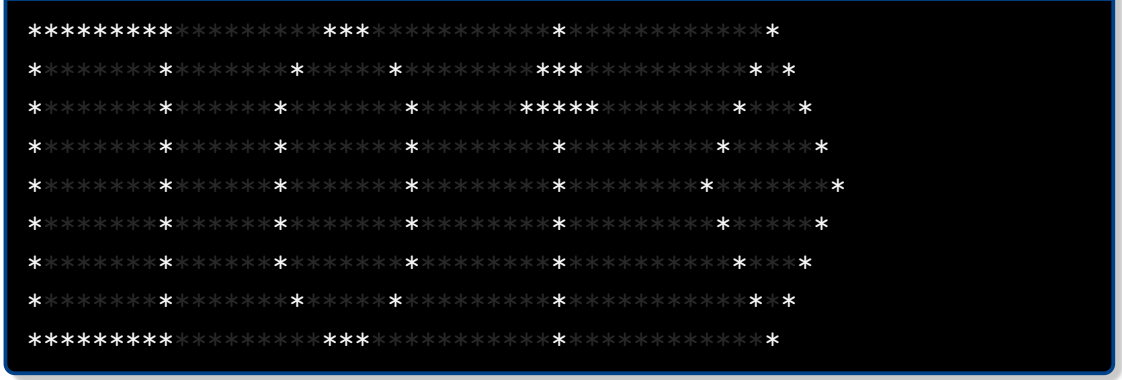
Saída

```
*******
******
*****
****
***
**
```

Exercício 1.5 (DEITEL; DEITEL, 2016): Escreva um programa que imprima o seguinte desenho quando executado.

Arquivo com a solução: ex1.5.c

Saída



⚠ | Atenção!

A partir do próximo exercício começaremos a trabalhar com a entrada de dados do usuário, sendo assim, além da caixa de **Saída**, será apresentada também a caixa de **Entrada**, com o objetivo lhe mostrar os dados que foram fornecidos ao programa para que ele gerasse a saída apropriada.

Exercício 1.6: Escreva um programa que peça para o usuário fornecer o valor de dois números inteiros. O programa deve usar o valor dos números para calcular o valor das quatro operações aritméticas básicas (adição, subtração, multiplicação e divisão). O resultado de cada operação deve ser armazenado em uma variável diferente. No final, o programa deve exibir ao usuário o resultado de cada operação.

Arquivo com a solução: ex1.6.c

Entrada

```
Primeiro numero: 7
Segundo numero: 3
```

Saída

```
7 + 3 = 10
7 - 3 = 4
7 * 3 = 21
7 / 3 = 2
```

Exercício 1.7: Escreva um programa que peça para o usuário fornecer o valor do lado de um quadrado em uma unidade arbitrária. O valor deve ser um número inteiro. O

programa deve calcular os valores da área e do perímetro desse quadrado. O resultado de cada cálculo deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário os valores obtidos. Lembrando que:

- $P = 4l$
- $A = l^2$
- Onde:
  - P é o perímetro do quadrado;
  - A é a área do quadrado;
  - l é o valor do lado do quadrado.

Arquivo com a solução: ex1.7.c

Entrada

```
Valor do lado: 5
```

Saída

```
Perimetro = 20
Area = 25
```

Exercício 1.8: Escreva um programa que peça para o usuário fornecer os valores da largura e da altura de um retângulo em uma unidade arbitrária. Os valores devem ser números inteiros. O programa deve calcular os valores da área e perímetro desse retângulo. O resultado de cada cálculo deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário os valores obtidos. Lembrando que:

- $P = (2l) + (2h)$
- $A = lh$
- Onde:
  - P é o perímetro do retângulo;
  - A é a área do retângulo;
  - l é o valor da largura do retângulo;
  - h é o valor da altura do retângulo.

Arquivo com a solução: ex1.8.c

Entrada

```
Valor da largura: 5
Valor da altura: 10
```

Saída

```
Perimetro = 30
Area = 50
```

Exercício 1.9: Escreva um programa que peça para o usuário fornecer os valores da base e da altura de um triângulo em uma unidade arbitrária. Os valores devem ser números inteiros. O programa deve calcular o valor da área desse triângulo. O resultado deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário o valor obtido. Lembrando que:

- $A = \frac{bh}{2}$
- Onde:
  - A é a área do triângulo;
  - b é o valor da base do triângulo;
  - h é o valor da altura do triângulo.

Arquivo com a solução: ex1.9.c

Entrada

```
Valor da base: 10
Valor da altura: 5
```

Saída

```
Area = 25
```

Exercício 1.10: Escreva um programa que peça para o usuário fornecer os valores da base maior, da base menor e da altura de um trapézio em uma unidade arbitrária. Os valores devem ser números inteiros. O programa deve calcular o valor da área desse trapézio. O resultado deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário o valor obtido. Lembrando que:

- $A = \frac{(B+b)h}{2}$
- Onde:
  - A é a área do trapézio;
  - B é o valor da base maior do trapézio;
  - b é o valor da base menor do trapézio;
  - h é o valor da altura do trapézio.

Arquivo com a solução: ex1.10.c

Entrada

```
Valor da base maior: 10
Valor da base menor: 6
Valor da altura: 5
```

Saída

```
Area = 40
```

Exercício 1.11: Escreva um programa que peça para o usuário fornecer os valores da diagonal maior e da diagonal menor de um losango em uma unidade arbitrária. Os valores devem ser números inteiros. O programa deve calcular o valor da área desse losango. O resultado deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário o valor obtido. Lembrando que:

- $A = \frac{Dd}{2}$
- Onde:
  - A é a área do losango;
  - D é o valor da diagonal maior do losango;
  - d é o valor da diagonal menor do losango.

Arquivo com a solução: ex1.11.c

Entrada

```
Valor da diagonal maior: 12
Valor da diagonal menor: 6
```

Saída

```
Area = 36
```

⚠ | Atenção!

A partir de agora, a maioria dos nossos programas que lidarão com números farão uso do tipo `float` (decimal) além do tipo `int` (inteiro). O tipo apropriado dependerá da natureza do valor, bem como de possíveis orientações contidas na atividade. Em todos os exercícios os números decimais precisarão ser formatados com uma quantidade específica de casas decimais. Essa formatação será normalmente informada nas tarefas, mas caso não sejam, confira na

saída a quantidade de casas usadas. Normalmente essa quantidade será duas casas decimais.

Exercício 1.12: Escreva um programa que peça para o usuário fornecer um valor qualquer que deve ser um número decimal. O programa deve exibir esse número três vezes: Na primeira, deve ser exibido o número sem nenhuma formatação. Na segunda, o número deve ser formatado para mostrar duas casas decimais. Por fim, na terceira, o número deve ser formatado para mostrar três casas decimais.

Arquivo com a solução: ex1.12.c

Entrada

```
Entre com um valor qualquer: 153.4671
```

Saída

```
153.467102
153.47
153.467
```

Exercício 1.13: Repita o Exercício 1.6, usando agora números decimais. Os resultados devem ser formatados usando duas casas decimais. Reescreva o programa ao invés de copiá-lo!

Arquivo com a solução: ex1.13.c

Entrada

```
Primeiro numero: 7.5
Segundo numero: 3.5
```

Saída

```
7.50 + 3.50 = 11.00
7.50 - 3.50 = 4.00
7.50 * 3.50 = 26.25
7.50 / 3.50 = 2.14
```

Exercício 1.14: Repita o Exercício 1.7, usando agora números decimais. Os resultados devem ser formatados usando duas casas decimais. Reescreva o programa ao invés de copiá-lo!

Arquivo com a solução: ex1.14.c

Entrada

```
Valor do lado: 5.5
```

Saída

```
Perimetro = 22.00
Area = 30.25
```

Exercício 1.15: Repita o Exercício 1.8, usando agora números decimais. Os resultados devem ser formatados usando duas casas decimais. Reescreva o programa ao invés de copiá-lo!

Arquivo com a solução: ex1.15.c

Entrada

```
Valor da largura: 5.5
Valor da altura: 9.5
```

Saída

```
Perimetro = 30.00
Area = 52.25
```

Exercício 1.16: Repita o Exercício 1.9, usando agora números decimais. Os resultados devem ser formatados usando duas casas decimais. Reescreva o programa ao invés de copiá-lo!

Arquivo com a solução: ex1.16.c

Entrada

```
Valor da base: 10.5
Valor da altura: 5.75
```

Saída

```
Area = 30.19
```

Exercício 1.17: Repita o Exercício 1.10, usando agora números decimais. Os resultados devem ser formatados usando duas casas decimais. Reescreva o programa ao invés de

copiá-lo!

Arquivo com a solução: ex1.17.c

Entrada

```
Valor da base maior: 10.5
Valor da base menor: 6.25
Valor da altura: 6.75
```

Saída

```
Area = 56.53
```

Exercício 1.18: Repita o Exercício 1.11, usando agora números decimais. Os resultados devem ser formatados usando duas casas decimais. Reescreva o programa ao invés de copiá-lo!

Arquivo com a solução: ex1.18.c

Entrada

```
Valor da diagonal maior: 12.25
Valor da diagonal menor: 6.6
```

Saída

```
Area = 40.42
```

Exercício 1.19: Escreva um programa que peça para o usuário fornecer o valor do raio de um círculo em uma unidade arbitrária. O valor deve ser um número decimal. O programa deve calcular os valores do diâmetro, da circunferência e da área desse círculo. O resultado de cada cálculo deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário os valores obtidos. Lembrando que:

- $D = 2r$
- $C = 2\pi r$
- $A = \pi r^2$
- Onde:
  - D é o diâmetro do círculo;
  - C é a circunferência do círculo;
  - A é a área do círculo;
  - r é o valor do raio do círculo;

– π é a constante matemática Pi. Para esse exercício, considere que π = 3.141592.

Arquivo com a solução: ex1.19.c

Entrada

```
Valor do raio do circulo: 10.5
```

Saída

```
Diametro = 21.00
Circunferencia = 65.97
Area = 346.36
```

Exercício 1.20: Escreva um programa que peça para o usuário fornecer dois números inteiros. O programa deve calcular e exibir a média aritmética desses dois números. Armazene essa média em uma variável.

Arquivo com a solução: ex1.20.c

Entrada

```
Primeiro numero: 5
Segundo numero: 10
```

Saída

```
Media aritmetica: 7
```

Exercício 1.21: Escreva um programa que peça para o usuário fornecer um número inteiro. O programa deve calcular exibir o sucessor e o antecessor desse número. Armazene ambos os números em variáveis.

Arquivo com a solução: ex1.21.c

Entrada

```
Forneca um numero inteiro: 1992
```

Saída

```
Sucessor de 1992: 1993
Antecessor de 1992: 1991
```

Exercício 1.22: Escreva um programa que peça para o usuário fornecer o valor de um produto. O programa deve calcular e exibir o preço de venda do produto, com um desconto de 9%, usando duas casas decimais. Armazene o preço de venda do produto em uma variável. Lembre-se que para imprimir um símbolo de porcentagem (%) na saída ao se usar a função `printf`, você deve precedê-lo de outro símbolo de porcentagem.

Arquivo com a solução: ex1.22.c

Entrada

```
Valor do produto: 5.79
```

Saída

```
Preco de venda com 9% de desconto: 5.27
```

Exercício 1.23: Escreva um programa que peça para o usuário fornecer o ano de seu nascimento e o ano atual. O programa deve calcular e exibir a idade atual aproximada do usuário.

Arquivo com a solução: ex1.23.c

Entrada

```
Ano de nascimento: 1985
Ano atual: 2018
```

Saída

```
Idade aproximada: 33 anos
```

Exercício 1.24: Escreva um programa que calcule e exiba, usando duas casas decimais, o valor líquido do salário de um professor. O programa deve pedir para o usuário fornecer o valor da hora/aula, a quantidade de aulas e a porcentagem de desconto do INSS.

Arquivo com a solução: ex1.24.c

Entrada

```
Valor da hora/aula: 20.78
Quantidade de aulas: 40
Porcentagem de desconto do INSS: 26.5
```

Saída

```
Salario Liquido: 610.93
```

Exercício 1.25: Escreva um programa que peça para o usuário fornecer uma temperatura em graus Fahrenheit. O programa deve calcular a temperatura correspondente em graus Celsius. O resultado do cálculo deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário o valor obtido. Lembrando que:

- $C = \frac{F - 32}{1,8}$
- Onde:
  - C é a temperatura em graus Celsius;
  - F é a temperatura em graus Fahrenheit.

Arquivo com a solução: ex1.25.c

Entrada

```
Temperatura em graus Fahrenheit: 125
```

Saída

```
125.00 graus Fahrenheit correspondem a 51.67 graus Celsius
```

Exercício 1.26: Escreva um programa que peça para o usuário fornecer uma temperatura em graus Celsius. O programa deve calcular a temperatura correspondente em graus Fahrenheit. O resultado do cálculo deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário o valor obtido. Lembrando que:

- $F = 1,8C + 32$
- Onde:
  - F é a temperatura em graus Fahrenheit;
  - C é a temperatura em graus Celsius.

Arquivo com a solução: ex1.26.c

Entrada

```
Temperatura em graus Celsius: 36
```

Saída

```
36.00 graus Celsius correspondem a 96.80 graus Fahrenheit
```

Espero que sua experiência no desenvolvimento das atividades esteja sendo prazerosa. Caso você seja aluno de algum curso técnico ou superior, a carreira que você escolheu na informática/computação para seu futuro será baseada nesse tipo de atividade. Vamos continuar!

CAPÍTULO 2

ESTRUTURAS CONDICIONAIS

"Que não daria eu, para voltar a mocidade? Aceitaria qualquer condição, menos, é claro, fazer exercícios, acordar cedo e tornar-me respeitável".

Oscar Wilde

S estruturas condicionais, ou de seleção/decisão, permitem que, a partir da avaliação de uma expressão que gera um valor lógico, ou seja, um valor verdadeiro ou falso, o fluxo de execução do programa seja alterado. Elas têm papel fundamental na construção de algoritmos e, por consequência, na escrita de programas. Neste Capítulo são apresentadas as estruturas condicionais `if` e `switch` que são utilizadas para esse objetivo.

2.1 Estrutura Condicional *if* (se)

Para entendermos melhor o funcionamento das estruturas condicionais iremos fazer um paralelo da sua sintaxe (forma de escrever), com sua execução por meio de diagramas de transição de estados que chamaremos de diagramas de fluxo. Esses diagramas são parecidos com os fluxogramas. Nos próximos três trechos de código você verá

números entre parênteses, por exemplo, `(1)`, que são análogos as números contidos dentro dos estados, representados por círculos, nos diagramas.

Saiba Mais

Caso queira saber mais sobre fluxogramas, o artigo da Wikipedia sobre o assunto é um bom começo. Veremos muitos tipos de diagramas durante o curso e eles normalmente são fáceis de entender, pois são bastante intuitivos. O link para o artigo sobre fluxogramas é **<https://pt.wikipedia.org/wiki/Fluxograma>**.

2.1.1 Exemplo e Diagrama de Fluxo da Estrutura Condicional *if*

A forma mais simples de usarmos a estrutura condicional *if* é utilizá-la para executar algum trecho de código apenas se a condição que ela verifica `(teste)` é verdadeira.

Estrutura do *if* - Verifique a Figura 2.1

```
1 // antes
2 (1)
3       (2)
4 if ( teste ) {
5     (3)
6 }
7 (4)
8 // depois
```

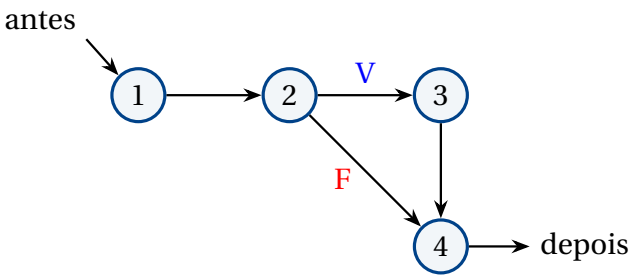


Figura 2.1: Fluxo de execução da estrutura condicional *if*

2.1.2 Exemplo e Diagrama de Fluxo da Estrutura Condicional *if...else* (se...senão)

Outra forma de utilizarmos a estrutura condicional *if* é associar também à ela um trecho de código que será executado caso sua condição seja avaliada como falsa. Ou seja, temos um bloco de código que executa caso a condição seja verdadeira e um caso ela seja falsa somente.

Estrutura do *if...else* - Verifique a Figura 2.2

```
1  // antes
2  (1)
3        (2)
4  if ( teste ) {
5      (3)
6  } else {
7      (4)
8  }
9  (5)
10 // depois
```

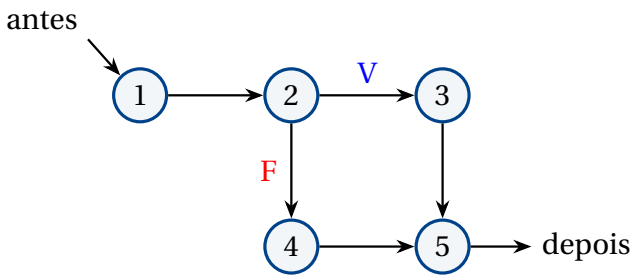


Figura 2.2: Fluxo de execução da estrutura condicional *if...else*

2.1.3 Exemplo e Diagrama de Fluxo da Estrutura Condicional *if...else f...else* (se...senão se...senão)

Por fim, também podemos realizar o encadeamento de diversas estruturas *if*, permitindo que possamos testar condições que dependem que outras tenham sido avaliadas como falsas previamente para que sejam executadas.

Estrutura do *if...else if...else* - Verifique a Figura 2.3

```
(1) // antes
       (2)
if ( teste1 ) {
    (3)
                (4)
} else if ( teste2 ) {
    (5)
                (6)
} else if ( teste3 ) {
    (7)
} else {
    (8)
}
(9) // depois
```

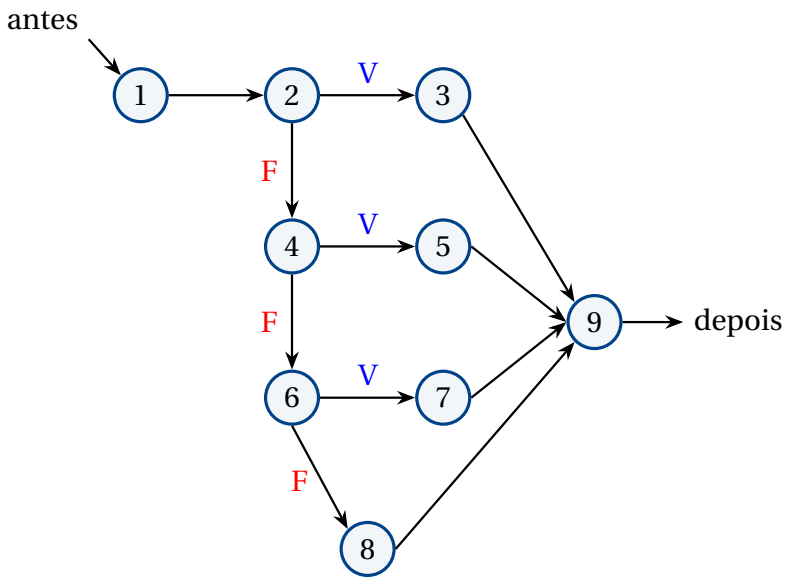


Figura 2.3: Fluxo de execução da estrutura condicional *if... else if... else*

Veja ainda que os símbolos de “abre chave” e “fecha chave” `{ e }` que delimitam os blocos de código da estrutura condicional *if*, bem como de outras estruturas que veremos mais adiante, são de uso opcional caso o bloco de código contenha apenas uma linha, mas...

Boa Prática

Independente da opção de usar ou não chaves para delimitar blocos com apenas uma linha de código é uma boa prática **sempre** utilizar chaves para delimitar blocos de uma linha, evitando assim possíveis erros lógicos.

2.1.4 Operadores de Igualdade, Relacionais e Lógicos

A construção de testes lógicos nas linguagens de programação é feita usando alguns conjuntos de operadores binários, ou seja, que atuam sobre dois operandos. Os operadores de igualdade verificam a igualdade ou a desigualdade entre dois valores/operandos. O operador de igualdade “igual a” da linguagem C `==` retorna verdadeiro como resultado caso dois valores sejam iguais ou falso caso não sejam. De modo inverso, o operador de igualdade “diferente de” da linguagem C `!=` retorna verdadeiro caso os dois valores avaliados sejam diferentes ou falso caso sejam iguais. Na Tabela 2.1 esses operadores são apresentados.

Operador	Significado
==	Igual a
!=	Diferente de

Tabela 2.1: Operadores de igualdade

Além dos operadores de igualdade, podemos também estabelecer relações entre os operandos. Para isso usamos os operadores relacionais, apresentados na Tabela 2.2. Esses operadores atuam de forma análoga aos operadores de igualdade. Por exemplo, o operador relacional “menor que” da linguagem C `<` retorna verdadeiro caso o primeiro operando seja estritamente menor que o segundo operando ou falso caso contrário.

Operador	Significado
<	Menor que
<=	Menor ou igual a
>	Maior que
>=	Maior ou igual a

Tabela 2.2: Operadores relacionais

Os conjuntos de operadores apresentados anteriormente lidam com quaisquer tipos de valores que podem ser comparados entre sí. Já os operadores lógicos, apresentados na Tabela 2.3, lidam com operandos que têm valores lógicos, ou seja, verdadeiro ou falso. As tabelas verdade desses operadores podem ser vistas na Tabela 2.4a, na Tabela 2.4b e na Tabela 2.4c.

Operador	Significado
&&	E
||	OU
!	NÃO

Tabela 2.3: Operadores lógicos de curto-circuito

E (&&)		
	V	F
V	V	F
F	F	F

(a) Operador E

OU (||)		
	V	F
V	V	V
F	V	F

(b) Operador OU

NÃO (!)	
V	F
F	V

(c) Operador NÃO

Tabela 2.4: Tabelas verdade dos operadores lógicos `E`, `OU` e `NÃO`

Ainda é importante frisar que na linguagem C, todo e qualquer valor diferente de zero é considerado como verdadeiro e todo e qualquer valor igual à zero é considerado como falso. Essa característica pode levar a problemas de lógica nos algoritmos pelo simples fato de um programador menos experiente, ou não avisado, confundir o operador de igualdade "igual a" `==` com o operador de atribuição `=`. Sendo assim:

⚠ | Atenção!

Cuidado ao criar expressões que testam a igualdade dentre dois operandos. O operador de igualdade "igual a" é presentado por dois sinais de igual juntos

`==`, enquanto o operador de atribuição, na linguagem C, é representado por apenas um sinal de igual `=`.

Para exemplificar tal problema, veja o exemplo abaixo:

Erro de lógica usando operador de atribuição

```c
int a = 1;
int b = 2;

// falso
if ( a == b ) {
    printf( "a é igual a b" );
}

// verdadeiro, pois a recebe b e
// o valor de a se torna 2
if ( a = b )
    printf( "a é igual a b" );
}
```

2.1.5 Operador Ternário

O operador ternário, simbolizado pelo caractere `?`, funciona de forma parecida com uma estrutura condicional *if*. Obrigatoriamente, ao utilizar o operador ternário, é necessário que seja gerado algum tipo de retorno. Veja os exemplos a seguir.

Exemplo de uso do operador ternário

```c
/*
 * Arquivo: EstruturasCondicionaisOperadorTernario.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>

int main() {

    int n = 20;
```

```
12
13      // n é par? se sim, retorna 1, caso contrário, retorna 0
14      int v = n % 2 == 0 ? 1 : 0;
15
16      return 0;
17
18  }
```

Código correspondente usando a estrutura condicional if

```
1   /*
2    * Arquivo: EstruturasCondicionaisOperadorTernarioIf.c
3    * Autor: Prof. Dr. David Buzatto
4    */
5
6   #include <stdio.h>
7   #include <stdlib.h>
8
9   int main() {
10
11      int n = 20;
12      int v;
13
14      // n é par? se sim, atribui 1 a v, caso contrário, atribui 0
15      if ( n % 2 == 0 ) {
16          v = 1;
17      } else {
18          v = 0;
19      }
20
21      return 0;
22
23  }
```

Há a possibilidade de se encadear diversos operadores ternários na mesma expressão, entretanto, essa prática pode acarretar em dificuldade de leitura do código, sendo assim:

| Boa Prática

Utilize o operador ternário **apenas** em situações simples e fáceis de serem lidas e interpretadas, evitando assim erros de lógica em seus algoritmos.

| Boa Prática

Evite utilizar mais de um operador ternário em uma expressão, visando sempre facilitar a leitura do seu código.

Veja abaixo como **não** utilizar o operador ternário.

Como não utilizar o operador ternário

```c
/*
 * Arquivo: EstruturasCondicionaisOperadorTernarioNao.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>

int main() {

    int n = 20;

    /* n é par e menor que zero? retorna 1, caso contrário, 0
     * n é ímpar e maior que zero? retorna 1, caso contrário, 0
     */
    int v = n % 2 == 0 ? n < 0 ? 1 : 0 : n > 0 ? 1 : 0;

    // correspondente usando if
    if ( n % 2 == 0 ) {
        if ( n < 0 ) {
            v = 1;
        } else {
            v = 0;
        }
    } else {
        if ( n > 0 ) {
            v = 1;
        } else {
```

```
29              v = 0;
30          }
31      }
32
33      return 0;
34
35  }
```

2.1.6 Cabeçalho `stdbool.h`

Na linguagem C, ao se incluir o cabeçalho `stdbool.h` no preâmbulo do arquivo de código fonte, é possível utilizar o tipo `bool`, que é um tipo lógico capaz de armazenar um valor verdadeiro (`true`) ou falso (`false`). O nome do tipo `bool` vem da palavra *boolean*, que por sua vez, se originou da Lógica de Boole, inventada por George Boole.

Saiba Mais

Quer saber mais sobre George Boole e a Álgebra Booleana? Acesso os seguites links:

- George Boole: **<https://pt.wikipedia.org/wiki/George_Boole>**
- Álgebra Booleana: **<https://brasilescola.uol.com.br/informatica/algebra-booleana.htm>**

Exemplo de uso do cabeçalho stdbool.h

```c
/*
 * Arquivo: EstruturasCondicionaisStdbool.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

/* stdbool.h é o cabeçalho que contém o tipo bool
 * e as palavras chave true e false.
 *
 * o tipo bool não é um tipo nativo da linguagem C
 * como int ou char. sendo assim, é necessário
 * incluir o cabeçalho stdbool caso queira utilizá-lo.
```

```c
 */

int main() {

    // variável do tipo bool
    bool maiorDeIdade;
    int idade;

    printf( "Entre com sua idade: " );
    scanf( "%d", &idade );

    if ( idade < 18 ) {
        // false é o valor que indica falso
        maiorDeIdade = false;
    } else {
        // true é o valor que indica verdadeiro
        maiorDeIdade = true;
    }

    // o if anterior poderia ser substituído por
    maiorDeIdade = idade >= 18;

    // ou
    maiorDeIdade = !( idade < 18 );

    /* maiorDeIdade armazena verdadeiro ou falso, sendo assim
     * pode ser usado diretamente no lugar do teste lógico.
     */
    if ( maiorDeIdade ) {
        printf( "Voce eh maior de idade!" );
    } else {
        printf( "Voce nao eh maior de idade!" );
    }

    return 0;

}
```

É importante reforçar que, como já vimos, na linguagem C, todo valor diferente de zero é interpretado/considerado como verdadeiro, enquanto valores iguais a zero, são considerados falsos. Isso implica que, em C, pode-se usar qualquer tipo de expressão que gere um valor no lugar de um teste lógico. Apesar de válido, evite tal prática!

Boa Prática

Evite utilizar expressões que geram valores como expressões dos testes lógicos. Apesar de válido na linguagem C, isso pode dificultar a leitura do seu código.

Vamos aos exercícios!

2.1.7 Exercícios

Exercício 2.1: Escreva um programa que peça para o usuário fornecer um número inteiro. O programa deve exibir se o número fornecido é par ou ímpar.

Arquivo com a solução: ex2.1.c

Entrada

```
Entre com um numero: 19
```

Saída

```
O numero 19 e impar.
```

Entrada

```
Entre com um numero: 8
```

Saída

```
O numero 8 e par.
```

Exercício 2.2: Escreva um programa que peça para o usuário fornecer dois números inteiros. O programa deve exibir esses dois números em ordem crescente.

Arquivo com a solução: ex2.2.c

Entrada

```
Entre com um numero: 7
Entre com outro numero: 2
```

Saída

```
Ordem crescente: 2 <= 7
```

Entrada

```
Entre com um numero: -2
Entre com outro numero: 9
```

Saída

```
Ordem crescente: -2 <= 9
```

Entrada

```
Entre com um numero: 4
Entre com outro numero: 4
```

Saída

```
Ordem crescente: 4 <= 4
```

Exercício 2.3: Escreva um programa que peça para o usuário fornecer dois números inteiros. O programa deve exibir esses dois números em ordem decrescente.

Arquivo com a solução: ex2.3.c

Entrada

```
Entre com um numero: 7
Entre com outro numero: 2
```

Saída

```
Ordem decrescente: 7 >= 2
```

Entrada

```
Entre com um numero: -30
Entre com outro numero: 20
```

Saída

```
Ordem decrescente: 20 >= -30
```

Entrada

```
Entre com um numero: 4
Entre com outro numero: 4
```

Saída

```
Ordem decrescente: 4 >= 4
```

Exercício 2.4: Escreva um programa que peça para o usuário fornecer três números inteiros. O programa deve exibir esses três números em ordem crescente.

Arquivo com a solução: ex2.4.c

Entrada

```
N1: 5
N2: 1
N3: 9
```

Saída

```
1 <= 5 <= 9
```

Entrada

```
N1: 15
N2: 8
N3: -4
```

Saída

```
-4 <= 8 <= 15
```

Entrada

```
N1: -4
N2: 8
N3: -4
```

Saída

```
-4 <= -4 <= 8
```

Exercício 2.5: Escreva um programa que peça para o usuário fornecer três números inteiros. O programa deve exibir esses três números em ordem decrescente.

Arquivo com a solução: ex2.5.c

Entrada

```
N1: 5
N2: 1
N3: 9
```

Saída

```
9 >= 5 >= 1
```

Entrada

```
N1: 15
N2: 8
N3: -4
```

Saída

```
15 >= 8 >= -4
```

Entrada

```
N1: -4
N2: 8
N3: -4
```

Saída

```
8 >= -4 >= -4
```

Exercício 2.6: Escreva um programa que peça para o usuário fornecer um número decimal. Se esse número for maior que 20, imprimir sua metade, caso contrário, imprimir seu triplo. Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: ex2.6.c

Entrada

```
Entre com um valor: 33.5
```

Saída

```
A metade de 33.50 e 16.75
```

Entrada

```
Entre com um valor: 9.5
```

Saída

```
O triplo de 9.50 e 28.50
```

Exercício 2.7: Escreva um programa que peça para o usuário fornecer dois números decimais. O programa deve somar esses dois números e se essa soma for maior que 10, os dois números devem ser exibidos. Caso contrário, a subtração dos dois números deve ser mostrada. Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: ex2.7.c

Entrada

```
Entre com um numero: 7
Entre com outro numero: 8.5
```

Saída

```
Os numeros fornecidos foram 7.00 e 8.50
```

Entrada

```
Entre com um numero: 3
Entre com outro numero: 2
```

Saída

```
A subtracao entre 3.00 e 2.00 e igual a 1.00
```

Exercício 2.8: Escreva um programa que peça para o usuário fornecer três números decimais e escrever a soma dos dois maiores. Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: ex2.8.c

Entrada

```
N1: 4
N2: 2
N3: 9
```

Saída

```
A soma dos dois numeros maiores fornecidos e 13.00
```

Entrada

```
N1: -1
N2: 7
N3: -2
```

Saída

```
A soma dos dois numeros maiores fornecidos e 6.00
```

Entrada

```
N1: 7
N2: 3
N3: 7
```

Saída

```
A soma dos dois numeros maiores fornecidos e 14.00
```

Exercício 2.9: Escreva um programa que peça para o usuário fornecer a quantidade de lados de um polígono regular (inteiro) e a medida do lado (decimal). Calcular e imprimir o seguinte:

- Se o número de lados for igual a 3, escrever: TRIANGULO (sem acento) e o valor do seu perímetro;
- Se o número de lados for igual a 4, escrever: QUADRADO e o valor da sua área;

- Se o número de lados for igual a 5, escrever: PENTAGONO (sem acento);
- Em qualquer outra situação, escrever: Poligono nao identificado (sem acentos).

Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: ex2.9.c

Entrada

```
Entre com a quantidade de lados: 3
Entre com a medida do lado: 7.5
```

Saída

```
TRIANGULO de perimetro 22.50
```

Entrada

```
Entre com a quantidade de lados: 4
Entre com a medida do lado: 5.5
```

Saída

```
QUADRADO de area 30.25
```

Entrada

```
Entre com a quantidade de lados: 5
Entre com a medida do lado: 2.5
```

Saída

```
PENTAGONO
```

Entrada

```
Entre com a quantidade de lados: 7
Entre com a medida do lado: 3.75
```

Saída

```
Poligono nao identificado
```

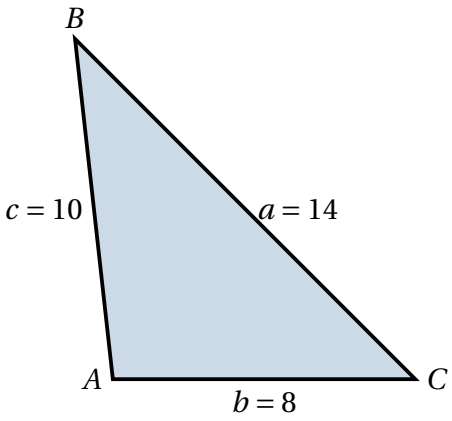
Exercício 2.10: Escreva um programa que leia as medidas dos lados de um triângulo (decimais) e escreva se ele é EQUILATERO, ISOSCELES ou ESCALENO (sem acentos). Observação:

- Triângulo equilátero: Possui 3 lados congruentes (mesma medida);
- Triângulo isósceles: Possui 2 lados congruentes e um diferente;
- Triângulo escaleno: Possui 3 lados diferentes;
- Caso os valores fornecidos não representem um triângulo válido, apresente uma mensagem de erro.

A condição de existência de um triângulo, para os lados a, b e c, é a seguinte:

$|a-b| < c < a+b \bigwedge$
$|a-c| < b < a+c \bigwedge$
$|b-c| < a < b+c = VERDADEIRO$

Exemplo para $a = 14, b = 8, c = 10$:



$|14-8| < 10 < 14+8 \bigwedge$
$|14-10| < 8 < 14+10 \bigwedge$
$|8-10| < 14 < 8+10 = VERDADEIRO$

Arquivo com a solução: ex2.10.c

Entrada

```
a: 5
b: 5
c: 5
```

Saída

```
Triangulo EQUILATERO
```

Entrada

```
a: 6.5
b: 9
c: 6.5
```

Saída

```
Triangulo ISOSCELES
```

Entrada

```
a: 6.5
b: 7
c: 3.5
```

Saída

```
Triangulo ESCALENO
```

Entrada

```
a: 15.8
b: 5.5
c: 3.5
```

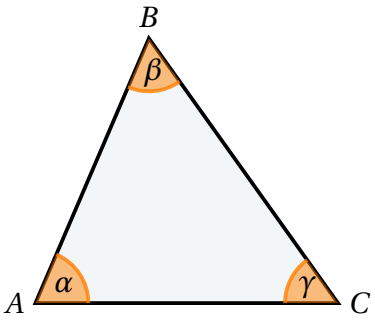
Saída

```
As medidas fornecidas dos lados nao representam um triangulo
valido!
```

Exercício 2.11: Escreva um programa que leia o valor de 3 ângulos internos (α, β e γ) (decimais) de um triângulo e escreva se o triângulo é ACUTANGULO, RETANGULO ou OBTUSANGULO (sem acentos). Observação:

- Triângulo retângulo: possui um ângulo reto (90 graus);
- Triângulo obtusângulo: possui um ângulo obtuso (ângulo maior que 90 graus);
- Triângulo acutângulo: possui 3 ângulos agudos (ângulo menor que 90 graus);
- Caso os valores fornecidos não representem um triângulo válido, apresente uma mensagem de erro.

Para ser um triângulo, a soma dos três ângulos internos deve ser igual a 180 graus, ou seja, $\alpha + \beta + \gamma = 180°$.



Arquivo com a solução: ex2.11.c

Entrada

```
alfa: 90
beta: 60
gama: 30
```

Saída

```
Triangulo RETANGULO
```

Entrada

```
alfa: 70
beta: 70
gama: 40
```

Saída

```
Triangulo ACUTANGULO
```

Entrada

```
alfa: 30
beta: 120
gama: 30
```

Saída

```
Triangulo OBTUSANGULO
```

Entrada

```
alfa: 90
beta: 60
gama: 60
```

Saída

```
As medidas fornecidas dos angulos nao representam um triangulo
valido!
```

Exercício 2.12: Escreva um programa que leia a idade de dois homens e duas mulheres, todos valores inteiros. Calcule e escreva a soma das idades do homem mais velho com a mulher mais nova e o produto das idades do homem mais novo com a mulher mais velha.

Arquivo com a solução: ex2.12.c

Entrada

```
Idade Homem 1: 20
Idade Homem 2: 25
Idade Mulher 1: 40
Idade Mulher 2: 15
```

Saída

```
Idade homem mais velho + idade mulher mais nova: 40
Idade homem mais novo * idade mulher mais velha: 800
```

Exercício 2.13: Escreva um programa que leia as notas das duas avaliações normais e a nota da avaliação optativa. Caso o aluno não tenha feito a optativa deve ser fornecido um valor negativo. Calcular a média do semestre considerando que a prova optativa substitui a nota mais baixa entre as duas primeiras avaliações, caso, é claro, ela seja maior que uma das duas notas. Escrever a média e uma mensagem que indique se o aluno foi aprovado, reprovado ou está em exame. Formate a saída dos números decimais usando 2 casas de precisão. Considere que se M é a média:

- Aprovado: $M \geq 6,0$;
- Exame: $4,0 \leq M < 6,0$:
- Reprovado: $M < 4,0$

Arquivo com a solução: ex2.13.c

Entrada

```
Nota Av. 1: 6.5
Nota Av. 2: 7.5
Nota Optativa: -1
```

Saída

```
Media: 7.00
Aprovado!
```

Entrada

```
Nota Av. 1: 3
Nota Av. 2: 4
Nota Optativa: 6
```

Saída

```
Media: 5.00
Exame.
```

Entrada

```
Nota Av. 1: 5
Nota Av. 2: 1
Nota Optativa: 2
```

Saída

```
Media: 3.50
Reprovado...
```

Exercício 2.14: Escreva um programa que peça para o usuário fornecer seu peso em quilogramas e sua altura em metros, ambos números decimais. O programa deve calcular o IMC (Índice de Massa Corpórea) do usuário e no final deve exibir, além

do índice, qual a situação do usuário na forma de uma mensagem, sem acentos, baseando-se nas seguintes regras:

- $IMC < 17,0$: Voce esta muito abaixo do peso ideal!
- $17,0 \leq IMC < 18,5$: Voce esta abaixo do peso ideal!
- $18,5 \leq IMC < 25,0$: Parabens! Voce esta em seu peso normal!
- $25,0 \leq IMC < 30,0$: Atencao, voce esta acima de seu peso (sobrepeso)!
- $30,0 \leq IMC < 35,0$: Cuidado! Obesidade grau I!
- $35,0 \leq IMC < 40,0$: Cuidado! Obesidade grau II!
- $IMC \geq 40,0$: Muito cuidado!!! Obesidade grau III!

Para o cálculo do IMC utilize:

- $IMC = \frac{p}{h^2}$
- Onde:
  - IMC é o índice de massa corpórea;
  - p é o valor do peso (na verdade, massa) em quilogramas;
  - h é o valor da altura em metros.

Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: ex2.14.c

Entrada

```
Entre com seu peso em quilogramas: 82
Entre com sua altura em metros: 1.8
```

Saída

```
IMC: 25.31
Atencao, voce esta acima de seu peso (sobrepeso)!
```

Entrada

```
Entre com seu peso em quilogramas: 50
Entre com sua altura em metros: 1.7
```

Saída

```
IMC: 17.30
Voce esta abaixo do peso ideal!
```

Entrada

```
Entre com seu peso em quilogramas: 120
Entre com sua altura em metros: 1.82
```

Saída

```
IMC: 36.23
Cuidado! Obesidade grau II!
```

Exercício 2.15: Escreva um programa que peça para o usuário fornecer sua idade em anos e que exiba a classe eleitoral, sem acentos, desse usuário, baseando-se nas seguintes regras:

- Idade abaixo de 16 anos: Nao eleitor;
- Idade maior ou igual a 18 anos e menor ou igual a 65 anos: Eleitor obrigatorio;
- Idade maior ou igual a 16 anos e menor que 18 anos ou maior que 65 anos: Eleitor facultativo.

Arquivo com a solução: ex2.15.c

Entrada

```
Entre com sua idade: 18
```

Saída

```
Eleitor obrigatorio.
```

Entrada

```
Entre com sua idade: 29
```

Saída

```
Eleitor obrigatorio.
```

Entrada

```
Entre com sua idade: 15
```

Saída

```
Nao eleitor.
```

Entrada

```
Entre com sua idade: 17
```

Saída

```
Eleitor facultativo.
```

Entrada

```
Entre com sua idade: 70
```

Saída

```
Eleitor facultativo.
```

Exercício 2.16: Escreva um programa que peça para o usuário fornecer um número no intervalo de 1, inclusive, e 3999, inclusive. Caso o valor fornecido esteja fora desse intervalo, o programa deve avisar o usuário e terminar. Caso contrário, o programa deve exibir o número romano correspondente ao número arábico fornecido. Lembrando que:

- 1 = I;
- 5 = V;
- 10 = X;
- 50 = L;
- 100 = C;
- 500 = D;
- 1000 = M;

Arquivo com a solução: ex2.16.c

Entrada

```
Entre com um numero entre 1 e 3999: 4
```

Saída

```
4 = IV
```

Entrada

```
Entre com um numero entre 1 e 3999: 9
```

Saída

```
9 = IX
```

Entrada

```
Entre com um numero entre 1 e 3999: 27
```

Saída

```
27 = XXVII
```

Entrada

```
Entre com um numero entre 1 e 3999: 251
```

Saída

```
251 = CCLI
```

Entrada

```
Entre com um numero entre 1 e 3999: 2796
```

Saída

```
2796 = MMDCCXCVI
```

Entrada

```
Entre com um numero entre 1 e 3999: 5000
```

Saída

```
Numero incorreto!
```

2.2 Estrutura Condicional *switch* (escolha)

2.2.1 Exemplo e Diagrama de Fluxo da Estrutura Condicional *switch*

Estrutura do *switch* - Verifique a Figura 2.4

```
1  (1) // antes
2  switch ( valor ) {
3      case valor1: (2)
4          (3)
5          break;
6      case valor2: (4)
7          (5)
8          break;
9      case valor3: (6)
10         (7)
11         break;
12     default:
13         (8)
14         break;
15 }
16 (9) // depois
```

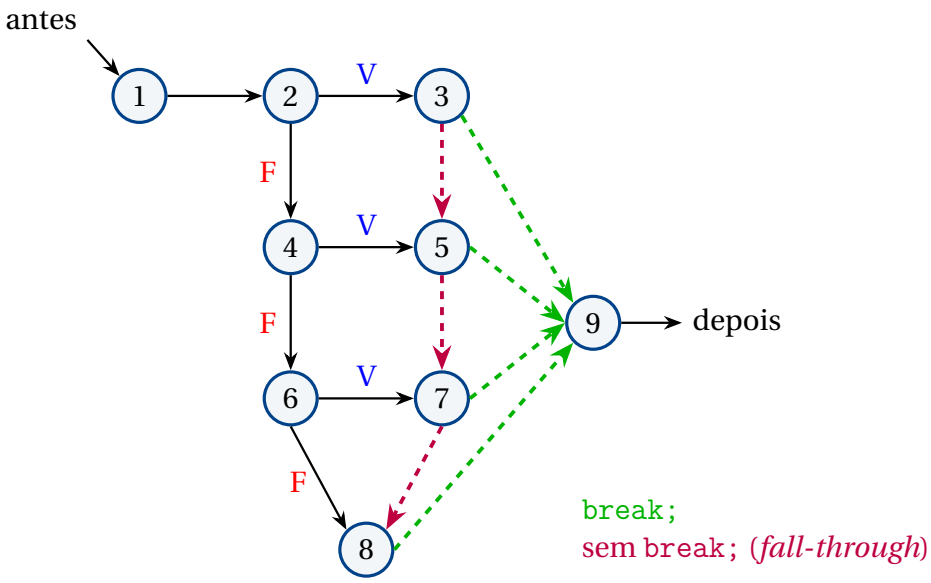


Figura 2.4: Fluxo de execução da estrutura condicional *switch*

A outra estrutura condicional que a linguagem de programação C suporta é a *switch*

(escolha). A diferença entre ela e o *if* é que na estrutura *switch* usa-se um valor, normalmente contido em uma variável, para ser comparado em cada cláusula `case` contida dentro do bloco. Caso o valor da variável passada seja igual ao valor esperado por algum dos *cases*, o bloco associado ao `case` em questão será executado. É necessário que se use a instrução `break` para que a execução de um bloco pare antes do próximo `case` (ou `default`), caso contrário, as próximas linhas de código serão executadas até encontrar um `break` ou até terminar o bloco do *switch*. Esse comportamento é chamado de *fall-through*. Outro detalhe é que a ordem dos *cases* não importa e que a cláusula `default` é opcional, além de ser usada para casar com qualquer valor diferente dos valores esperados por todos os cases especificados.

Vamos aos exercícios!

2.2.2 Exercícios

Exercício 2.17: Escreva um programa que peça para o usuário fornecer um número inteiro. Use um `switch` para verificar se o número é igual a 2, ou 4, ou 6, ou 8. Caso seja um desses números, exiba uma mensagem informando ao usuário o número que foi digitado. Caso não seja nenhum dos números esperados, informe o usuário que o valor inserido é inválido.

Arquivo com a solução: ex2.17.c

Entrada

```
Entre com um valor inteiro: 6
```

Saída

```
O valor fornecido foi 6.
```

Entrada

```
Entre com um valor inteiro: 7
```

Saída

```
Valor invalido.
```

Exercício 2.18: Escreva um programa que peça para o usuário fornecer dois números decimais. Após a inserção de tais números, o programa deve mostrar ao usuário um menu, onde ele poderá escolher entre as quatro operações básicas (adição, subtração,

multiplicação e divisão). Fazer a leitura dessa opção como um caractere (tipo `char`). Dependendo da operação escolhida, o programa deve executar o cálculo correspondente e exibir ao usuário o resultado. Caso o usuário forneça uma opção inválida, o programa deve exibir uma mensagem dizendo que a opção é inválida e deve terminar sua execução. Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: ex2.18.c

Entrada

```
N1: 28
N2: 8
Escolha uma operacao de acordo com o menu:
    +) Adicao;
    -) Subtracao;
    *) Multiplicacao;
    /) Divisao.
Operacao: +
```

Saída

```
28.00 + 8.00 = 36.00
```

Entrada

```
N1: 16
N2: 3
Escolha uma operacao de acordo com o menu:
    +) Adicao;
    -) Subtracao;
    *) Multiplicacao;
    /) Divisao.
Operacao: /
```

Saída

```
16.00 / 3.00 = 5.33
```

Entrada

```
N1: 13
N2: 76
Escolha uma operacao de acordo com o menu:
    +) Adicao;
    -) Subtracao;
    *) Multiplicacao;
    /) Divisao.
Operacao: x
```

Saída

```
Opcao invalida!
```

Exercício 2.19: Escreva um programa que exiba um menu ao usuário, onde ele poderá escolher entre converter um valor em graus Celsius para graus Fahrenheit, ou então converter um valor em graus Fahrenheit para graus Celsius. Os valores das temperaturas são valores decimais. Caso o usuário forneça uma opção inválida, o programa deve exibir uma mensagem dizendo que a opção é inválida e deve terminar sua execução. Lembrando que:

- $C = \frac{F-32}{1,8}$
- $F = 1,8C + 32$
- Onde:
  - C é a temperatura em graus Celsius;
  - F é a temperatura em graus Fahrenheit.

Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: ex2.19.c

Entrada

```
Escolha uma operacao de acordo com o menu:
    C) Celsius -> Fahrenheit;
    F) Fahrenheit -> Celsius.
Opcao: C
Entre com a temperatura em graus Celsius: 38.5
```

Saída

```
38.50 graus Celsius correspondem a 101.30 graus Fahrenheit
```

Entrada

```
Escolha uma operacao de acordo com o menu:
    C) Celsius -> Fahrenheit;
    F) Fahrenheit -> Celsius.
Opcao: F
Entre com a temperatura em graus Fahrenheit: 125.7
```

Saída

```
125.70 graus Fahrenheit correspondem a 52.06 graus Celsius
```

Entrada

```
Escolha uma operação de acordo com o menu:
    C) Celsius -> Fahrenheit;
    F) Fahrenheit -> Celsius.
Opcao: x
```

Saída

```
Opcao invalida!
```

CAPÍTULO 3

ESTRUTURAS DE REPETIÇÃO

"Bem. E daí? Daí, nada. Quanto a mim, autor de uma vida, me dou mal com a repetição: a rotina me afasta de minhas possíveis novidades".

Clarice Lispector

S estruturas de repetição permitem que seja possível escrever trechos de código que serão executados repetidamente, a partir da satisfação de uma determinada condição. Neste Capítulo serão apresentadas as estruturas de repetição que existem na linguagem de programação C.

3.1 Estrutura de Repetição *for*

A primeira estrutura de repetição que veremos é a estrutura *for*. O *for* é em geral usado quando sabemos previamente quantas vezes o bloco de código associado à estrutura será executado. Isso não é uma regra absoluta, mas normalmente é assim que é aplicado. Essa quantidade de vezes que o bloco executará, em geral, é relacionada à uma quantidade inteira ou ao tamanho de alguma estrutura de dados. O *for* é construído na maioria das vezes usando apenas uma variável de controle que existirá somente dentro do bloco de código, mas podemos, dependendo da situação, permitir

que essa variável seja enxergada também fora do bloco do *for*, além de podermos ter mais de uma variável de controle. Novamente, o uso de apenas uma variável que tenha o escopo no bloco do *for* não é uma regra absoluta, mas é o usual. Outro padrão adotado é que as variáveis que controlam o for são normalmente nomeadas de `i`, `j`, `k`, ..., `n`, pois têm base matemática e histórica. Do lado da matemática, devido a uma boa parte da programação se dar na tradução de problemas matemáticos e na solução dos mesmos e do lado histórico por causa da linguagem de programação FORTRAN, onde variáveis do tipo inteiro devem iniciar o nome dos identificadores usando as letras de `I` a `N`. Em alguns exercícios veremos esse paralelo da representação de equações matemáticas em linguagem de programação.

3.1.1 Operadores Unários de Incremento e Decremento

Usualmente a cada iteração (repetição) do *for* nós precisaremos atualizar o valor da variável de controle. Isso se dá normalmente incrementando (aumentando) ou decrementando (diminuindo) o valor da mesma em uma unidade. Mais uma vez, isso não é uma regra imutável, mas na grande maioria das vezes é isso que precisará ser feito. Sendo assim, existem dois operadores unários (aplicados a apenas um operando) que são usados. Após a apresentação da sintaxe e do diagrama de fluxo do *for*, há uma listagem de código mostrando o uso dos mesmos. Esses operadores podem ser vistos na Tabela 3.1.

Operador	Significado
++	Incremento (soma um) pré e pós fixado
–	Decremento (subtrai um) pré e pós fixado

Tabela 3.1: Operadores unários de incremento e decremento

Boas Práticas

- Nomeie as variáveis de controle das estruturas de repetição *for* como `i`, `j`, `k`, ..., `n`;
- Sempre que possível utilize apenas uma variável para o controle da execução do *for*;
- Sempre utilize chaves para delimitar o bloco de código do *for*, mesmo que ele contenha apenas uma linha de código;
- Realize o teste de mesa para verificar o que o seu algoritmo com repetição está fazendo de modo a encontrar problemas.

| Boas Práticas

- Atualize o valor das variáveis de controle preferencialmente na seção de passo;
- Procure utilizar as três seções do for, mesmo que as três sejam opcionais.

| Atenção!

As seções de inicialização, teste e passo do *for* são **separadas por ponto e vírgula**.

3.1.2 Exemplo e Diagrama de Fluxo da Estrutura de Repetição *for* (para)

Estrutura do *for* - Verifique a Figura 3.1

```
1 // antes
2 (1)
3              (2)        (3)     (4)
4 for ( inicialização; teste; passo ) {
5     (5)
6 }
7 (6)
8 // depois
```

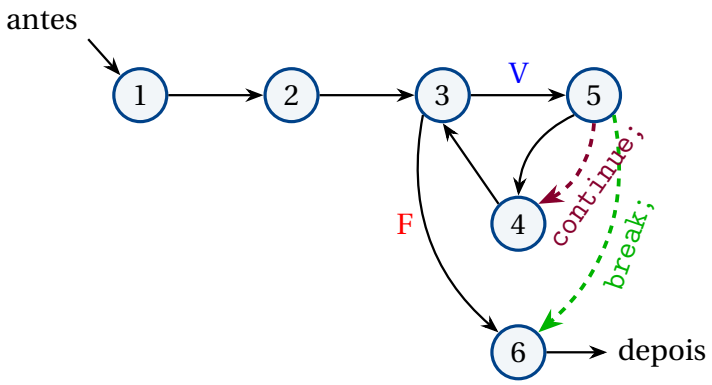


Figura 3.1: Fluxo de execução da estrutura de repetição *for*

Trecho de código exemplificando o funcionamento do operador unário de incremento

```c
/*
 * Arquivo: EstruturasDeRepeticaoOperadoresIncrDecr.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>

int main() {

    /* o funcionamento do operador unário de decremento
     * é análogo ao operador unário de incremento.
     */

    int i = 0;

    printf( "%d\n", i );   // imprime 0

    i++;  // incremento pós-fixado
    printf( "%d\n", i );   // imprime 1

    ++i;  // incremento pré-fixado
    printf( "%d\n", i );   // imprime 2

    /* incremento pós-fixado em uma expressão
     * usa o valor, depois incrementa.
     */
    printf( "%d\n", i++ ); // imprime 2
    printf( "%d\n", i );   // imprime 3

    /* incremento pré-fixado em uma expressão
     * incrementa depois usa o valor.
     */
    printf( "%d\n", ++i ); // imprime 4
    printf( "%d", i );   // imprime 4

    return 0;

}
```

⚠ | Atenção!

Tome cuidado com a geração de laços/*loops* infinitos, ou seja, laços que nunca terminam. Preste sempre atenção no estado das variáveis que controlam as estruturas de repetição. Caso ocorra algum *loop* infinito durante a execução do seu programa, investigue seu algoritmo e o estado da ou das variáveis de controle das suas estruturas de repetição.

Vamos aos exercícios!

3.1.3 Exercícios

Exercício 3.1: Escreva um programa que imprima os números inteiros de 0, inclusive, a 20, inclusive, usando a estrutura de repetição `for`.

Arquivo com a solução: ex3.1.c

Saída

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Exercício 3.2: Escreva um programa que imprima os números inteiros pares que estão no intervalo entre 0, inclusive, e 50, inclusive, usando a estrutura de repetição `for`.

Arquivo com a solução: ex3.2.c

Saída

```
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44
46 48 50
```

Exercício 3.3: Escreva um programa que imprima os números inteiros de 20, inclusive, a 0, inclusive, usando a estrutura de repetição `for`

Arquivo com a solução: ex3.3.c

Saída

```
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

Exercício 3.4: Escreva um programa que apresente o quadrado dos números inteiros de 15, inclusive, a 30, inclusive, usando a estrutura de repetição `for`

Arquivo com a solução: ex3.4.c

Saída

```
225 256 289 324 361 400 441 484 529 576 625 676 729 784 841 900
```

Exercício 3.5: Escreva um programa que peça para o usuário entrar com um número inteiro maior ou igual a 0. Se um valor incorreto for digitado, o programa deve avisar o usuário imprimindo na tela a mensagem "Valor incorreto (negativo)" e terminar. Caso o número seja correto, o programa deve exibir os números de 0 ao número digitado.

Arquivo com a solução: ex3.5.c

Entrada

```
Forneca um numero maior ou igual a zero: 7
```

Saída

```
0 1 2 3 4 5 6 7
```

Entrada

```
Forneca um numero maior ou igual a zero: -5
```

Saída

```
Valor incorreto (negativo)
```

Exercício 3.6: Escreva um programa que peça para o usuário entrar com um número inteiro maior ou igual a 0. Se um valor incorreto for digitado, o programa deve avisar o usuário imprimindo na tela a mensagem "Valor incorreto (negativo)" e terminar. Caso o número seja correto, o programa deve exibir os números do número digitado até 0.

Arquivo com a solução: ex3.6.c

Entrada

```
Forneca um numero maior ou igual a zero: 7
```

Saída

```
7 6 5 4 3 2 1 0
```

Entrada

```
Forneca um numero maior ou igual a zero: -10
```

Saída

```
Valor incorreto (negativo)
```

Exercício 3.7: Escreva um programa que peça para o usuário entrar com um número inteiro menor ou igual a 0. Se um valor incorreto for digitado, o programa deve avisar o usuário imprimindo na tela a mensagem "Valor incorreto (positivo)" e terminar. Caso o número seja correto, o programa deve exibir os números do número digitado até 0.

Arquivo com a solução: ex3.7.c

Entrada

```
Forneca um numero menor ou igual a zero: -10
```

Saída

```
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0
```

Entrada

```
Forneca um numero menor ou igual a zero: 20
```

Saída

```
Valor incorreto (positivo)
```

Exercício 3.8: Escreva um programa que peça para o usuário entrar com um número inteiro menor ou igual a 0. Se um valor incorreto for digitado, o programa deve avisar o usuário imprimindo na tela a mensagem "Valor incorreto (positivo)" e terminar. Caso o número seja correto, o programa deve exibir os números de 0 ao número digitado.

Arquivo com a solução: ex3.8.c

Entrada

```
Forneca um numero menor ou igual a zero: -9
```

Saída

```
0 -1 -2 -3 -4 -5 -6 -7 -8 -9
```

Entrada

```
Forneca um numero menor ou igual a zero: 15
```

Saída

```
Valor incorreto (positivo)
```

Exercício 3.9: Escreva um programa que peça para o usuário fornecer um número inteiro. O programa deve exibir a tabuada de 0 a 10 desse número.

Arquivo com a solução: ex3.9.c

Entrada

```
Tabuada do Numero: 5
```

Saída

```
5 x 0 = 0
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

Exercício 3.10: Escreva um programa que apresente se cada número inteiro no intervalo entre 45, inclusive, e 60, inclusive, é divisível ou não por 4. Utilize a estrutura de repetição `for`.

Arquivo com a solução: ex3.10.c

Saída

```
45: indivisivel
46: indivisivel
47: indivisivel
48: divisivel
49: indivisivel
50: indivisivel
51: indivisivel
52: divisivel
53: indivisivel
54: indivisivel
55: indivisivel
56: divisivel
57: indivisivel
58: indivisivel
59: indivisivel
60: divisivel
```

Exercício 3.11: Escreva um programa que peça para o usuário fornecer dois números inteiros. Se o primeiro número for menor ou igual ao segundo, o programa deve exibir todos os números no intervalo entre os números digitados em ordem crescente. Caso o primeiro número seja maior que o segundo, o programa deve exibir todos os números no intervalo entre os números digitados em ordem decrescente.

Arquivo com a solução: ex3.11.c

Entrada

```
N1: 2
N2: 10
```

Saída

```
2 3 4 5 6 7 8 9 10
```

Entrada

```
N1: 30
N2: 20
```

Saída

```
30 29 28 27 26 25 24 23 22 21 20
```

Exercício 3.12: Escreva um programa que peça para o usuário fornecer dois números inteiros. O programa deve contar e exibir a quantidade de números pares que existem no intervalo compreendido entre os dois números informados, considerando esses dois números. Fique atento à ordem de entrada dos números.

Arquivo com a solução: ex3.12.c

Entrada

```
N1: 5
N2: 100
```

Saída

```
Numeros pares entre 5 e 100: 48
```

Entrada

```
N1: 20
N2: -30
```

Saída

```
Numeros pares entre -30 e 20: 26
```

Exercício 3.13: Escreva um programa que conte quantos números inteiros múltiplos de 2, múltiplos de 3 e múltiplos de 4 existem no intervalo de dois números inteiros fornecidos pelo usuário. Esses contadores devem ser exibidos no final. Fique atento à ordem de entrada dos números.

Arquivo com a solução: ex3.13.c

Entrada

```
N1: -50
N2: 200
```

Saída

```
Multiplos de 2: 126
Multiplos de 3: 83
Multiplos de 4: 63
```

Entrada

```
N1: 50
N2: -100
```

Saída

```
Multiplos de 2: 76
Multiplos de 3: 50
Multiplos de 4: 38
```

Exercício 3.14: Escreva um programa que calcule o somatório dos valores compreendidos entre dois números inteiros fornecidos pelo usuário, incluindo tais números no cálculo. Fique atento à ordem de entrada dos números.

Arquivo com a solução: ex3.14.c

Entrada

```
N1: -5
N2: 50
```

Saída

```
Somatorio entre -5 e 50: 1260
```

Entrada

```
N1: 80
N2: -10
```

Saída

```
Somatorio entre -10 e 80: 3185
```

Exercício 3.15: Escreva um programa que peça para o usuário fornecer um número inteiro positivo e que calcule o fatorial desse número. Caso o número seja negativo, o programa deve avisar o usuário, usando a mensagem "Nao ha fatorial de numero negativo." (sem acentos) e terminar. Caso contrário o programa deve calcular o fatorial do número digitado e exibi-lo. Lembrando que:

- $n! = \prod_{i=1}^{n} i,\ n \in \mathbb{N}^*$
- Onde:
  - $n!$ é o fatorial de n.

Arquivo com a solução: ex3.15.c

Entrada

```
Numero: 5
```

Saída

```
5! = 120
```

Entrada

```
Numero: -10
```

Saída

```
Nao ha fatorial de numero negativo.
```

Exercício 3.16: Escreva um programa que exiba os vinte primeiros termos da série de Fibonacci. A série de Fibonacci inicia com 1 e 1, sendo os próximos termos gerados pela soma dos dois últimos termos. Os nove primeiros termos da série de Fibonacci são: 1 1 2 3 5 8 13 21 34. Obs: O primeiro termo tem posição 0, o segundo termo tem posição 1, o terceiro termo tem posição 2 e assim por diante.

Arquivo com a solução: ex3.16.c

Saída

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
6765
```

Exercício 3.17: Escreva um programa que peça ao usuário o termo da série de Fibonnaci que ele quer que seja obtido e que então exiba esse termo.

Arquivo com a solução: ex3.17.c

Entrada

```
Termo desejado: 0
```

Saída

```
Fibonacci de 0 e 1
```

Entrada

```
Termo desejado: 1
```

Saída

```
Fibonacci de 1 e 1
```

Entrada

```
Termo desejado: 5
```

Saída

```
Fibonacci de 5 e 8
```

Entrada

```
Termo desejado: 15
```

Saída

```
Fibonacci de 15 e 987
```

Exercício 3.18: Escreva um programa que exiba o seguinte desenho usando, obrigatoriamente, a estrutura de repetição `for`.

Arquivo com a solução: ex3.18.c

Saída

```
*
**
***
****
*****
```

Exercício 3.19: Escreva um programa que exiba o seguinte desenho usando, obrigatoriamente, a estrutura de repetição `for`.

Arquivo com a solução: ex3.19.c

Saída

```
*
**
***
****
*****
****
***
**
*
```

Exercício 3.20: Escreva um programa que exiba o seguinte desenho usando, obrigatoriamente, a estrutura de repetição `for`. Os asteriscos quase pretos indicam espaços.

Arquivo com a solução: ex3.20.c

Saída

```
*
**
***
****
*****

*****
****
***
**
*

****
***
**
*
*****

*****
****
***
**
*****
```

Exercício 3.21: Escreva um programa que leia uma altura em inteiro e imprima um triângulo de asteriscos, baseado nessa altura. Uma altura negativa deve resultar em um triângulo de cabeça para baixo. Uma altura igual a zero não produzirá um triângulo. Os asteriscos quase pretos indicam espaços.

Arquivo com a solução: ex3.21.c

Entrada

```
Altura: 3
```

Saída

```
**
*
*****
```

Entrada

```
Altura: 11
```

Saída

```
**********
*********
********
*******
******
*****
****
***
**
*
```

Entrada

```
Altura: -5
```

Saída

```
*********
*******
*****
***
*
```

Exercício 3.22: Escreva um programa que peça para o usuário fornecer 5 números positivos. Caso algum seja menor ou igual a zero, após o usuário fornecer todos os números, o programa deve avisar o usuário e terminar com a mensagem “Forneca apenas numeros positivos.”. Os valores fornecidos devem ser usados para desenhar um gráfico de barras usando o símbolo asterisco. Os asteriscos quase pretos indicam espaços.

Arquivo com a solução: ex3.22.c

Entrada

```
N1: 3
N2: 5
N3: 10
N4: 2
N5: 6
```

Saída

```
0010*****
0009*****
0008*****
0007*****
0006*******
0005*******
0004*******
0003******
0002*****
0001*****
```

Entrada

```
N1: 4
N2: 8
N3: 0
N4: -7
N5: 2
```

Saída

```
Forneca apenas numeros positivos.
```

Dica

Para formatar um valor inteiro usando zeros à esquerda para preenchimento, use o especificador de formato `"%0nd"`, onde `n` é a quantidade de casas que o valor e os zeros ocuparão. Por exemplo, uma variável que contém o valor 15, ao ser formatada usando o especificador de formato `"%04d"`, resultará em 0015, ou seja, o número inteiro 15, precedido de dois zeros, ocupando quatro casas.

Veja o código abaixo:

```
1 int n = 15;
2 printf( "%04d", n );     // imprime 0015
```

Exercício 3.23: Escreva um programa para ler as notas de 10 alunos de uma turma e calcular e exibir a média aritmética destas notas. Armazene a média em uma variável. As notas são números decimais. Utilize a estrutura de repetição `for` para coletar as notas.

Arquivo com a solução: ex3.23.c

Entrada

```
Forneca a nota de 10 alunos:
Nota 01: 6
Nota 02: 8
Nota 03: 9
Nota 04: 8.75
Nota 05: 7
Nota 06: 5
Nota 07: 6
Nota 08: 7.5
Nota 09: 8
Nota 10: 9
```

Saída

```
A media aritmetica das dez notas e: 7.43
```

3.2 Estruturas de Repetição *while* (enquanto) e *do... while* (faça ... enquanto)

Assim como o *for*, o *while* e o *do... while* são usados para realizar iterações, mas a diferença de aplicação é que enquanto no *for* sabemos quantas vezes ele executará com base em uma quantidade, no *while* e no *do... while* nós não temos certeza quantas vezes os laços executarão. Mais uma vez, essa regra não é absoluta, pois podemos fazer a mesma coisa com as três estruturas de repetição, "simulando" uma

na outra. Há professores que ensinam as três, os modos de uso, mas depois pedem para os alunos escreverem programas usando *while* onde se deveria usar o *for* usualmente, além de outas combinações, mas eu acho isso uma perda de tempo e pode criar confusão, ainda mais em um momento que o aprendizado fatalmente começa a ter uma certa dificuldade. A diferença entre as estruturas *while* e *do... while* é que no *while* o teste que será feito ocorre antes da execução do bloco de código associado, ou seja, caso o teste gere um valor falso na primeira vez em que for executado, o bloco de código não executará. No *do... while* é garantido que o bloco de código executa pelo menos uma vez, pois o teste ocorre após a execução do bloco de código.

Boa Prática

Sempre utilize a estrutura de repetição apropriada para o problema que está sendo resolvido.

Dicas

- Se a quantidade de iterações é sabida, seja de forma fixa, ou baseada em um valor numérico ou no tamanho de algo que precisa ser processado, utilize *for*.
- Se a quantidade de iterações não é sabida, utilize *while* ou *do... while* e:
  - Se você precisa que o **bloco de código execute após o teste, use *while***;
  - Se o **bloco de código precisa ser executado antes do teste, use *do... while***.

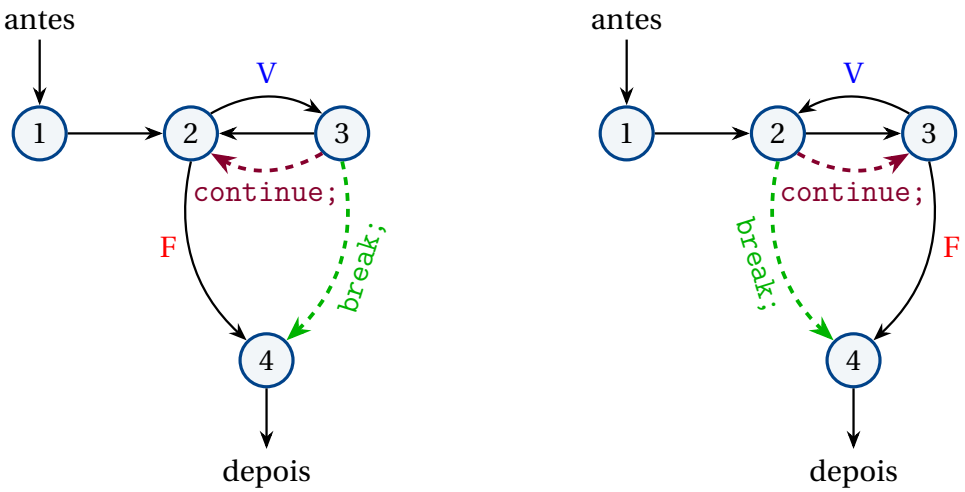
3.2.1 Exemplo e Diagrama de Fluxo da Estrutura de Repetição *while* e *do... while*

Estrutura do *while* - Verifique a Figura 3.2a

```
1 // antes
2 (1)
3           (2)
4 while ( teste ) {
5     (3)
6 }
7 (4) // depois
```

Estrutura do *do...while* - Verifique a Figura 3.2b

```
// antes
(1)
do {
    (2)
            (3)
} while ( teste );
(4) // depois
```



(a) Fluxo do *while* **(b)** Fluxo do *do... while*

Figura 3.2: Fluxos de execução das estruturas de repetição *while* e *do... while*

Vamos aos exercícios!

3.2.2 Exercícios

Exercício 3.24: Escreva um programa que solicite a idade de várias pessoas e imprima o total de pessoas com menos de 21 anos e o total de pessoas com mais de 50 anos. O programa deve terminar e exibir os resultados quando a idade fornecida for negativa.

Arquivo com a solução: ex3.24.c

Entrada

```
Idade da pessoa 01: 10
Idade da pessoa 02: 55
Idade da pessoa 03: -1
```

Saída

```
Total de pessoas menores de 21 anos: 1
Total de pessoas com mais de 50 anos: 1
```

Entrada

```
Idade da pessoa 01: 9
Idade da pessoa 02: 15
Idade da pessoa 03: 57
Idade da pessoa 04: 20
Idade da pessoa 05: 23
Idade da pessoa 06: 19
Idade da pessoa 07: 43
Idade da pessoa 08: 66
Idade da pessoa 09: -10
```

Saída

```
Total de pessoas menores de 21 anos: 4
Total de pessoas com mais de 50 anos: 2
```

Exercício 3.25: Escreva um programa que efetue a leitura sucessiva de valores numéricos decimais e apresente no final o somatório, a média e a quantidade de valores lidos, armazenada como inteiro. O programa deve continuar lendo os números até que seja fornecido um número negativo. Esse número negativo não deve entrar nos cálculos! Formate a saída dos números decimais usando 2 casas de precisão. Caso o número negativo seja o primeiro a ser fornecido, o programa deve exibir um somatório igual a zero, uma média igual a zero e uma quantidade igual a zero.

Arquivo com a solução: ex3.25.c

Entrada

```
Entre com um valor: 4
Entre com um valor: 8
Entre com um valor: -1
```

Saída

```
Somatorio: 12.00
Media: 6.00
Quantidade: 2
```

Entrada

```
Entre com um valor: 5
Entre com um valor: 8
Entre com um valor: 10
Entre com um valor: 15
Entre com um valor: 2
Entre com um valor: 9
Entre com um valor: 3
Entre com um valor: 2
Entre com um valor: -1
```

Saída

```
Somatorio: 54.00
Media: 6.75
Quantidade: 8
```

Entrada

```
Entre com um valor: -5
```

Saída

```
Somatorio: 0.00
Media: 0.00
Quantidade: 0
```

Exercício 3.26: Escreva um programa que efetue a leitura sucessiva de valores numéricos inteiros e apresente no final o menor e o maior número que foram fornecidos. O programa deve continuar lendo os números até que seja fornecido um número negativo, que por sua vez não deve ser apresentado como menor ou maior número. Caso o número negativo seja o primeiro a ser fornecido, o programa deve exibir tanto o menor quanto o maior número como zero. Pense no que precisa ser feito para inicializar apropriadamente os valores das variáveis menor e maior.

Arquivo com a solução: ex3.26.c

Entrada

```
Entre com um valor: 7
Entre com um valor: 15
Entre com um valor: 3
Entre com um valor: 29
Entre com um valor: 2
Entre com um valor: 103
Entre com um valor: 0
Entre com um valor: 34
Entre com um valor: -1
```

Saída

```
Menor numero: 0
Maior numero: 103
```

Entrada

```
Entre com um valor: 5
Entre com um valor: -1
```

Saída

```
Menor numero: 5
Maior numero: 5
```

Entrada

```
Entre com um valor: -5
```

Saída

```
Menor numero: 0
Maior numero: 0
```

Exercício 3.27: Escreva um programa para ler um número indeterminado de dados de pesos de pessoas como números decimais. O último dado, que não entrará nos cálculos, deve ser um valor negativo. O programa deve calcular e imprimir a média aritmética dos pesos das pessoas que possuem mais de 60kg e o peso do indivíduo

mais pesado. Formate a saída dos números decimais usando 2 casas de precisão. Caso o número negativo seja o primeiro a ser fornecido, o programa deve exibir a média e o peso mais pesado ambos como zero.

Arquivo com a solução: ex3.27.c

Entrada

```
Entre com o peso da pessoa 01: 55.8
Entre com o peso da pessoa 02: 102.7
Entre com o peso da pessoa 03: 86.3
Entre com o peso da pessoa 04: -1
```

Saída

```
Media dos pesos acima de 60kg: 94.50
A pessoa mais pesada possui 102.70kg
```

Entrada

```
Entre com o peso da pessoa 01: -1
```

Saída

```
Media dos pesos acima de 60kg: 0.00
A pessoa mais pesada possui 0.00kg
```

Entrada

```
Entre com o peso da pessoa 01: 30.0
Entre com o peso da pessoa 02: -1
```

Saída

```
Media dos pesos acima de 60kg: 0.00
A pessoa mais pesada possui 30.00kg
```

Entrada

```
Entre com o peso da pessoa 01: 90.0
Entre com o peso da pessoa 02: -1
```

Saída

```
Media dos pesos acima de 60kg: 90.00
A pessoa mais pesada possui 90.00kg
```

Exercício 3.28: Escreva um programa para ler o saldo inicial de uma conta bancária, um valor decimal. A seguir ler um número indeterminado de pares de valores indicando respectivamente o tipo da operação (codificado da seguinte forma: 1.Depósito 2.Retirada e 3.Fim) e o valor que será movimentado. Quando for informado para o tipo da operação o código 3, o programa deve ser encerrado e impresso o saldo final da conta com as seguintes mensagens: "Sem saldo." caso o saldo seja zero, "Conta devedora.", se o saldo for negativo ou "Conta preferencial.", se o saldo seja positivo. Caso seja fornecido um tipo incorreto de operação, ou seja, diferente de 1, 2 ou 3, o programa deve exibir ao usuário a mensagem "Operacao invalida." e solicitar novamente a operação. Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: ex3.28.c

Entrada

```
Saldo inicial: 3000
Operacoes:
    1) Deposito;
    2) Saque;
    3) Fim.
Operacao desejada: 1
Valor a depositar: 500
Operacao desejada: 1
Valor a depositar: 300
Operacao desejada: 1
Valor a depositar: 100
Operacao desejada: 2
Valor a sacar: 2555
Operacao desejada: 3
```

Saída

```
Saldo final: R$1345.00
Conta preferencial.
```

Entrada

```
Saldo inicial: 1000
Operacoes:
    1) Deposito;
    2) Saque;
    3) Fim.
Operacao desejada: 2
Valor a sacar: 500
Operacao desejada: 2
Valor a sacar: 300
Operacao desejada: 2
Valor a sacar: 300
Operacao desejada: 3
```

Saída

```
Saldo final: -R$100.00
Conta devedora.
```

Entrada

```
Saldo inicial: 2000
Operacoes:
    1) Deposito;
    2) Saque;
    3) Fim.
Operacao desejada: 2
Valor a sacar: 1500
Operacao desejada: 2
Valor a sacar: 500
Operacao desejada: 3
```

Saída

```
Saldo final: R$0.00
Sem saldo.
```

Exercício 3.29: Escreva um programa para ler 2 valores inteiros e imprimir o resultado da divisão do primeiro pelo segundo. Se o segundo valor informado for zero, deve ser impressa uma mensagem de “Nao existe divisao inteira por zero!” (sem acentos) e lido

um novo valor. Ao final do programa, deve ser impressa a seguinte mensagem: "Voce deseja realizar outro calculo? (S/N): " Se a resposta for 'S' o programa deverá retornar ao começo, repetindo o processo, caso contrário deverá encerrar a sua execução imprimindo quantos cálculos foram feitos.

Arquivo com a solução: ex3.29.c

Entrada

```
N1: 10
N2: 5
Voce deseja realizar outro calculo? (S/N): S
N1: 11
N2: 3
Voce deseja realizar outro calculo? (S/N): S
N1: 15
N2: 0
Entre novamente com N2: 0
Entre novamente com N2: 5
Voce deseja realizar outro calculo? (S/N): N
```

Saída

```
10 / 5 = 2
11 / 3 = 3
Nao existe divisao inteira por zero!
Nao existe divisao inteira por zero!
15 / 5 = 3
```

CAPÍTULO 4

ARRAYS UNIDIMENSIONAIS

"Algoritmos + Estruturas de Dados = Programas".

Niklaus Wirth

S arrays, ou arranjos, algumas vezes também chamados erroneamente de vetores, são estruturas de dados que armazenam um ou vários elementos de um mesmo tipo, ou seja, são estruturas homogêneas. Neste capítulo serão apresentados os arrays unidimensionais, que são estruturas lineares e indexadas a partir da posição 0.

4.1 Exemplos em Linguagem C

Declaração e inicialização de arrays

```c
/*
 * Arquivo: ArraysUnidimensionaisDeclaracaoInicializacao.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>
```

```
8
9  // definição da macro N
10 #define N 10
11
12 int main() {
13
14     // declaração de um array de inteiros de 5 posições
15     int array1[5];
16
17     // declarando um array e inicializando com valores 2
18     // valor inicializado = { 2, 2, 2, 2, 2 }
19     int array2[5] = { 2, 2, 2, 2, 2 };
20
21     // declarando um array e inicializando com valores 3
22     // valor inicializado = { 3, 3, 0, 0, 0 }
23     int array3[5] = { 3, 3 };
24
25     // declarando um array e inicializando com zeros
26     // valor inicializado = { 0, 0, 0, 0, 0 }
27     int array4[5] = { 0 };
28
29     // se o inicializador for usado, o tamanho pode ser omitido
30     // valor inicializado = { 5, 5, 5 }
31     int array5[] = { 5, 5, 5 };
32
33     // declaração de um array de inteiros de N posições
34     // N é uma macro que será expandida ao ser usada
35     int array6[N];
36
37     /* array6 = { 0 };
38      * ou
39      * array6[] = { 0 };
40      *
41      * inválido, pois o inicializador só pode ser usado na declaração
42      */
43
44     // cálculo do tamanho usando o operador sizeof
45     int tamanhoArray1 = (int) ( sizeof( array1 ) / sizeof( array1[0] ) );
46     int tamanhoArray2 = (int) ( sizeof( array2 ) / sizeof( array2[0] ) );
47     int tamanhoArray3 = (int) ( sizeof( array3 ) / sizeof( array3[0] ) );
48     int tamanhoArray4 = (int) ( sizeof( array4 ) / sizeof( array4[0] ) );
49     int tamanhoArray5 = (int) ( sizeof( array5 ) / sizeof( array5[0] ) );
```

```c

    for ( int i = 0; i < tamanhoArray1; i++ ) {
        printf( "%d ", array1[i] );
    }
    printf( "\n" );

    for ( int i = 0; i < tamanhoArray2; i++ ) {
        printf( "%d ", array2[i] );
    }
    printf( "\n" );

    for ( int i = 0; i < tamanhoArray3; i++ ) {
        printf( "%d ", array3[i] );
    }
    printf( "\n" );

    for ( int i = 0; i < tamanhoArray4; i++ ) {
        printf( "%d ", array4[i] );
    }
    printf( "\n" );

    for ( int i = 0; i < tamanhoArray5; i++ ) {
        printf( "%d ", array5[i] );
    }
    printf( "\n" );

    for ( int i = 0; i < N; i++ ) {
        printf( "%d ", array6[i] );
    }
    printf( "\n" );


    return 0;

}
```

Entrada de dados em arrays

```c
/*
 * Arquivo: ArraysUnidimensionaisEntradaDados.c
 * Autor: Prof. Dr. David Buzatto
```

```c
 */

#include <stdio.h>
#include <stdlib.h>

int main() {

    // declaração de um array de inteiros de 5 posições
    int array[5];

    // cálculo do tamanho usando o operador sizeof
    int t = (int) ( sizeof( array ) / sizeof( array[0] ) );


    // lista os dados do array (não foi inicializado!)
    for ( int i = 0; i < t; i++ ) {
        printf( "%d ", array[i] );
    }
    printf( "\n" );

    // inserção dos dados
    for ( int i = 0; i < t; i++ ) {
        printf( "Entre com o valor da posicao %d: ", i );
        scanf( "%d", &array[i] );
    }

    printf( "Dados inseridos: " );
    for ( int i = 0; i < t; i++ ) {
        printf( "%d ", array[i] );
    }
    printf( "\n" );


    return 0;

}
```

4.2 Representação Gráfica de Arrays

Os dados armazenados nos arrays são organizados de forma contígua na memória, ou seja, cada um dos elementos está situado “lado a lado” em endereços de memória

adjacentes, tornando rápido o acesso à cada posição. O padrão de indexação dos elementos do array é algo que gera uma certa confusão para algumas pessoas que estão aprendendo a programar e isso é super normal. Por exemplo, se um array tem capacidade para armazenar 10 elementos, ou seja, seu tamanho é igual a 10, o primeiro elemento estará sempre na posição 0 enquanto o último na posição 9. Isso acontecerá na maioria das linguagens de programação que você terá contato na sua vida profissional, pois a grande parte dessas linguagens são baseadas direta ou indiretamente na linguagem de programação C. Veja o exemplo gráfico de um array de uma dimensão na Figura 4.1. Usualmente utilizamos uma variável nomeada como `i` para acessar as posições de um array de uma dimensão.

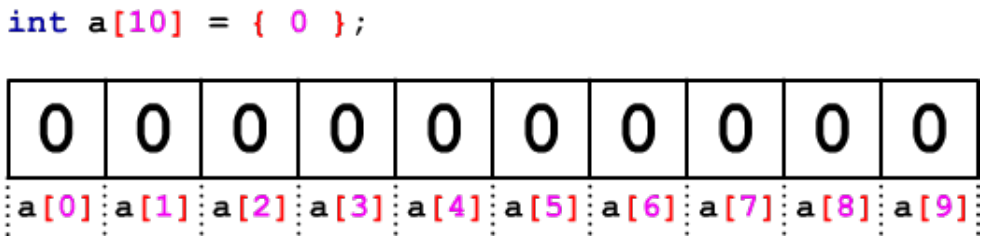


Figura 4.1: Indexação dos Arrays

⚠ | Atenção!

De modo geral, para um array de n elementos, o primeiro elemento está situado na posição 0 e o último na posição $n-1$.

Vamos aos exercícios!

4.3 Exercícios

Exercício 4.1: Escreva um programa que preencha um array de números inteiros de 5 posições a partir de números fornecidos pelo usuário. O programa deve armazenar em um segundo array o cubo de cada elemento do primeiro array. Por fim, o programa deve exibir os valores do array que contém o cubo do primeiro array.

Arquivo com a solução: ex4.1.c

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
```

Saída

```
arrayCubo[0] = 64
arrayCubo[1] = 125
arrayCubo[2] = 343
arrayCubo[3] = 1000
arrayCubo[4] = -512
```

Exercício 4.2: Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. Logo em seguida, o programa deve pedir o valor de um número inteiro. O programa deve multiplicar cada um dos valores do array inicial pelo valor fornecido e armazenar, em um segundo array, também de 5 posições, o valor da multiplicação de cada posição do array inicial pelo valor fornecido após a leitura do primeiro array. Por fim, o programa deve exibir os valores do array que contém a multiplicação de cada item.

Arquivo com a solução: ex4.2.c

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
Multiplicar por: 5
```

Saída

```
arrayMult[0] = 20
arrayMult[1] = 25
arrayMult[2] = 35
arrayMult[3] = 50
arrayMult[4] = -40
```

Exercício 4.3: Escreva um programa que preencha um array de números decimais de 5 posições com valores fornecidos pelo usuário. Após o preenchimento, o programa deve percorrer o array com os dados fornecidos, calculando o somatório e o produtório dos valores contidos no mesmo. Esses resultados devem ser exibidos ao final da execução do programa e devem estar formatados usando duas casas decimais de precisão. Lembrando que:

- $S = \sum_{i=0}^{n-1} a[i]$
- $P = \prod_{i=0}^{n-1} a[i]$
- Onde:
  - S é o somatório dos n elementos do array `a`;
  - P é o produtório dos n elementos do array `a`.

Arquivo com a solução: ex4.3.c

Entrada

```
array[0]: 4
array[1]: 5.5
array[2]: 7
array[3]: 10.7
array[4]: -8
```

Saída

```
Somatorio: 19.20
Produtorio: -13182.40
```

Exercício 4.4: Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. Logo em seguida, o programa deve pedir o valor de um número inteiro. O programa deve apresentar ao usuário a

mensagem “ACHEI” caso o valor seja encontrado em um determinado índice (posição) ou “NAO ACHEI” caso contrário.

Arquivo com a solução: ex4.4.c

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
Buscar por: 5
```

Saída

```
Indice 0: NAO ACHEI
Indice 1: ACHEI
Indice 2: NAO ACHEI
Indice 3: NAO ACHEI
Indice 4: NAO ACHEI
```

Exercício 4.5: Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. Logo em seguida, o programa deve pedir o valor de um número inteiro. O programa deve contar quantas ocorrências do número fornecido foram encontradas no array, apresentando, ao final, essa contagem.

Arquivo com a solução: ex4.5.c

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
Buscar por: 5
```

Saída

```
O array contem 1 ocorrencia do valor 5.
```

Entrada

```
array[0]: 7
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: 7
Buscar por: 7
```

Saída

```
O array contem 3 ocorrencias do valor 7.
```

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
Buscar por: 0
```

Saída

```
O array nao contem o valor 0.
```

Exercício 4.6: Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. Logo em seguida, o programa deve pedir o valor de um número inteiro. O programa deve apresentar ao usuário todos os índices (posições) em que o valor fornecido foi encontrado no array.

Arquivo com a solução: ex4.6.c

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
Buscar por: 5
```

Saída

```
O valor 5 foi encontrado no indice 1 do array.
```

Entrada

```
array[0]: 9
array[1]: 5
array[2]: 7
array[3]: 9
array[4]: 7
Buscar por: 9
```

Saída

```
O valor 9 foi encontrado nos indices 0 e 3 do array.
```

Entrada

```
array[0]: 7
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: 7
Buscar por: 7
```

Saída

```
O valor 7 foi encontrado nos indices 0, 2 e 4 do array.
```

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
Buscar por: 0
```

Saída

```
O array nao contem o valor 0.
```

Exercício 4.7: Escreva um programa que preencha dois arrays de números inteiros de 5 posições com valores fornecidos pelo usuário. O programa deve preencher um terceiro array com a soma dos dois arrays preenchidos previamente e então exibir o array que contém a soma.

Arquivo com a solução: ex4.7.c

Entrada

```
Forneca os valores do primeiro array:
    array1[0]: 5
    array1[1]: 8
    array1[2]: 7
    array1[3]: 2
    array1[4]: 8

Forneca os valores do segundo array:
    array2[0]: 12
    array2[1]: -10
    array2[2]: -5
    array2[3]: -20
    array2[4]: 9
```

Saída

```
arraySoma[0] = 17
arraySoma[1] = -2
arraySoma[2] = 2
arraySoma[3] = -18
arraySoma[4] = 17
```

Exercício 4.8: Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. O programa deve exibir os números pares desse array e depois os números ímpares, todos na ordem em que aparecem no array.

Arquivo com a solução: ex4.8.c

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
```

Saída

```
Numeros pares: 4 10 -8.
Numeros impares: 5 7.
```

Entrada

```
array[0]: 4
array[1]: 8
array[2]: 6
array[3]: 10
array[4]: -8
```

Saída

```
Numeros pares: 4 8 6 10 -8.
Numeros impares: nao ha.
```

Entrada

```
array[0]: 5
array[1]: 9
array[2]: 7
array[3]: 13
array[4]: -9
```

Saída

```
Numeros pares: nao ha.
Numeros impares: 5 9 7 13 -9.
```

Exercício 4.9: Escreva um programa que preencha um array de números inteiros de 5

posições com valores fornecidos pelo usuário. O programa deve copiar os valores desse array para um segundo array, sendo que no segundo array, os valores serão inseridos de forma inversa. Ao final, o programa deve exibir os valores do array invertido.

Arquivo com a solução: ex4.9.c

Entrada

```
array[0]: 4
array[1]: 8
array[2]: 6
array[3]: 10
array[4]: -8
```

Saída

```
arrayInv[0] = -8
arrayInv[1] = 10
arrayInv[2] = 6
arrayInv[3] = 8
arrayInv[4] = 4
```

Exercício 4.10: Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. Logo em seguida, o programa deve pedir o valor de um número inteiro. O programa deve copiar para um segundo array, todos os valores do primeiro array que são maiores que o último valor fornecido. Ao final, o programa deve exibir esses valores.

Arquivo com a solução: ex4.10.c

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
Copiar maiores que: 5
```

Saída

```
arrayCopia[0] = 7
arrayCopia[1] = 10
```

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
Copiar maiores que: 100
```

Saída

```
Nao houve copia!
```

Exercício 4.11: Escreva um programa para ler a quantidade de elementos que serão armazenados em um array de 10 posições de números inteiros. O programa deve aceitar apenas valores entre 1, inclusive, e 9, inclusive. Caso o seja fornecido um valor incorreto, ou seja, fora desse intervalo, o programa deve requisitar novamente a entrada da quantidade. Após a leitura de uma quantidade válida, ele deve ler a quantidade de elementos informados, armazenando-os no array a partir da primeira posição. Logo em seguida, o programa deve pedir o valor de um número inteiro. Esse número deve ser inserido na primeira posição do array. Antes da inserção, perceba que há a necessidade de deslocar os elementos existentes para a casa à direita. Por fim, o programa deve imprimir o array após o deslocamento e a inclusão.

Arquivo com a solução: ex4.11.c

Entrada

```
Quantidade de elementos (1 a 9): 20
Quantidade incorreta, forneca novamente!
Quantidade de elementos (1 a 9): 5
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
Valor que sera inserido: 15
```

Saída

```
array[0] = 15
array[1] = 4
array[2] = 5
array[3] = 7
array[4] = 10
array[5] = -8
```

Exercício 4.12: Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. O primeiro elemento do array deve ser "excluído", deslocando para isso todos os elementos a partir da segunda posição para a esquerda. Por fim, o programa deve imprimir o array após a remoção.

Arquivo com a solução: ex4.12.c

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
```

Saída

```
array[0] = 5
array[1] = 7
array[2] = 10
array[3] = -8
```

Exercício 4.13: Escreva um programa que preencha um array de números inteiros de 10 posições com valores fornecidos pelo usuário. Em seguida, o programa deve ler o índice de uma posição do array, ou seja, um valor de 0 a 9. Caso seja informado uma posição inválida, o programa deve informar o usuário e pedir novamente a posição. Após a leitura da posição válida, o programa deve "remover" do array o elemento contido na posição fornecida. Por fim, o programa deve imprimir o array após a remoção.

Arquivo com a solução: ex4.13.c

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
array[5]: 9
array[6]: 10
array[7]: 14
array[8]: 3
array[9]: 121
Posicao a ser removida (0 a 9): 20
Posicao invalida, forneca novamente!
Posicao a ser removida (0 a 9): 5
```

Saída

```
array[0] = 4
array[1] = 5
array[2] = 7
array[3] = 10
array[4] = -8
array[5] = 10
array[6] = 14
array[7] = 3
array[8] = 121
```

Exercício 4.14: Escreva um programa que preencha um array de números inteiros de 10 posições com valores fornecidos pelo usuário. O programa deve remover do array todos os elementos que forem pares. Por fim, o programa deve imprimir o array após as remoções.

Arquivo com a solução: ex4.14.c

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
array[5]: 9
array[6]: 10
array[7]: 14
array[8]: 3
array[9]: 121
```

Saída

```
array[0] = 5
array[1] = 7
array[2] = 9
array[3] = 3
array[4] = 121
```

Exercício 4.15: Escreva um programa que preencha dois arrays de números inteiros de

5 posições com valores fornecidos pelo usuário. Um terceiro array deve ser preenchido, contendo a intersecção dos elementos contidos nos dois primeiros arrays, ou seja, os valores que são comuns aos dois. Nos dois arrays fornecidos, pode haver repetição de elementos, mas essa repetição não deve ser refletida no array de intersecção. Por fim, o programa deve imprimir o array que contém os valores comuns aos dois arrays fornecidos.

Arquivo com a solução: ex4.15.c

Entrada

```
Forneca os valores do primeiro array:
    array1[0]: 5
    array1[1]: 8
    array1[2]: 7
    array1[3]: 2
    array1[4]: 8

Forneca os valores do segundo array:
    array2[0]: 5
    array2[1]: -10
    array2[2]: -5
    array2[3]: 8
    array2[4]: 2
```

Saída

```
arrayInterseccao[0] = 5
arrayInterseccao[1] = 8
arrayInterseccao[2] = 2
```

Entrada

```
Forneca os valores do primeiro array:
    array1[0]: 5
    array1[1]: 8
    array1[2]: 7
    array1[3]: 2
    array1[4]: 8

Forneca os valores do segundo array:
    array2[0]: 12
    array2[1]: -10
    array2[2]: -5
    array2[3]: -20
    array2[4]: 9
```

Saída

```
Nao ha interseccao entre os elementos dos dois arrays
fornecidos!
```

4.4 Desafios

Desafio 4.1: Escreva um programa que preencha um array de números inteiros de 10 posições com valores fornecidos pelo usuário. O programa deve exibir os números pares desse array em ordem crescente e depois os números ímpares em ordem decrescente.

Arquivo com a solução: de4.1.c

Entrada

```
array[0]: 12
array[1]: 7
array[2]: 5
array[3]: 10
array[4]: -8
array[5]: 121
array[6]: 10
array[7]: 14
array[8]: 3
array[9]: 9
```

Saída

```
Valores pares em ordem crescente: -8 10 10 12 14.
Valores impares em ordem decrescente: 121 9 7 5 3.
```

Entrada

```
array[0]: 8
array[1]: 2
array[2]: 14
array[3]: 16
array[4]: 8
array[5]: 12
array[6]: 2
array[7]: 22
array[8]: 6
array[9]: 44
```

Saída

```
Valores pares em ordem crescente: 2 2 6 8 8 12 14 16 22 44.
Valores impares em ordem decrescente: nao ha.
```

Entrada

```
array[0]: 9
array[1]: 7
array[2]: 3
array[3]: 1
array[4]: 5
array[5]: 75
array[6]: 9
array[7]: 15
array[8]: 13
array[9]: 27
```

Saída

```
Valores pares em ordem crescente: nao ha.
Valores impares em ordem decrescente: 75 27 15 13 9 9 7 5 3 1.
```

CAPÍTULO 5

ARRAYS MULTIDIMENSIONAIS

"Se as pessoas não acreditam que a Matemática é simples, é só porque não percebem o quão complicada é a vida".

John von Neumann

OS arrays multidimensionais permitem que os dados sejam armazenados em mais de uma dimensão. Neste capítulo serão apresentados esses arrays, que na memória são armazenados de forma contígua, assim como nos arrays de uma dimensão, mas que podem ter uma interpretação geométrica para que seja facilitada sua visualização. Na grande maioria das vezes utilizaremos arrays de duas dimensões, algumas vezes chamados erroneamente de matrizes. Essa nomenclatura errada dos arrays se dá, pois muitas vezes os dados armazenados nos mesmos são dados de vetores ou matrizes da álgebra, mas o nome da estrutura é, de fato, array ou arranjo.

5.1 Exemplos em Linguagem C

Declaração e inicialização de arrays multidimensionais

```c
/*
 * Arquivo: Arrays2DDeclaracaoInicializacao.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>

// definição da macro M
#define M 5

// definição da macro N
#define N 2

int main() {

    // declaração de um array de dimensões 3x3 de inteiros
    int array1[3][3];

    /* declarando um array de dimensões 2x2 e inicializando
     * com valores 2
     * valor inicializado = { { 2, 2 },
     *                        { 2, 2 } }
     */
    int array2[2][2] = { 2, 2, 2, 2 };

    /* declarando um array de dimensões 2x3 e inicializando
     * com valores
     * valor inicializado = { { 3, 3, 3 },
     *                        { 0, 0, 0 } }
     */
    int array3[2][3] = { 3, 3, 3 };

    /* declarando um array de dimensões 5x2 e inicializando
     * com zeros
     * valor inicializado = { { 0, 0 },
     *                        { 0, 0 },
     *                        { 0, 0 },
```

```
39      *                       { 0, 0 },
40      *                       { 0, 0 } }
41      */
42     int array4[5][2] = { 0 };
43
44     /* se o inicializador para mais de uma dimensão for usado
45      * o tamanho é obrigatório pelo menos para a primeira dimensão
46      */
47     int array5[3][3] = { { 5, 5, 5 }, { 5, 5, 5 } };
48     int array6[][3] = { { 1, 2, 3 }, { 1, 2, 3 }, { 1, 2, 3 }, { 1, 2, 3 } };
49
50     /* declaração de um array de inteiros de dimensões MxN
51      * M e N são macros que serão expandidas ao serem usadas
52      */
53     int array7[M][N];
54
55
56     // cálculo da quantidade de linhas e de colunas usando o operador sizeof
57     int linhasArray1 = (int) ( sizeof( array1 ) / sizeof( array1[0] ) );
58     int colunasArray1 = (int) ( sizeof( array1[0] ) / sizeof( array1[0][0] )
       ↪ );
59
60     int linhasArray2 = (int) ( sizeof( array2 ) / sizeof( array2[0] ) );
61     int colunasArray2 = (int) ( sizeof( array2[0] ) / sizeof( array2[0][0] )
       ↪ );
62
63     int linhasArray3 = (int) ( sizeof( array3 ) / sizeof( array3[0] ) );
64     int colunasArray3 = (int) ( sizeof( array3[0] ) / sizeof( array3[0][0] )
       ↪ );
65
66     int linhasArray4 = (int) ( sizeof( array4 ) / sizeof( array4[0] ) );
67     int colunasArray4 = (int) ( sizeof( array4[0] ) / sizeof( array4[0][0] )
       ↪ );
68
69     int linhasArray5 = (int) ( sizeof( array5 ) / sizeof( array5[0] ) );
70     int colunasArray5 = (int) ( sizeof( array5[0] ) / sizeof( array5[0][0] )
       ↪ );
71
72     int linhasArray6 = (int) ( sizeof( array6 ) / sizeof( array6[0] ) );
73     int colunasArray6 = (int) ( sizeof( array6[0] ) / sizeof( array6[0][0] )
       ↪ );
74
```

```
75
76      /* i será usado normalmente para controlar a linha atual
77       * e j será usado normalmente para controlar a coluna atual
78       * em um array bidimensional.
79       */
80
81      for ( int i = 0; i < linhasArray1; i++ ) {
82          for ( int j = 0; j < colunasArray1; j++ ) {
83              printf( "%d ", array1[i][j] );
84          }
85          printf( "\n" );
86      }
87      printf( "\n" );
88
89      for ( int i = 0; i < linhasArray2; i++ ) {
90          for ( int j = 0; j < colunasArray2; j++ ) {
91              printf( "%d ", array2[i][j] );
92          }
93          printf( "\n" );
94      }
95      printf( "\n" );
96
97      for ( int i = 0; i < linhasArray3; i++ ) {
98          for ( int j = 0; j < colunasArray3; j++ ) {
99              printf( "%d ", array3[i][j] );
100         }
101         printf( "\n" );
102     }
103     printf( "\n" );
104
105     for ( int i = 0; i < linhasArray4; i++ ) {
106         for ( int j = 0; j < colunasArray4; j++ ) {
107             printf( "%d ", array4[i][j] );
108         }
109         printf( "\n" );
110     }
111     printf( "\n" );
112
113     for ( int i = 0; i < linhasArray5; i++ ) {
114         for ( int j = 0; j < colunasArray5; j++ ) {
115             printf( "%d ", array5[i][j] );
116         }
```

```
117         printf( "\n" );
118     }
119     printf( "\n" );
120 
121     for ( int i = 0; i < linhasArray6; i++ ) {
122         for ( int j = 0; j < colunasArray6; j++ ) {
123             printf( "%d ", array6[i][j] );
124         }
125         printf( "\n" );
126     }
127     printf( "\n" );
128 
129     for ( int i = 0; i < M; i++ ) {
130         for ( int j = 0; j < N; j++ ) {
131             printf( "%d ", array7[i][j] );
132         }
133         printf( "\n" );
134     }
135     printf( "\n" );
136 
137     return 0;
138 
139 }
```

Entrada de dados em arrays multidimensionais

```
1  /*
2   * Arquivo: Arrays2DEntradaDados.c
3   * Autor: Prof. Dr. David Buzatto
4   */
5 
6  #include <stdio.h>
7  #include <stdlib.h>
8 
9  int main() {
10 
11     // declaração de um array de inteiros de dimensões 2x3
12     int array[2][3];
13 
14     // cálculo da quantidade de linhas e de colunas usando o operador sizeof
15     int linhas = (int) ( sizeof( array ) / sizeof( array[0] ) );
```

```
16      int colunas = (int) ( sizeof( array[0] ) / sizeof( array[0][0] ) );
17
18      // lista os dados do array (não foi inicializado!)
19      for ( int i = 0; i < linhas; i++ ) {
20          for ( int j = 0; j < colunas; j++ ) {
21              printf( "%d ", array[i][j] );
22          }
23          printf( "\n" );
24      }
25      printf( "\n" );
26
27      // inserção dos dados
28      for ( int i = 0; i < linhas; i++ ) {
29          for ( int j = 0; j < colunas; j++ ) {
30              printf( "Entre com o valor da posicao [%d][%d]: ", i, j );
31              scanf( "%d", &array[i][j] );
32          }
33      }
34
35      printf( "Dados inseridos:\n" );
36      for ( int i = 0; i < linhas; i++ ) {
37          for ( int j = 0; j < colunas; j++ ) {
38              printf( "%d ", array[i][j] );
39          }
40          printf( "\n" );
41      }
42      printf( "\n" );
43
44      return 0;
45
46  }
```

5.2 Representação Gráfica de Arrays Multidimensionais

Na Figura 5.1 pode-se ver a representação gráfica de um array de duas dimensões como um reticulado. Usualmente utilizamos uma variável nomeada como `i` para acessar as “linhas” dessa estrutura bidimensioanl e uma variável nomeada como `j` para acessar as “colunas”. Além disso, convenciona-se que a primeira dimensão é a que corresponde às linhas enquanto que a segunda é a associada às colunas. Por exemplo, se `a` é um array bidimensional, `a[2][3]` refere-se ao elemento situado na

quarta coluna da terceira linha. Lembre-se da indexação iniciada em zero!

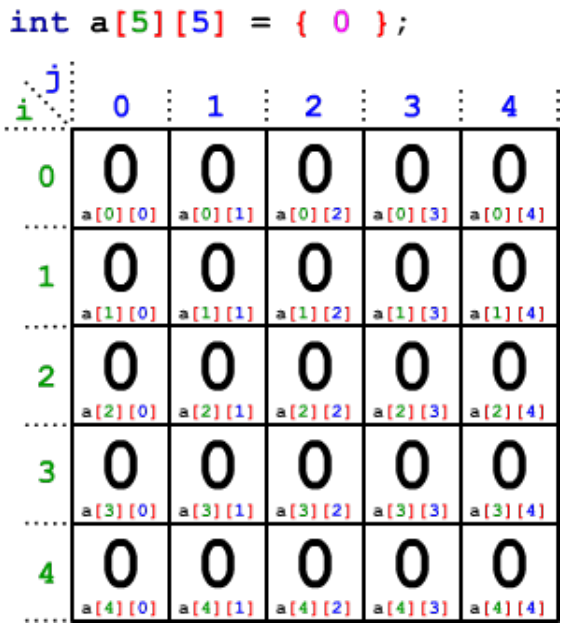


Figura 5.1: Indexação dos Arrays de Duas Dimensões

Já um array de três dimensões pode ser interpretado como um paralelepípedo reto/retângulo. Veja a Figura 5.2. Agora a primeira dimensão corresponde à profundidade desse paralelepípedo, a segunda às linhas de cada reticulado e a terceira dimensão às colunas, ou seja, se `a` é um array tridimensional, `a[2][3][1]` refere-se ao elemento situado na segunda coluna, da quarta linha do terceiro reticulado. Raramente você precisará dessa quantidade de dimensões, muito menos de quantidades maiores, apesar de permitidas. Fica aí um exercício mental. Tente imaginar um array de quatro ou cinco dimensões geometricamente.

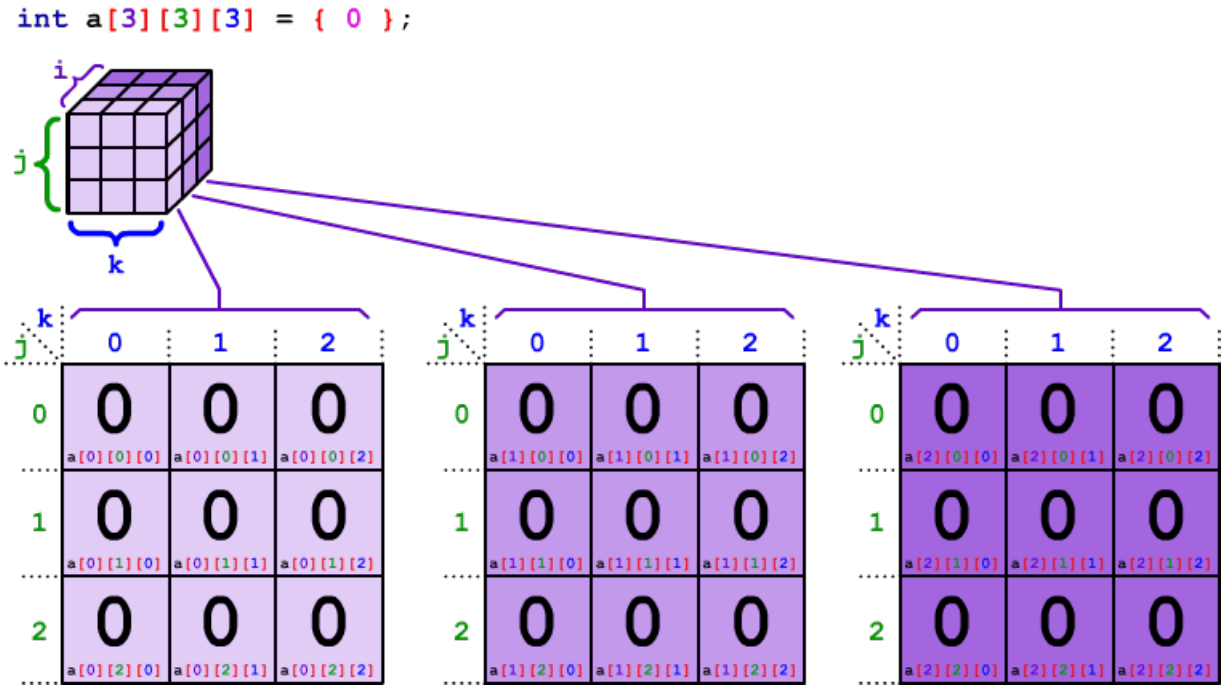


Figura 5.2: Indexação dos Arrays de Três Dimensões

Vamos aos exercícios!

5.3 Exercícios

Exercício 5.1: Escreva um programa que preencha um array de dimensões 3x2 de inteiros com valores fornecidos pelo usuário e o exiba na forma de uma matriz.

Arquivo com a solução: ex5.1.c

Entrada

```
array[0][0]: 1
array[0][1]: 2
array[1][0]: 3
array[1][1]: 4
array[2][0]: 5
array[2][1]: 6
```

Saída

```
001 002
003 004
005 006
```

Exercício 5.2: Escreva um programa que preencha dois arrays de dimensões 3x3 de inteiros com valores fornecidos pelo usuário e armazene a soma desses dois arrays em um terceiro array de dimensões 3x3. No final, o programa deve exibir os três arrays no formado apresentado a seguir.

Arquivo com a solução: ex5.2.c

Entrada

```
array1[0][0]: 4
array1[0][1]: 7
array1[0][2]: 8
array1[1][0]: 5
array1[1][1]: 1
array1[1][2]: 2
array1[2][0]: 6
array1[2][1]: 5
array1[2][2]: 8
array2[0][0]: 9
array2[0][1]: 5
array2[0][2]: 2
array2[1][0]: 1
array2[1][1]: 4
array2[1][2]: 5
array2[2][0]: 6
array2[2][1]: 3
array2[2][2]: 2
```

Saída

```
array1:*******array2:*******arraySoma:
004 007 008***009 005 002***013 012 010
005 001 002 + 001 004 005 = 006 005 007
006 005 008***006 003 002***012 008 010
```

Exercício 5.3: Escreva um programa que preencha um array de dimensões 3x4 de inteiros com valores fornecidos pelo usuário. Em seguida, o programa deve ler o valor de um número inteiro. Armazene em um segundo array de dimensões 3x4 a multiplicação do valor fornecido pelas posições do array preenchido inicialmente. No final, o programa deve exibir o array contendo a multiplicação na forma de uma matriz.

Arquivo com a solução: ex5.3.c

Entrada

```
array[0][0]: 1
array[0][1]: 4
array[0][2]: 5
array[0][3]: 8
array[1][0]: 7
array[1][1]: 4
array[1][2]: 5
array[1][3]: 2
array[2][0]: 3
array[2][1]: 6
array[2][2]: 5
array[2][3]: 4
Multiplicar por: 5
```

Saída

```
arrayMult:
005 020 025 040
035 020 025 010
015 030 025 020
```

Exercício 5.4: Escreva um programa que preencha um array de dimensões 2x2 de inteiros com valores fornecidos pelo usuário. O programa deve calcular e exibir o determinante da matriz representada por esse array. Lembrando que:

- Para $M_{2x2} = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}$,
- $D = a_{1,1} \cdot a_{2,2} - (a_{1,2} \cdot a_{2,1})$
- Onde:
  - M_{2x2} é uma matriz de dimensões 2x2;
  - D é o determinante dessa matriz;

  - **Obs:** cuidado com os índices do array, pois iniciam em 0 e não 1.

Arquivo com a solução: ex5.4.c

Entrada

```
array[0][0]: 4
array[0][1]: 5
array[1][0]: 6
array[1][1]: 1
```

Saída

```
Determinante: -26
```

Exercício 5.5: Escreva um programa que preencha um array de dimensões 3x3 de inteiros com valores fornecidos pelo usuário. O programa deve calcular e exibir o determinante da matriz representada por esse array. Lembrando que:

- Para $M_{3x3} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$,
- $D = \begin{matrix} a_{1,1} \cdot a_{2,2} \cdot a_{3,3} + \cdot a_{1,2} \cdot a_{2,3} \cdot a_{3,1} + \cdot a_{1,3} \cdot a_{2,1} \cdot a_{3,2} - \\ (a_{1,3} \cdot a_{2,2} \cdot a_{3,1} + \cdot a_{1,1} \cdot a_{2,3} \cdot a_{3,2} + \cdot a_{1,2} \cdot a_{2,1} \cdot a_{3,3}) \end{matrix}$
- Onde:
  - M_{3x3} é uma matriz de dimensões 3x3;
  - D é o determinante dessa matriz;
  - **Obs:** cuidado com os índices do array, pois iniciam em 0 e não 1.

Arquivo com a solução: ex5.5.c

Entrada

```
array[0][0]: 4
array[0][1]: 5
array[0][2]: 7
array[1][0]: 8
array[1][1]: 2
array[1][2]: 1
array[2][0]: 3
array[2][1]: 6
array[2][2]: 5
```

Saída

```
Determinante: 125
```

Exercício 5.6: Escreva um programa que preencha um array de dimensões 2x3 de inteiros com valores fornecidos pelo usuário. Esse array será considerado como uma matriz. O programa deve preencher um segundo array de dimensões 3x2 com os valores que representem a matriz transposta da matriz contida do primeiro array. Por fim, o programa deve exibir a matriz original e a matriz transposta. Lembrando que:

- Para $M_{2x3} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix}$,
- $M^t = \begin{bmatrix} a_{1,1} & a_{2,1} \\ a_{1,2} & a_{2,2} \\ a_{1,3} & a_{2,3} \end{bmatrix}$
- Onde:
  - M_{2x3} é uma matriz de dimensões 2x3;
  - M^t é a matriz transposta de M de dimensões 3x2;
  - **Obs:** cuidado com os índices do array, pois iniciam em 0 e não 1.

Arquivo com a solução: ex5.6.c

Entrada

```
array[0][0]: 1
array[0][1]: 2
array[0][2]: 3
array[1][0]: 4
array[1][1]: 5
array[1][2]: 6
```

Saída

```
M:
001 002 003
004 005 006

Mt:
001 004
002 005
003 006
```

Exercício 5.7: Escreva um programa que preencha dois arrays de inteiros, um de dimensões 3x2 enquanto o outro de dimensões 2x3. As duas matrizes representadas pelos arrays devem ser multiplicadas e o resultado deve ser armazenado em um terceiro array bidimensional de dimensões 3x3. Por fim, o programa deve exibir o array que contém a multiplicação. Lembrando que:

- Para $A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{bmatrix}$ e
- $B = \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$,
- $A \cdot B = \begin{bmatrix} a_{1,1} \cdot b_{1,1} + a_{1,2} \cdot b_{2,1} & a_{1,1} \cdot b_{1,2} + a_{1,2} \cdot b_{2,2} & a_{1,1} \cdot b_{1,3} + a_{1,2} \cdot b_{2,3} \\ a_{2,1} \cdot b_{1,1} + a_{2,2} \cdot b_{2,1} & a_{2,1} \cdot b_{1,2} + a_{2,2} \cdot b_{2,2} & a_{2,1} \cdot b_{1,3} + a_{2,2} \cdot b_{2,3} \\ a_{3,1} \cdot b_{1,1} + a_{3,2} \cdot b_{2,1} & a_{3,1} \cdot b_{1,2} + a_{3,2} \cdot b_{2,2} & a_{3,1} \cdot b_{1,3} + a_{3,2} \cdot b_{2,3} \end{bmatrix}$
- Onde:
  - A é uma matriz de dimensões 3x2;
  - B é uma matriz de dimensões 2x3;
  - **Obs:** cuidado com os índices do array, pois iniciam em 0 e não 1.

Arquivo com a solução: ex5.7.c

Entrada

```
array1[0][0]: 2
array1[0][1]: 3
array1[1][0]: 0
array1[1][1]: 1
array1[2][0]: -1
array1[2][1]: 4
array2[0][0]: 1
array2[0][1]: 2
array2[0][2]: 3
array2[1][0]: -2
array2[1][1]: 0
array2[1][2]: 4
```

Saída

```
A x B =
-04 004 018
-02 000 004
-09 -02 013
```

Exercício 5.8 (BEECROWD, 2023): Escreva um programa que leia um inteiro no intervalo de 1, inclusive, a 100, inclusive. Caso o valor fornecido esteja fora desse intervalo, o programa deve avisar o usuário e terminar. Caso contrário, esse valor deve ser usado para gerar uma saída de acordo com o padrão apresentado em cada exemplo. Os valores inteiros devem ser formatados alinhados à direita, com largura de três caracteres, preenchidos com espaços, usando para isso o especificador de formato `"%3d"` e separados por um espaço. Os asteriscos quase pretos indicam espaços. **Obs.:** Este exercício é uma adaptação do problema número 1435 da plataforma de desafios de programação beecrowd (<https://www.beecrowd.com.br/judge/en/problems/view/1435>).

Arquivo com a solução: ex5.8.c

Entrada

```
Numero entre 1 e 100: 1
```

Saída

```
**1
```

Entrada

```
Numero entre 1 e 100: 2
```

Saída

```
**1***1
**1***1
```

Entrada

```
Numero entre 1 e 100: 3
```

Saída

```
**1***1***1
**1***2***1
**1***1***1
```

Entrada

```
Numero entre 1 e 100: 4
```

Saída

```
**1***1***1***1
**1***2***2***1
**1***2***2***1
**1***1***1***1
```

Entrada

```
Numero entre 1 e 100: 5
```

Saída

```
**1***1***1***1***1
**1***2***2***2***1
**1***2***3***2***1
**1***2***2***2***1
**1***1***1***1***1
```

Entrada

```
Numero entre 1 e 100: 10
```

Saída

```
**1***1***1***1***1***1***1***1***1***1
**1***2***2***2***2***2***2***2***2***1
**1***2***3***3***3***3***3***3***2***1
**1***2***3***4***4***4***4***3***2***1
**1***2***3***4***5***5***4***3***2***1
**1***2***3***4***5***5***4***3***2***1
**1***2***3***4***4***4***4***3***2***1
**1***2***3***3***3***3***3***3***2***1
**1***2***2***2***2***2***2***2***2***1
**1***1***1***1***1***1***1***1***1***1
```

Entrada

```
Numero entre 1 e 100: 0
```

Saída

```
Numero incorreto!
```

Entrada

```
Numero entre 1 e 100: 200
```

Saída

```
Numero incorreto!
```

Exercício 5.9 (BEECROWD, 2023): Escreva um programa que leia um inteiro no intervalo de 1, inclusive, a 100, inclusive. Caso o valor fornecido esteja fora desse intervalo, o programa deve avisar o usuário e terminar. Caso contrário, esse valor deve ser usado para gerar uma saída de acordo com o padrão apresentado em cada exemplo. Os valores inteiros devem ser formatados alinhados à direita, com largura de três caracteres, preenchidos com espaços, usando para isso o especificador de formato `"%3d"` e separados por um espaço. Os asteriscos quase pretos indicam espaços. **Obs.:** Este exercício é uma adaptação do problema número 1478 da plataforma de desafios de programação beecrowd (<https://www.beecrowd.com.br/judge/en/problems/view/1478>).

Arquivo com a solução: ex5.9.c

Entrada

```
Numero entre 1 e 100: 1
```

Saída

```
**1
```

Entrada

```
Numero entre 1 e 100: 2
```

Saída

```
**1***2
**2***1
```

Entrada

```
Numero entre 1 e 100: 3
```

Saída

```
**1***2***3
**2***1***2
**3***2***1
```

Entrada

```
Numero entre 1 e 100: 4
```

Saída

```
**1***2***3***4
**2***1***2***3
**3***2***1***2
**4***3***2***1
```

Entrada

```
Numero entre 1 e 100: 5
```

Saída

```
**1***2***3***4***5
**2***1***2***3***4
**3***2***1***2***3
**4***3***2***1***2
**5***4***3***2***1
```

Entrada

```
Numero entre 1 e 100: 10
```

Saída

```
 1  2  3  4  5  6  7  8  9 10
 2  1  2  3  4  5  6  7  8  9
 3  2  1  2  3  4  5  6  7  8
 4  3  2  1  2  3  4  5  6  7
 5  4  3  2  1  2  3  4  5  6
 6  5  4  3  2  1  2  3  4  5
 7  6  5  4  3  2  1  2  3  4
 8  7  6  5  4  3  2  1  2  3
 9  8  7  6  5  4  3  2  1  2
10  9  8  7  6  5  4  3  2  1
```

Entrada

```
Numero entre 1 e 100: 0
```

Saída

```
Numero incorreto!
```

Entrada

```
Numero entre 1 e 100: 200
```

Saída

```
Numero incorreto!
```

5.4 Desafios

Desafio 5.1: Escreva um programa que preencha dois arrays de inteiros, cada um com dimensões que podem variar de 1x1 a 10x10. As dimensões dos arrays iniciais devem ser fornecidas pelo usuário e devem ser compatíveis para que o programa continue, caso contrário o programa deve terminar. As duas matrizes representadas pelos arrays devem ser multiplicadas e o resultado deve ser armazenado em um terceiro array. Por fim, o programa deve exibir o array que contém a multiplicação.

Arquivo com a solução: de5.1.c

Desafio 5.2: Escreva um programa que preencha um array de inteiros, com dimensões que podem variar de 2x2 a 10x10. A dimensão do array inicial deve ser fornecida pelo usuário e deve ser compatível para que o programa continue, caso contrário o programa deve terminar. O programa deve calcular e exibir o determinante da matriz representada por esse array.

Arquivo com a solução: de5.2.c

CAPÍTULO 6

BIBLIOTECA MATEMÁTICA PADRÃO

"A Matemática não mente. Mente quem faz mau uso dela".

Albert Einstein

maioria das linguagens de programação modernas possuem funcionalidades que permitem a execução de diversas funções matemáticas. Neste Capítulo serão apresentadas as funções matemáticas mais comuns que são fornecidas na linguagem de programação C por meio da biblioteca matemática padrão.

6.1 Exemplos em Linguagem C

Principais funções matemáticas em C

```c
/*
 * Arquivo: FuncoesMatematicas.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>
```

```c
#include <math.h>

/* math.h é o include necessário para utilizar as
 * funções matemáticas na linguagem C.
 */

int main() {

    /* constante matemática PI
     * a palavra chave "const" indica que o valor de uma
     * variável não pode ser alterada após a atribuição.
     *
     * recomenda-se que o valor da constante seja gerado
     * usando a função do arco cosseno.
     */
    const double PI = acos(-1);

    printf( "**** modulo ****\n" );

    // função abs: retorna o valor absoluto de um inteiro
    printf( "abs(+3)     = %d\n", abs(+3) );
    printf( "abs(-3)     = %d\n\n", abs(-3) );

    // função fabs: retorna o valor absoluto de um decimal
    printf( "fabs(+3)    = %f\n", fabs(+3.0) );
    printf( "fabs(-3)    = %f\n\n", fabs(-3.0) );



    printf( "**** minimo e maximo ****\n" );

    /* função fmin: retorna o menor valor entre dois valores
     * decimais comparados.
     */
    printf( "fmin(2, 1)  = %.2f\n", fmin(2, 1) );

    /* função fmax: retorna o maior valor entre dois valores
     * decimais comparados.
     */
    printf( "fmax(2, 1)  = %.2f\n\n", fmax(2, 1) );


```

```
50
51      printf( "**** potenciacao e radiciacao ****\n" );
52
53      // função pow (power): eleva uma base a um expoente
54      printf( "pow(2, 10)  = %.2f\n", pow(2, 10) );
55      printf( "pow(1024, 1.0/10) = %.2f\n", pow(1024, 1.0/10) );
56
57      /* função sqrt (square root): calcula a raiz quadrada de um
58       * valor decimal.
59       */
60      printf( "sqrt(100)   = %.2f\n", sqrt(100) );
61
62      /* função cbrt (cube root): calcula a raiz cúbida de um valor
63       * decimal.
64       */
65      printf( "cbrt(729)   = %.2f\n\n", cbrt(729) );
66
67
68
69      printf( "**** funcoes trigonometricas ****\n" );
70
71      /* função sin (sine): calcula o seno de um ângulo com medida
72       * em radianos.
73       */
74      printf( "sin(pi/6)   = %.2f\n", sin(PI/6));    // 30 graus
75
76      /* função cos (cosine): calcula o cosseno de um ângulo com
77       * medida em radianos.
78       */
79      printf( "cos(pi/3)   = %.2f\n", cos(PI/3));    // 60 graus
80
81      /* função tan (tangent): calcula a tangente de um ângulo com
82       * medida em radianos.
83       */
84      printf( "tan(pi/4)   = %.2f\n\n", tan(PI/4)); // 45 graus
85
86
87
88      printf( "**** funcoes trigonometricas " );
89      printf( "inversas (funcoes arco) ****\n" );
90
91      /* função asin (arcsine): calcula o grau em radianos de um
```

```
92      * seno.
93      */
94     printf( "asin(0.5)   = %.2f radianos => %.2f graus\n",
95             asin(0.5), 180/PI * asin(0.5) );
96
97     /* função acos (arccosine): calcula o grau em radianos de
98      * um cosseno.
99      */
100    printf( "acos(0.5)   = %.2f radianos => %.2f graus\n",
101            acos(0.5), 180/PI * acos(0.5) );
102
103    /* função atan (arctangent): calcula o grau em radianos
104     * de uma tangente.
105     */
106    printf( "atan(1)     = %.2f radianos => %.2f graus\n\n",
107            atan(1), 180/PI * atan(1) );
108
109    /* função hypot (hypotenuse): calcula o valor da hipotenusa
110     * com base no valor dos dois catetos.
111     */
112    printf( "hypot(3, 4) = %f\n", hypot(3, 4) );
113
114    /* função atan2 (arctangent2): obtém o ângulo de uma
115     * coordenada cartesiana.
116     */
117    printf( "atan2(4, 3) = (3; 4) cartesiano " );
118    printf( "corresponde a (%.2f, %.2f) polar\n",
119            hypot(4, 3), atan2(4, 3) );
120    printf( "               note que %.2f radianos => ", atan2(4, 3) );
121    printf( "%.2f graus\n\n", 180/PI * atan2(4, 3) );
122
123
124
125    printf( "**** funcoes de arredondamento ****\n" );
126
127    /* função ceil: arredonda um número decimal para o maior
128     * inteiro mais próximo.
129     */
130    printf( "ceil(+2.4)  =  %.2f\n", ceil(2.4) );
131    printf( "ceil(-2.4)  = %.2f\n\n", ceil(-2.4) );
132
133    /* função floor: arredonda um número decimal para o menor
```

```
134       * inteiro mais próximo.
135       */
136     printf( "floor(+2.7) =  %.2f\n", floor(2.7) );
137     printf( "floor(-2.7) = %.2f\n\n", floor(-2.7) );
138
139     // função trunc: remove a parte decimal
140     printf( "trunc(+2.7) =  %.2f\n", trunc(2.7) );
141     printf( "trunc(-2.7) = %.2f\n\n", trunc(-2.7) );
142
143     // função round: arredonda para o inteiro mais próximo
144     printf( "round(+2.3) = %.2f\n", round(2.3) );
145     printf( "round(+2.5) = %.2f\n", round(2.5) );
146     printf( "round(+2.7) = %.2f\n", round(2.7) );
147     printf( "round(-2.3) = %.2f\n", round(-2.3) );
148     printf( "round(-2.5) = %.2f\n", round(-2.5) );
149     printf( "round(-2.7) = %.2f\n", round(-2.7) );
150
151     return 0;
152
153 }
```

Uma função trigonométrica interessante é a `atan2()` que é usada para converter coordenadas cartesianas em coordenadas polares. Na Figura 6.1 pode-se ver um esquema gráfico do uso dessa função. Um exemplo de uso dessa função é simular um olho que acompanha/olha o cursor do mouse. Peça para o professor fazer um exemplo em aula!

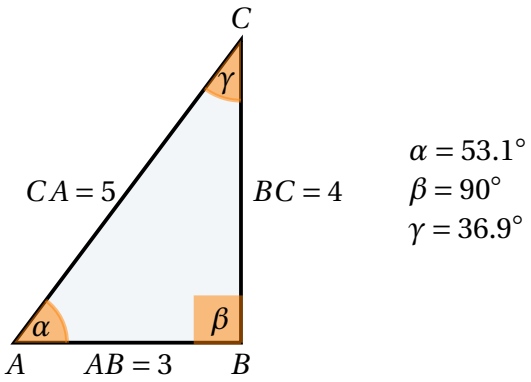


Figura 6.1: Esquema gráfico para entendimento da chamada `atan2(4, 3)` que resulta em 53.1°

Saiba Mais

A lista completa e a documentação das funções matemáticas da linguagem C pode ser encontrada aqui: **<https://en.cppreference.com/w/c/numeric/math>**

Vamos aos exercícios!

6.2 Exercícios

Exercício 6.1: Escreva um programa que peça para o usuário fornecer os coeficientes a, b e c de um polinômio do segundo grau. O programa deve calcular as duas raízes da equação do segundo grau representada por esse polinômio e apresentar o conjunto solução ($S = \{x_1, x_2\}$) ao usuário, sendo que os valores de x devem ser apresentados em ordem crescente. Caso o coeficiente a seja igual a zero, significa que não existe equação do segundo grau, então uma mensagem deve ser exibida ao usuário e o programa deve finalizar. Caso o discriminante da equação (Δ) seja menor que zero, não existem raízes reais, sendo assim, o conjunto solução é vazio. Caso seja igual a zero, as duas raízes têm o mesmo valor e apenas uma deve ser apresentada no conjunto solução. Caso seja maior que zero, existem duas raízes reais distintas que devem ser apresentadas no conjunto solução, em ordem crescente. Apresente também o valor de Δ. Todos os valores são decimais e devem ser apresentados usando duas casas de precisão. Lembrando que, para $ax^2 + bx + c = 0$, tem-se:

- $x = \dfrac{-b \pm \sqrt{\Delta}}{2a}$
- $\Delta = b^2 - 4ac$

Arquivo com a solução: ex6.1.c

Entrada

```
a: 1
b: 5
c: 4
```

Saída

```
Delta: 9.00
S = {-4.00, -1.00}
```

Entrada

```
a: 1
b: 4
c: 4
```

Saída

```
Delta: 0.00
S = {-2.00}
```

Entrada

```
a: 2
b: 2
c: 1
```

Saída

```
Delta: -4.00
S = {}
```

Entrada

```
a: 0
b: 3
c: -2
```

Saída

```
Nao existe equacao do segundo grau!
```

Exercício 6.2: Escreva um programa que peça para o usuário fornecer dois números decimais. Um desses números é a base, enquanto o outro é o expoente. Seu programa deve calcular a base elevada ao expoente e exibir o valor obtido. Exiba o resultado usando duas casas decimais de precisão.

Arquivo com a solução: ex6.2.c

Entrada

```
Base: 2
Expoente: 10
```

Saída

```
2.00 ^ 10.00 = 1024.00
```

Exercício 6.3: Escreva um programa que peça para o usuário fornecer um número decimal. O programa deve calcular e exibir o maior e o menor inteiro mais próximo ao valor fornecido. Exiba os resultados usando duas casas decimais de precisão.

Arquivo com a solução: ex6.3.c

Entrada

```
Numero: 3.5
```

Saída

```
Maior inteiro mais proximo: 4.00
Menor inteiro mais proximo: 3.00
```

Entrada

```
Numero: -3.5
```

Saída

```
Maior inteiro mais proximo: -3.00
Menor inteiro mais proximo: -4.00
```

Exercício 6.4: Escreva um programa que peça para o usuário fornecer um número decimal. O programa deve calcular e exibir o valor absoluto (módulo) do valor fornecido. Exiba o resultado usando duas casas decimais de precisão.

Arquivo com a solução: ex6.4.c

Entrada

```
Numero: 9.5
```

Saída

```
Valor absoluto: 9.50
```

Entrada

```
Numero: -9.5
```

Saída

```
Valor absoluto: 9.50
```

Exercício 6.5: Escreva um programa que peça para o usuário fornecer um número decimal. Caso o número seja positivo, o programa deve calcular e exibir sua raiz quadrada, caso contrário, deve calcular e exibir o quadrado do número. Exiba o resultado usando duas casas decimais de precisão.

Arquivo com a solução: ex6.5.c

Entrada

```
Numero: 9
```

Saída

```
Raiz quadrada de 9.00: 3.00
```

Entrada

```
Numero: -5
```

Saída

```
Quadrado de -5.00: 25.00
```

CAPÍTULO 7

FUNÇÕES

"Form ever follows function".

Louis Henri Sullivan

S funções são a unidade de programação básica das linguagens de programação estruturadas como a linguagem C. Neste Capítulo serão apresentadas as possíveis formas de se declarar e implementar funções em C.

7.1 Exemplos em Linguagem C

Exemplos de prototipação e implementação de funções

```
1 /*
2  * Arquivo: Funcoes.c
3  * Autor: Prof. Dr. David Buzatto
4  */
5
6 #include <stdio.h>
7 #include <stdlib.h>
8
```

```c
/*
 * Protótipos de Funções:
 *
 * O uso de protótipos é aconselhado, visto que seu objetivo
 * é informar ao compilador que essas funções estarão presentes no
 * código. Essa ação de informar é denominada "declaração da função".
 *
 * Pode-se também implementar funções diretamente, sem a declaração
 * de protótipos, entretanto, quando existe dependência entre funções,
 * há a necessidade de implementá-las em ordem, o que nem sempre é
 * possível.
 *
 * Na linguagem C não pode haver mais de uma função com o mesmo nome
 * em um mesmo escopo.
 *
 * As funções devem ser nomeadas, preferencialmente, usando o padrão
 * camel case com a primeira letra em minúscula.
 *
 * Um parâmetro de uma função é parte da função e descreve um tipo de
 * dado que será recebido. É a variável contida na declaração da função.
 *
 * Um argumento é o valor em si, passado através de um parâmetro, para a
 * função utilizar.
 */

/* protótipo da função adicao:
 *   - possui dois parâmetros inteiros => ( int, int );
 *   - retorna um inteiro => int antes do nome da função;
 *   - obs: no protótipo de uma função, não é obrigatório
 *          fornecer o nome/identificador do parâmetro.
 */
int adicao( int, int );

/* protótipo da função subtracao:
 *   - possui dois parâmetros inteiros => ( int n1, int n2 );
 *   - retorna um inteiro => int antes do nome da função;
 *   - obs: no protótipo de uma função, não é obrigatório
 *          fornecer o nome/identificador do parâmetro.
 */
int subtracao( int n1, int n2 );

/* protótipo da função pularLinha
```

```
51  *   - não possui parâmetros => ( void );
52  *   - não retorna nada => void antes do nome da função;
53  *   - obs1: quanto uma função não possui parâmetros, opcionalmente
54  *           usa-se a palavra chave void na lista de parâmetros.
55  *   - obs2: funções que não retornam valores são chamadas também
56  *           de procedimentos.
57  */
58 void pularLinha( void );
59
60 /* protótipo da função imprimirNumeros
61  *   - não possui parâmetros => ();
62  *   - não retorna nada => void antes do nome da função;
63  *   - obs1: quanto uma função não possui parâmetros, opcionalmente
64  *           usa-se a palavra chave void na lista de parâmetros.
65  *   - obs2: funções que não retornam valores são chamadas também
66  *           de procedimentos.
67  */
68 void imprimirNumeros();
69
70 /* protótipo da função processarArray
71  *   - possui dois parâmetros, um array de inteiros e um inteiro => ( int a[],
    ↪  int n );
72  *   - não retorna nada => void antes do nome da função;
73  *   - obs: parâmetros que são arrays tem um comportamento "especial".
74  *          Iremos aprender os detalhes disso posteriormente!
75  *          Por enquanto, entenda que um array passado como parâmetro
    ↪  pode/será
76  *          ser modificado dentro da função.
77  */
78 void processarArray( int a[], int n );
79
80 /* função imprimeTabuada
81  *   - possui um parâmetro inteiros => ( int n );
82  *   - não retorna nada => void antes do nome da função;
83  *   - obs1: quando uma função for implementada, é obrigatória
84  *           a identificação de seus parâmetros
85  *   - obs2: obrigatoriamente, para funções que não possuem
86  *           protótipo, é necessário implementá-las antes de
87  *           usá-las.
88  */
89 void imprimeTabuada( int n ) {
90
```

```
91      for ( int i = 0; i <= 10; i++ ) {
92          printf( "%d x %d = %d\n", n, i, n*i );
93      }
94
95  }
96
97
98  /* função main
99   *    - não possui parâmetros => ();
100  *    - retorna um inteiro => int antes do nome da função.
101  */
102 int main() {
103
104     int n1 = 3;
105     int n2 = 4;
106     int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
107     int resultado;
108
109     printf( "%d + %d = %d", n1, n2, adicao( n1, n2 ) );
110     pularLinha();
111
112     resultado = subtracao( n1, n2 );
113     printf( "%d - %d = %d", n1, n2, resultado );
114     pularLinha();
115
116     printf( "zero a dez: " );
117     imprimirNumeros();
118
119     imprimeTabuada( 5 );
120
121     printf( "Dados do array (fora da funcao):\n" );
122     for ( int i = 0; i < 10; i++ ) {
123         printf( "a[%d] = %d\n", i, a[i] );
124     }
125     processarArray( a, 10 );
126     printf( "Dados do array (apos execucao da funcao):\n" );
127     for ( int i = 0; i < 10; i++ ) {
128         printf( "a[%d] = %d\n", i, a[i] );
129     }
130
131     return 0;
132
```

```c
}

/*
 * implementação da função adicao
 */
int adicao( int n1, int n2 ) {
    return n1 + n2;
}

/*
 * implementação da função subtracao
 */
int subtracao( int n1, int n2 ) {

    /* variável resultado é interna à função!
     * ela tem escopo local à função.
     */
    int resultado = n1 - n2;

    return resultado;

}

/*
 * implementação da função pularLinha
 */
void pularLinha( void ) {
    printf( "\n" );
}

/*
 * implementação da função processarArray
 */
void processarArray( int a[], int n ) {

    /* cuidado, dentro da função não é possível calcular
     * o tamanho do array usando o operador sizeof.
     */

    printf( "Dados do array (dentro da funcao):\n" );
    for ( int i = 0; i < n; i++ ) {
        printf( "a[%d] = %d\n", i, a[i] );
```

```
175     }
176
177     printf( "Modificando os dados do array (dentro da funcao)...\n" );
178     for ( int i = 0; i < n; i++ ) {
179         a[i] += 2;
180     }
181
182     printf( "Dados do array apos modificacao (dentro da funcao):\n" );
183     for ( int i = 0; i < n; i++ ) {
184         printf( "a[%d] = %d\n", i, a[i] );
185     }
186
187 }
188
189 /*
190  * implementação da função imprimirNumeros
191  */
192 void imprimirNumeros() {
193
194     for ( int i = 0; i <= 10; i++ ) {
195         printf( "%d ", i );
196     }
197
198     pularLinha();
199
200 }
```

⚠ | Atenção!

A passagem dos argumentos para as funções em C **sempre** é feita por valor, ou seja, **sempre** são copiados os valores das variáveis que forem usadas na chamada da função para os seus respectivos parâmetros. Infelizmente esse assunto gera muita confusão e as vezes é mal ensinado, onde professores dizem que existe passagem por valor e por referência. Isso está errado! Em C, como em Java, **a passagem é sempre por valor**. Em C++ existe um tipo diferente de variável que é chamada de referência e, nesse caso, existe a passagem por referência, mas isso foge do escopo desse livro. No exemplo apresentado abaixo é mostrado o uso de uma função e na Figura 7.1 pode-se ver uma representação de como isso se dá na memória principal.

Usando uma função

```c
int somar( int n1, int n2 );
int main() {
    int a = 1;
    int b = 2;
    printf( "%d + %d = %d", a, b, somar( a, b ) );
    return 0;
}
int somar( int n1, int n2 ) {
    return n1 + n2;
}
```

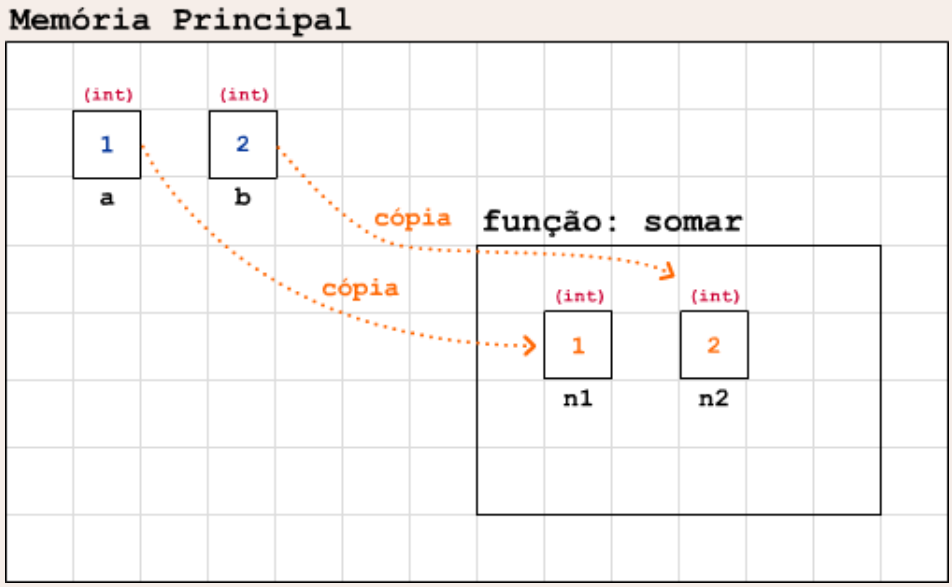


Figura 7.1: Invocação de funções e passagem por valor

7.2 Exercícios

Exercício 7.1: Escreva um programa que leia 5 valores inteiros e imprima para cada um o seu valor absoluto. Para obter o valor absoluto do número utilize a função "absoluto", especificada abaixo:

- **Nome:** `absoluto`
- **Descrição:** Calcula o valor absoluto, ou módulo, do número fornecido.

- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** O valor absoluto de n `(int)`.

Arquivo com a solução: ex7.1.c

Entrada

```
n0: 5
n1: 6
n2: -7
n3: 8
n4: 9
```

Saída

```
absoluto(5) = 5
absoluto(6) = 6
absoluto(-7) = 7
absoluto(8) = 8
absoluto(9) = 9
```

Exercício 7.2: Escreva um programa que leia o valor do raio de um círculo. O programa deve calcular e imprimir a área e o perímetro do círculo representado por esse raio. Para obter o valor da área do círculo o programa deverá chamar a função "areaCirculo" e para obter o valor do seu perímetro o programa deverá invocar a função "circunferenciaCirculo". Para o valor de π, use o dialeto indicado no Capítulo sobre a biblioteca matemática padrão.

- **Nome:** `areaCirculo`
- **Descrição:** Calcula a área do círculo representado pelo raio fornecido.
- **Entrada/Parâmetro(s):** `float raio`
- **Saída/Retorno:** A área do círculo `(float)`.

- **Nome:** `circunferenciaCirculo`
- **Descrição:** Calcula a circunferência do círculo representado pelo raio fornecido.
- **Entrada/Parâmetro(s):** `float raio`
- **Saída/Retorno:** A circunferência do círculo `(float)`.

Arquivo com a solução: ex7.2.c

Entrada

```
Raio: 5
```

Saída

```
Area = 78.54
Circunferencia = 31.42
```

Exercício 7.3: Escreva um programa que leia 5 pares de valores decimais. Todos os valores lidos devem ser positivos. Caso um valor menor ou igual a zero for fornecido, esse valor deve ser lido novamente. Para cada par lido deve ser impresso o valor do maior elemento do par ou a frase "Eles sao iguais" se os valores do par forem iguais. Para obter o maior elemento do par utilize a função "maiorNumero".

- **Nome:** `maiorNumero`
- **Descrição:** Calcula o maior valor entre os dois valores.
- **Entrada/Parâmetro(s):** `float n1, float n2`
- **Saída/Retorno:** Retorna o maior valor os dois fornecidos ou -1 caso sejam iguais `(float)`.
- **Observação:** Considere que os valores de entrada serão sempre positivos.

Arquivo com a solução: ex7.3.c

Entrada

```
n1[0]: 2
n2[0]: 3
n1[1]: 4
n2[1]: 6
n1[2]: 5
n2[2]: 5
n1[3]: -6
Entre com um valor positivo!
n1[3]: 4
n2[3]: -7
Entre com um valor positivo!
n2[3]: -8
Entre com um valor positivo!
n2[3]: 3
n1[4]: 4
n2[4]: 2
```

Saída

```
2.00, 3.00: O maior valor e 3.00
4.00, 6.00: O maior valor e 6.00
5.00, 5.00: Eles sao iguais
4.00, 3.00: O maior valor e 4.00
4.00, 2.00: O maior valor e 4.00
```

Exercício 7.4: Escreva um programa que leia 5 números inteiros positivos, utilizando, para isso, a função "lePositivo". Para cada valor lido escrever o somatório dos inteiros de 1 ao número informado. O resultado do cálculo desse somatório deve ser obtido através da função "somatorio".

- **Nome:** `lePositivo`
- **Descrição:** Faz a leitura de um valor. Se ele for negativo ou zero, a leitura deve ser repetida até que o valor lido seja positivo.
- **Entrada/Parâmetro(s):** nenhum.
- **Saída/Retorno:** Retorna o valor lido `(int)`.

- **Nome:** `somatorio`
- **Descrição:** Calcula o somatório dos inteiros de 1 ao número fornecido como parâmetro.
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** O valor do somatório `(int)`.

Arquivo com a solução: ex7.4.c

Entrada

```
n[0]: 5
n[1]: 4
n[2]: 9
n[3]: -7
Entre com um valor positivo: 8
n[4]: -8
Entre com um valor positivo: -9
Entre com um valor positivo: -4
Entre com um valor positivo: 3
```

Saída

```
Somatorio de 1 a 5: 15
Somatorio de 1 a 4: 10
Somatorio de 1 a 9: 45
Somatorio de 1 a 8: 36
Somatorio de 1 a 3: 6
```

Exercício 7.5: Escreva um programa que leia dois números 5 vezes. O programa deve verificar se o primeiro número fornecido é par e se o primeiro número é divisível pelo segundo, ou seja, se o resto da divisão do primeiro pelo segundo é zero. Para fazer tais verificações, utilize os métodos estáticos "ehPar" e "ehDivisivel".

- **Nome:** `ehPar`
- **Descrição:** Verifica se o número fornecido é ou não par.
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** `true` caso o número seja par ou `false` caso contrário.

- **Nome:** `ehDivisivel`
- **Descrição:** Verifica se um número é divisível por outro.
- **Entrada/Parâmetro(s):** `int dividendo, int divisor`
- **Saída/Retorno:** `true` caso o dividendo seja divisível pelo divisor ou `false` caso contrário.

Arquivo com a solução: ex7.5.c

Entrada

```
n1[0]: 8
n2[0]: 4
n1[1]: 7
n2[1]: 3
n1[2]: 21
n2[2]: 7
n1[3]: 9
n2[3]: 5
n1[4]: 10
n2[4]: 5
```

Saída

```
8 eh par e 8 eh divisivel por 4
7 eh impar e 7 nao eh divisivel por 3
21 eh impar e 21 eh divisivel por 7
9 eh impar e 9 nao eh divisivel por 5
10 eh par e 10 eh divisivel por 5
```

Exercício 7.6: Escreva um programa que leia 5 números inteiros positivos (utilizar "lePositivo"). Para cada número informado escrever a soma de seus divisores (exceto ele mesmo). Utilize a função "somaDivisores" para obter a soma.

- **Nome:** `somaDivisores`
- **Descrição:** Calcula a soma dos divisores do número informado, exceto ele mesmo.
- **Exemplo:** Para o valor 8, tem-se que $1+2+4=7$
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** A soma dos divisores do número fornecido.

Arquivo com a solução: ex7.6.c

Entrada

```
n[0]: 8
n[1]: 10
n[2]: 5
n[3]: -8
Entre com um valor positivo: 9
n[4]: -7
Entre com um valor positivo: -8
Entre com um valor positivo: -7
Entre com um valor positivo: 50
```

Saída

```
Soma dos divisores de 8: 7
Soma dos divisores de 10: 8
Soma dos divisores de 5: 1
Soma dos divisores de 9: 4
Soma dos divisores de 50: 43
```

Exercício 7.7: Escreva um programa que imprima na tela os números primos exis-

tentes entre 1, inclusive, e 20, inclusive. Para verificar se um número é primo utilize a função “ehPrimo”.

- **Nome:** `ehPrimo`
- **Descrição:** Verifica se um número é ou não primo.
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** `true` caso o número seja primo ou `false` caso contrário.

Arquivo com a solução: ex7.7.c

Saída

```
1: nao eh primo
2: eh primo
3: eh primo
4: nao eh primo
5: eh primo
6: nao eh primo
7: eh primo
8: nao eh primo
9: nao eh primo
10: nao eh primo
11: eh primo
12: nao eh primo
13: eh primo
14: nao eh primo
15: nao eh primo
16: nao eh primo
17: eh primo
18: nao eh primo
19: eh primo
20: nao eh primo
```

Exercício 7.8: Escreva um programa que leia 5 pares de valores positivos (“lePositivo”). Imprima se os elementos de cada par são números amigos ou não. Dois números “a” e “b” são amigos se a soma dos divisores de “a” excluindo “a” é igual a “b” e a soma dos divisores de “b” excluindo “b” é igual a “a”. Para verificar se dois números são amigos utilize a função “saoAmigos”.

- **Nome:** `saoAmigos`
- **Descrição:** Verifica se dois números são amigos.

- **Observação:** Utilize a função "somaDividores" do exercício anterior.
- **Exemplo:** 220 e 284 são amigos, pois:
  - **220:** $1+2+4+5+10+11+20+22+44+55+110=284$
  - **284:** $1+2+4+71+142=220$
- **Entrada/Parâmetro(s):** `int n1, int n2`
- **Saída/Retorno:** `true` caso os números sejam amigos ou `false` caso contrário.

Arquivo com a solução: ex7.8.c

Entrada

```
n1[0]: 220
n2[0]: 284
n1[1]: 128
n2[1]: 752
n1[2]: 789
n2[2]: 568
n1[3]: 1184
n2[3]: 1210
n1[4]: 874
n2[4]: 138
```

Saída

```
220 e 284 sao amigos
128 e 752 nao sao amigos
789 e 568 nao sao amigos
1184 e 1210 sao amigos
874 e 138 nao sao amigos
```

Exercício 7.9: Escreva um programa que leia as medidas dos lados de 5 triângulos. Para cada triângulo imprimir a sua classificação (Não é triângulo, Triângulo Equilátero, Isósceles ou Escaleno). O programa deve aceitar apenas valores positivos para as medidas dos lados (utilizar "lePositivo"). Para verificar se as medidas formam um triângulo chamar a função "ehTriangulo". Para obter o código da classificação utilize a função "tipoTriangulo".

- **Nome:** `ehTriangulo`
- **Descrição:** Verifica se as 3 medidas informadas permitem formar um triângulo. Essa condição de existência já foi apresentada em um Capítulo anterior.
- **Entrada/Parâmetro(s):** `int ladoA, int ladoB, int ladoC`

- **Saída/Retorno:** `true` caso os valores representam um triângulo ou `false` caso contrário.

- **Nome:** `tipoTriangulo`
- **Descrição:** A partir das medidas dos lados de um triângulo, verifica o tipo do triângulo.
- **Entrada/Parâmetro(s):** `int ladoA, int ladoB, int ladoC`
- **Saída/Retorno:** Um inteiro, sendo que:
  - **0:** se não formam um triângulo;
  - **1:** se for um triângulo equilátero;
  - **2:** se for um triângulo isósceles;
  - **3:** se for um triângulo escaleno.

Arquivo com a solução: ex7.9.c

Entrada

```
ladoA[0]: 2
ladoB[0]: 2
ladoC[0]: 2
ladoA[1]: 2
ladoB[1]: 3
ladoC[1]: -10
Entre com um valor positivo: -5
Entre com um valor positivo: 5
ladoA[2]: 3
ladoB[2]: 4
ladoC[2]: 5
ladoA[3]: 7
ladoB[3]: 7
ladoC[3]: -15
Entre com um valor positivo: -19
Entre com um valor positivo: 8
ladoA[4]: 1
ladoB[4]: 10
ladoC[4]: 20
```

Saída

```
Valores 2, 2 e 2: triangulo equilatero
Valores 2, 3 e 5: nao formam um triangulo
Valores 3, 4 e 5: triangulo escaleno
Valores 7, 7 e 8: triangulo isosceles
Valores 1, 10 e 20: nao formam um triangulo
```

Exercício 7.10: Escreva um programa que leia um valor inteiro de 1 a 9999 e imprima o seu dígito verificador. Para obter o valor do dígito verificador utilize a função "calculaDigito". Caso seja fornecido um número fora da faixa estabelecida, o programa deve ser encerrado sem apresentar nenhuma mensagem.

- **Nome:** `calculaDigito`
- **Descrição:** Calcula o dígito verificador de um número. Para evitar erros de digitação em números de grande importância, como código de uma conta bancária, geralmente se adiciona ao número um dígito verificador. Por exemplo, o número 1841 é utilizado normalmente como 18414, onde o 4 é o dígito verificador. Ele é calculado da seguinte forma:
  - **a)** Cada algarismo do número é multiplicado por um peso começando em 2, da direita para a esquerda. Para cada algarismo o peso é acrescido de 1. Soma-se então os produtos obtidos. Exemplo: $1*5+8*4+4*3+1*2=51$
  - **b)** Calcula-se o resto da divisão desta soma por 11: $51\%11=7$
  - **c)** Subtrai-se de 11 o resto obtido: $11-7=4$
  - **d)** Se o valor obtido for 10 ou 11, o dígito verificador será o 0, nos outros casos, o dígito verificador é o próprio valor encontrado.
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** O dígito verificador do número `(int)`.

Arquivo com a solução: ex7.10.c

Entrada

```
Numero: 1841
```

Saída

```
Digito verificador de 1841: 4
```

Entrada

```
Numero: 6857
```

Saída

```
Digito verificador de 6857: 8
```

Entrada

```
Numero: 751
```

Saída

```
Digito verificador de 751: 0
```

Exercício 7.11: Escreva um programa que leia um valor inteiro de 10 a 99999, onde o último algarismo representa o seu dígito verificador. Imprima uma mensagem indicando se ele foi digitado corretamente ou não. Utilize a função "numeroCorreto" para verificar se o número está correto. Caso seja fornecido um número fora da faixa estabelecida, o programa deve ser encerrado sem apresentar nenhuma mensagem.

- **Nome:** `numeroCorreto`
- **Descrição:** Verifica se um número, em conjunto com seu dígito, está correto.
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** `true` se o número está correto ou `false` caso contrário.
- **Observação:** Use as funções abaixo: "obtemNumero", "obtemDigito" e "calculaDigito".

- **Nome:** `obtemDigito`
- **Descrição:** Separa o dígito verificador (a unidade) do número.
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** O último algarismo do número `(int)`.
- **Exemplo:** Para o valor 1823 o dígito é 3.

- **Nome:** `obtemNumero`
- **Descrição:** Separa o número do dígito verificador.
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** O número sem o valor da unidade `(int)`.
- **Exemplo:** Para o valor 1823 o número é 182.

Arquivo com a solução: ex7.11.c

Entrada

```
Numero: 18414
```

Saída

```
Numero completo: 18414
Numero: 1841
Digito: 4
Digito calculado: 4
O numero fornecido esta correto!
```

Entrada

```
Numero: 68577
```

Saída

```
Numero completo: 68577
Numero: 6857
Digito: 7
Digito calculado: 8
O numero fornecido esta incorreto!
```

Entrada

```
Numero: 7510
```

Saída

```
Numero completo: 7510
Numero: 751
Digito: 0
Digito calculado: 0
O numero fornecido esta correto!
```

Exercício 7.12: Escreva um programa que leia 3 duplas de valores inteiros. Exibir cada dupla em ordem crescente. A ordem deve ser impressa através da chamada da função "classificaDupla" especificada abaixo:

- **Nome:** `classificaDupla`
- **Descrição:** Imprime em ordem crescente dois valores inteiros.

- **Entrada/Parâmetro(s):** `int n1, int n2`
- **Saída/Retorno:** nenhum.

Arquivo com a solução: ex7.12.c

Entrada

```
n1[0]: 7
n2[0]: 9
n1[1]: 10
n2[1]: 5
n1[2]: 2
n2[2]: 2
```

Saída

```
7 e 9: 7 <= 9
10 e 5: 5 <= 10
2 e 2: 2 <= 2
```

Exercício 7.13: Escreva um programa que leia 3 trincas de valores inteiros. Exibir cada trinca em ordem crescente. A ordem deve ser impressa através da chamada da função "classificaTrinca" especificada abaixo:

- **Nome:** `classificaTrinca`
- **Descrição:** Imprime em ordem crescente três valores inteiros.
- **Entrada/Parâmetro(s):** `int n1, int n2, int n3`
- **Saída/Retorno:** nenhum.

Arquivo com a solução: ex7.13.c

Entrada

```
n1[0]: 9
n2[0]: 5
n2[0]: 1
n1[1]: 8
n2[1]: 7
n2[1]: 6
n1[2]: 3
n2[2]: 3
n2[2]: 3
```

Saída

```
9, 5 e 1: 1 <= 5 <= 9
8, 7 e 6: 6 <= 7 <= 8
3, 3 e 3: 3 <= 3 <= 3
```

Exercício 7.14: Escreva um programa que leia 5 duplas de valores inteiros. Após a leitura de todos os elementos, imprimir as duplas que foram armazenadas nas posições pares em ordem crescente e aquelas armazenadas nas posições ímpares em ordem decrescente. Utilize a função "imprimeDuplaClassificada" especificada abaixo para escrever os elementos na ordem desejada.

- **Nome:** `imprimeDuplaClassificada`
- **Descrição:** Imprime os dois inteiros fornecidos na ordem desejada. A ordem é especificada através do parâmetro `emOrdemCrescente`.
- **Entrada/Parâmetro(s):** `int n1, int n2, bool emOrdemCrescente`
- **Saída/Retorno:** nenhuma.

Arquivo com a solução: ex7.14.c

Entrada

```
n1[0]: 7
n2[0]: 9
n1[1]: 9
n2[1]: 7
n1[2]: 8
n2[2]: 2
n1[3]: 6
n2[3]: 4
n1[4]: 9
n2[4]: 9
```

Saída

```
7 e 9: 7 <= 9
9 e 7: 9 >= 7
8 e 2: 2 <= 8
6 e 4: 6 >= 4
9 e 9: 9 <= 9
```

CAPÍTULO 8

PONTEIROS

"Por referências indiretas, descubra os rumos a seguir".

William Shakespeare

S ponteiros são um recurso fundamental da linguagem de programação C e são usados para armazenar endereços de memória. Nesse Capítulo você aprenderá como declarar, inicializar e utilizar ponteiros.

8.1 Exemplos em Linguagem C

Declaração, inicialização e uso de ponteiros

```c
/*
 * Arquivo: PonteirosDeclaracaoInicializacao.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>

int main() {
```

```
10
11      int numeroInt = 10;
12      float numeroFloat = 15.5;
13
14      // declaração de um ponteiro para inteiro
15      int *pInt;
16
17      /* declaração de um ponteiro para float
18       * e inicialização. O operador & é chamado
19       * operador de endereço
20       */
21      float *pFloat = &numeroFloat;
22
23      // atribuindo um endereço ao ponteiro pInt
24      pInt = &numeroInt;
25
26      /* utilização de operador de indireção (*) para
27       * acessar o valor de uma variável de forma indireta.
28       */
29      printf( "numeroInt: %d\n", numeroInt );
30      printf( "numeroFloat: %.2f\n", numeroFloat );
31      printf( "*pInt: %d\n", *pInt );
32      printf( "*pFloat: %.2f\n\n", *pFloat );
33
34      /* impressão de endereços. especificador de
35       * formato %p
36       */
37      printf( "&numeroInt: %p\n", &numeroInt );
38      printf( "&numeroFloat: %p\n", &numeroFloat );
39      printf( "pInt: %p\n", pInt );
40      printf( "pFloat: %p\n\n\n", pFloat );
41
42
43      numeroInt = 4;
44
45      /* utilização de operador de indireção (*) para
46       * alterar o valor de uma variável de forma indireta.
47       */
48      *pFloat = 21.7;
49
50      printf( "numeroInt: %d\n", numeroInt );
51      printf( "numeroFloat: %.2f\n", numeroFloat );
```

```
52      printf( "*pInt: %d\n", *pInt );
53      printf( "*pFloat: %.2f\n\n", *pFloat );
54
55      printf( "&numeroInt: %p\n", &numeroInt );
56      printf( "&numeroFloat: %p\n", &numeroFloat );
57      printf( "pInt: %p\n", pInt );
58      printf( "pFloat: %p\n\n\n", pFloat );
59
60      return 0;
61
62  }
```

Ponteiros como parâmetros de funções e aritmética de ponteiros

```
1   /*
2    * Arquivo: PonteirosFuncoesAritmetica.c
3    * Autor: Prof. Dr. David Buzatto
4    */
5
6   #include <stdio.h>
7   #include <stdlib.h>
8
9   /* protótipo da função zeraArray:
10   *      percorre um array e atribui zero a cada uma de suas
11   *      posições.
12   *      obs: o array será passado como ponteiro e seus
13   *           valores poderão ser alterados.
14   */
15  void zerarArray( int *a, int n );
16
17  /* protótipo da função zeraArrayErro:
18   *      percorre um array e atribui zero a cada uma de suas
19   *      posições.
20   *      obs: o array será passado como ponteiro e será
21   *           de somente leitura (const)!
22   */
23  void zerarArrayErro( const int *a, int n );
24
25  /* protótipo da função imprimirArray:
26   *      percorre um array e imprime os valores
27   *      obs: o array será passado como ponteiro e será
```

```
28  *          de somente leitura (const)! Na verdade,
29  *          a declaração de um parâmetro como array ou
30  *          como ponteiro são equivalentes!
31  */
32 void imprimirArray( const int *a, int n );
33
34 /* protótipo da função maiorMenor:
35  *     percorre um array e encontra o maior e o menor valor
36  *     obs: a palavra chave const, no local onde foi utilizada
37  *          indica que o array passado é de somente leitura.
38  */
39 void maiorMenor( const int a[], int n, int *max, int *min );
40
41 int main() {
42
43     int arrayZerado[10];
44     int array[10] = { 2, 3, 4, 1, 0, 2, 6, 4, 15, -5 };
45     int *p = array;          // ponteiro para um array
46     int quantidade = 10;
47     int maior;
48     int menor;
49
50     zerarArray( arrayZerado, 10 );
51     imprimirArray( arrayZerado, 10 );
52     printf( "\n\n" );
53
54     maiorMenor( array, quantidade, &maior, &menor );
55
56     imprimirArray( array, 10 );
57     printf( "\nMaior: %d\n", maior );
58     printf( "Menor: %d\n\n", menor );
59
60     // **** aritmética de ponteiros ****
61
62     // altera o primeiro elemento do array
63     *array = 10;
64
65     // altera o segundo elemento do array
66     *(array+1) = 19;
67
68     imprimirArray( array, 10 );
69
```

```
70      printf( "\n\n" );
71      printf( "p = %p\n", p );
72      printf( "array = %p\n\n", array );
73      printf( "array[0] = %d\n", array[0] );
74      printf( "*p = %d\n\n", *p );
75      p++;
76      printf( "array[1] = %d\n", array[1] );
77      printf( "*p = %d\n\n", *p );
78      printf( "array[2] = %d\n", array[2] );
79      printf( "*p = %d", *(++p) );
80
81      return 0;
82
83  }
84
85  void zerarArray( int *a, int n ) {
86
87      for ( int i = 0; i < n; i++ ) {
88          a[i] = 0;
89      }
90
91  }
92
93  void zerarArrayErro( const int *a, int n ) {
94
95      for ( int i = 0; i < n; i++ ) {
96          // alteração de valor de posição, ERRO de compilação
97          //a[i] = 0;
98      }
99
100 }
101
102 void maiorMenor( const int a[], int n, int *max, int *min ) {
103
104     *max = a[0];
105     *min = a[0];
106
107     for ( int i = 1; i < n; i++ ) {
108         if ( *max < a[i] ) {
109             *max = a[i];
110         }
111         if ( *min > a[i] ) {
```

```
112             *min = a[i];
113         }
114     }
115
116 }
117
118 void imprimirArray( const int *a, int n ) {
119
120     for ( int i = 0; i < n; i++ ) {
121         printf( "%d ", a[i] );
122     }
123
124 }
```

 # | Exemplo

No exemplo apresentado abaixo é mostrada a declaração e inicialização uma variável inteira e um ponteiro para inteiros, sendo que a representação gráfica do que acontece na memória principal após a execução desse código é apresentada na Figura 8.1.

Declaração e inicialização de ponteiros

```
1 int n = 10;
2 int *p = &n;
```

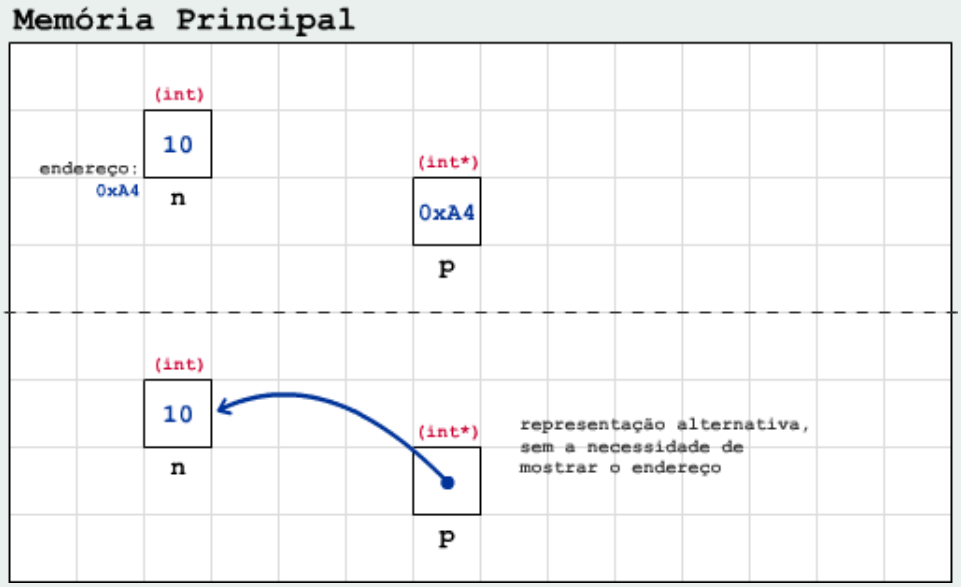


Figura 8.1: Ponteiros na memória

 # | Exemplo

Um detalhe que foi deixado de lado por enquanto é que os arrays na verdade são ponteiros "disfarçados". Sempre que declaramos um array, ele na verdade é um ponteiro que aponta para o endereço do primeiro elemento na estrutura. No exemplo abaixo é declarado um array de `int` e na Figura 8.2 é apresentado o esquema gráfico de como isso se dá na memória.

Declaração e inicialização de um array

```
1 int a[10] = { 0 };
```

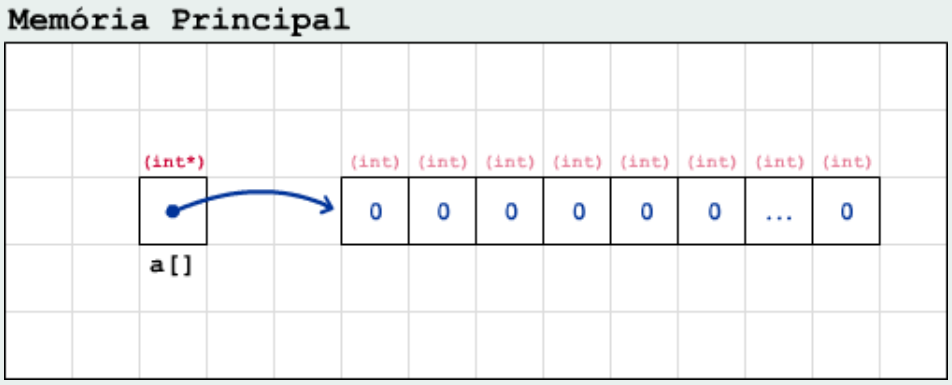


Figura 8.2: Arrays na memória

| Exemplo

Como você já viu, podemos usar ponteiros como parâmetros de funções, permitindo que nós possamos acessar de forma indireta variáveis externas ao escopo da função em questão. No exemplo apresentado abaixo isso é apresentado e um esquema gráfico do que acontece pode ser visto na Figura 8.3.

Função com ponteiros como parâmetro

```
1  // calcula a soma e a média de dois valores
2  int sm( float n1, float n2, float *soma, float *media );
3  int main() {
4      float a = 1;
5      float b = 2;
6      float s;
7      float m;
8      sm( a, b, &s, &m );
9      printf( "Soma = %.2f\n", s );
10     printf( "Média = %.2f", m );
11     return 0;
12 }
13 int sm( float n1, float n2, float *soma, float *media ) {
14     *soma = n1 + n2;
15     *media = *soma / 2;
16 }
```

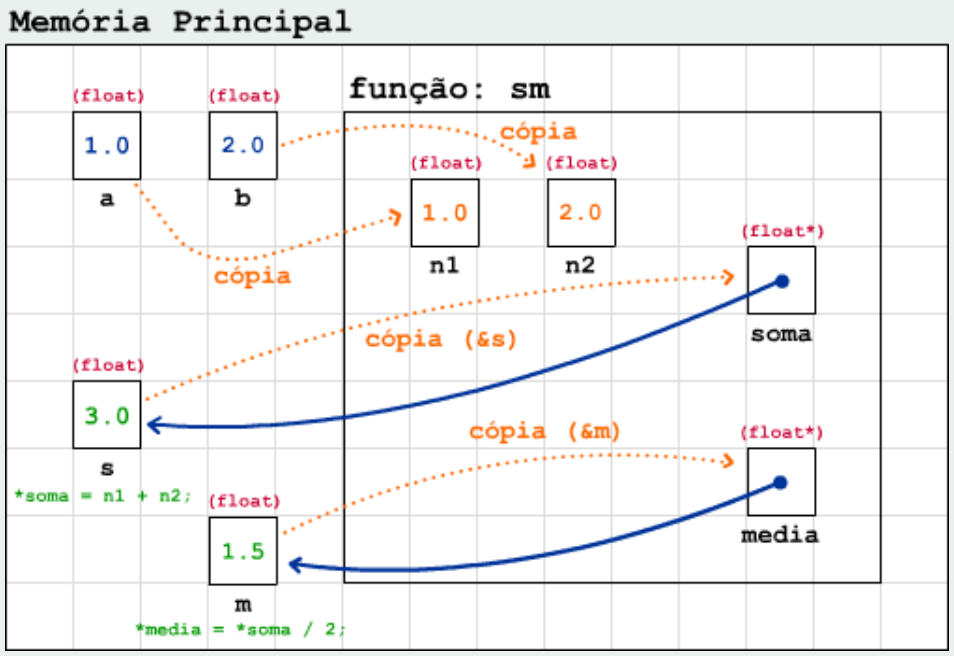


Figura 8.3: Invocação de funções com ponteiros como parâmetro

8.2 Tipos da Linguagem C

Além dos tipos de dados que já aprendemos, em C ainda existem formas de se expandir, diminuir ou restringir a capacidade de representação de tais tipos. Na Tabela 8.1 você pode consultar essas variações e seus tamanhos em bytes e na Tabela 8.2 são apresentados os intervalos de representação de tais tipos e seus respectivos especificadores de formato. Note que alguns tipos podem variar em tamanho dependendo da arquitetura. Por fim, na Tabela 8.3 é possível verificar a precisão dos tipos em ponto flutuante.

Tipo	Descrição	Tamanho (Bytes)
`char`	Unidade básica do conjunto de caracteres. Internamente é um inteiro. Pode ser sinalizado ou não.	1
`signed char`	Mesmo tamanho de char, mas com sinal.	1
`unsigned char`	Mesmo tamanho de char, mas sem sinal.	1

`short`
`short int`
`signed short`
`signed short int`
	Inteiro curto com sinal.	2

`unsigned short` `unsigned short int`	Inteiro curto sem sinal.	2

`int`
`signed`
`signed int`
	Tipo inteiro básico com sinal.	4

`unsigned` `unsigned int`	Tipo inteiro básico sem sinal.	4

`long`
`long int`
`signed long`
`signed long int`
	Tipo inteiro longo com sinal.	4

`unsigned long` `unsigned long int`	Tipo inteiro longo sem sinal.	4

`long long`
`long long int`
`signed long long`
`signed long long int`
	Tipo inteiro longo longo com sinal.	8

`unsigned long long` `unsigned long long int`	Tipo inteiro longo longo sem sinal.	8
`float`	Tipo em ponto flutuante de precisão simples.	4
`double`	Tipo em ponto flutuante de precisão dupla.	8
`long double`	Tipo em ponto flutuante de precisão extendida.	10

Tabela 8.1: Tipos fundamentais - Descrição e Tamanho

Tipo	Intervalo	Especificador de Formato
char	-128 a 127	%c
signed char	-128 a 127	%c ou %hhi para saída numérica
unsigned char	0 a 255	%c ou %hhu para saída numérica
short short int signed short signed short int	-32.768 a 32.767	%hi
unsigned short unsigned short int	0 a 65.535	%hu
int signed signed int	-2.147.483.648 a 2.147.483.647	%i ou %d
unsigned unsigned int	0 a 4.294.967.295	%u
long long int signed long signed long int	-2.147.483.648 a 2.147.483.647	%li
unsigned long unsigned long int	0 a 4.294.967.295	%lu
long long long long int signed long long signed long long int	-9.223.372.036.854.780.000 a 9.223.372.036.854.780.000	%lli
unsigned long long unsigned long long int	0 a 18.446.744.073.709.600.000	%llu
float	1.2E-38 a 3.4E+38	%f %F %g %G %e %E %a %A
double	2.3E-308 a 1.7E+308	%lf %lF %lg %lG %le %lE %la %lA
long double	3.4E-4932 a 1.1E+4932	%Lf %LF %Lg %LG %Le %LE %La %LA

Tabela 8.2: Tipos fundamentais - Intervalo e Especificador de Formato

Tipo	Precisão
`float`	6 casas decimais
`double`	15 casas decimais
`long double`	19 casas decimais

Tabela 8.3: Precisão dos tipos fundamentais de ponto flutuante

Vamos aos exercícios!

8.3 Exercícios

Exercício 8.1 (KING, 2008): Escreva um programa que leia 10 valores decimais, calcule o somatório e a média aritmética dos valores fornecidos e apresente o resultado. O cálculo deve ser feito por meio da função (que você deve implementar):

```
void somatorioMedia( float a[], int n, float *somatorio, float *media )
```

Arquivo com a solução: ex8.1.c

Entrada

```
n[0]: 3
n[1]: 6
n[2]: 7.6
n[3]: 5
n[4]: 4
n[5]: 3
n[6]: 9.8
n[7]: 3
n[8]: 4
n[9]: 7
```

Saída

```
Somatorio: 52.40
Media: 5.24
```

Exercício 8.2 (KING, 2008): Escreva um programa que leia dois valores inteiros e que use a função `void trocar( int *n1, int *n2 )`, implementada por você, para trocar o valor de uma variável com a outra. Ao final, apresente a ordem original e os valores invertidos.

Arquivo com a solução: ex8.2.c

Entrada

```
n1: 6
n2: 19
```

Saída

```
Antes:
    n1: 6
    n2: 19
Depois:
    n1: 19
    n2: 6
```

Exercício 8.3 (KING, 2008): Escreva um programa que leia um valor inteiro que representa uma quantidade de tempo em segundos e que obtenha a quantidade de horas, minutos e segundos contidos nessa quantidade original. O cálculo deve ser feito por meio da função (que você deve implementar):

```
void decompoeTempo( int totalSeg, int *horas, int *minutos, int *seg )
```

Arquivo com a solução: ex8.3.c

Entrada

```
Total de segundos: 12456
```

Saída

```
12456 segundo(s) corresponde(m) a:
    3 hora(s)
    27 minuto(s)
    36 segundo(s)
```

Exercício 8.4 (KING, 2008): Escreva um programa que leia um valor inteiro que representa o dia de um ano (1 a 365) e o ano em si. Não há necessidade de verificar se o dia do ano fornecido está no intervalo correto. A partir desses dados, o programa deve calcular qual é o mês e o dia do mês que correspondem ao dia do ano fornecido, lembrando que um ano bissexto é todo o ano que é divisível por 400 ou por 4, mas não por 100. Para isso, implemente e utilize as funções:

```
1 void decompoeData( int diaDoAno, int ano, int *mes, int *dia )
2 bool ehBissexto( int ano )
```

Arquivo com a solução: ex8.4.c

Entrada

```
Dia do ano: 123
Ano: 2019
```

Saída

```
O dia 123 do ano 2019 cai no dia 3 do mes 5.
```

Entrada

```
Dia do ano: 123
Ano: 2016
```

Saída

```
O dia 123 do ano 2016 cai no dia 2 do mes 5.
```

Exercício 8.5 (KING, 2008): Escreva um programa que leia um array de inteiros de 10 posições e um valor a mais, que será usado para verificar se o mesmo existe no conjunto fornecido. Como resultado do processamento, deve ser apresentado o primeiro índice em que se encontrou o valor desejado. Para isso, implemente e utilize a função `int buscar( const int *a, int n, int chave )`. Essa função deve retornar -1 caso o valor não seja encontrado.

Arquivo com a solução: ex8.5.c

Entrada

```
n[0]: 2
n[1]: 3
n[2]: 5
n[3]: 7
n[4]: 2
n[5]: 3
n[6]: 9
n[7]: 1
n[8]: 2
n[9]: 8
Buscar por: 3
```

Saída

```
O valor 3 foi encontrado na posicao 1.
```

Entrada

```
n[0]: 2
n[1]: 3
n[2]: 5
n[3]: 7
n[4]: 2
n[5]: 3
n[6]: 9
n[7]: 1
n[8]: 2
n[9]: 8
Buscar por: 15
```

Saída

```
O valor 15 nao foi encontrado.
```

Exercício 8.6 (KING, 2008): Escreva um programa que calcule e apresente o produto interno dos valores contidos em dois arrays de números decimais de 5 posições. O produto interno de dois arrays corresponde a $pi[0] = a1[0] * a2[0], pi[1] = a1[1] * a2[1], \ldots, pi[n-1] = a1[n-1] * a2[n-1]$. Para isso, implemente utilize a função:

```
void pInterno( const double *a1, const double *a2, double *pi, int n )
```

Arquivo com a solução: ex8.6.c

Entrada

```
a1[0]: 2
a1[1]: 3
a1[2]: 4
a1[3]: 5
a1[4]: 6
a2[0]: 2
a2[1]: 2
a2[2]: 2
a2[3]: 2
a2[4]: 2
```

Saída

```
2.00 x 2.00 = 4.00
3.00 x 2.00 = 6.00
4.00 x 2.00 = 8.00
5.00 x 2.00 = 10.00
6.00 x 2.00 = 12.00
```

CAPÍTULO 9

CARACTERES E STRINGS

"Adapte os atos às palavras, as palavras aos atos".

William Shakespeare

linguagem de programação C é capaz de lidar com dados do tipo caractere, entretanto não há um tipo específico para trabalhar com cadeias de caracteres, as Strings. Nesse Capítulo você aprenderá como declarar, inicializar e utilizar Strings da forma que a linguagem C foi projetada para lidar com tais dados, além de aprender diversas funções para manipulação de caracteres e Strings.

9.1 Exemplos em Linguagem C

Funções para manipulação de caracteres (DEITEL; DEITEL, 2016)

```
1 /*
2  * Arquivo: CaracteresStringsManipulacaoCaracteres.c
3  * Autor: Prof. Dr. David Buzatto
4  */
5
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

/* ctype.h é o include necessário para utilizar as
 * funções de manipulação de caracteres na linguagem C.
 */

int main() {

    /* int isalpha( int c )
     * Retorna um valor verdadeiro se c for uma letra e 0 (falso)
     * em caso contrário.
     */
    printf ( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n", "De acordo com isalpha: ",
            isalpha('A') ? "A eh uma " : "A nao eh uma ", "letra",
            isalpha('b') ? "b eh uma " : "b nao eh uma ", "letra",
            isalpha('&') ? "& eh uma " : "& nao eh uma ", "letra",
            isalpha('4') ? "4 eh uma " : "4 nao eh uma ", "letra" );

    /* int isalnum( int c )
     * Retorna um valor verdadeiro se c for um dígito ou uma
     * letra e 0 (falso) caso contrário.
     */
    printf( "%s\n%s%s\n%s%s\n%s%s\n\n", "De acordo com isalnum: ",
            isalnum('A') ? "A eh um " : "A nao eh um ",
           "digito ou uma letra",
            isalnum('8') ? "8 eh um " : "8 nao eh um ",
           "digito ou uma letra",
            isalnum('#') ? "# eh um " : "# nao eh um ",
           "digito ou uma letra" );

    /* int isdigit( int c )
     * Retorna um valor verdadeiro se c for um dígito e 0 (falso)
     * caso contrário
     */
    printf( "%s\n%s%s\n%s%s\n\n", "De acordo com isdigit: ",
           isdigit('8') ? "8 eh um " : "8 nao eh um ", "digito",
           isdigit('#') ? "# eh um " : "# nao eh um ", "digito" );

    /* int isxdigit( int c )
     * Retorna um valor verdadeiro se c for um caractere de dígito
```

```
48       * hexadecimal e 0 (falso) caso contrário.
49       */
50      printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
51             "De acordo com isxdigit:",
52              isxdigit('F') ? "F eh um " : "F nao eh um ",
53             "digito hexadecimal",
54              isxdigit('J') ? "J eh um " : "J nao eh um ",
55             "digito hexadecimal",
56              isxdigit('7') ? "7 eh um " : "7 nao eh um ",
57             "digito hexadecimal",
58              isxdigit('$') ? "$ eh um " : "$ nao eh um ",
59             "digito hexadecimal",
60              isxdigit('f') ? "f eh um " : "f nao eh um ",
61             "digito hexadecimal" );
62
63      /* int islower( int c )
64       * Retorna um valor verdadeiro se c for uma letra minúscula e 0
65       * caso contrário.
66       */
67      printf( "%s\n%s%s%s\n%s%s%s\n\n", "De acordo com islower:",
68              "a", islower('a') ? " eh um " : " nao eh um ",
69              "caractere em caixa baixa (minusculo)",
70              "A", islower('A') ? " eh um " : " nao eh um ",
71              "caractere em caixa baixa (minusculo)" );
72
73      /* int isupper( int c )
74       * Retorna um valor verdadeiro se c for uma letra maiúscula e 0
75       * caso contrário.
76       */
77      printf( "%s\n%s%s%s\n%s%s%s\n\n", "De acordo com isupper:",
78              "a", isupper('a') ? " eh um " : " nao eh um ",
79              "caractere em caixa alta (maiusculo)",
80              "A", isupper('A') ? " eh um " : " nao eh um ",
81              "caractere em caixa alta (maiusculo)" );
82
83      /* int tolower( int c )
84       * Se c for uma letra maiúscula, tolower retorna c como uma
85       * letra minúscula. Caso contrário, tolower retorna o argumento
86       * inalterado.
87       */
88      printf( "%s\n%s%c\n%s%c\n%s%c\n\n", "Usando tolower:",
89              "tolower('a'): ", tolower('a'),
```

```
90              "tolower('A'): ", tolower('A'),
91              "tolower('#'): ", tolower('#') );
92
93      /* int toupper( int c )
94       * Se c for uma letra minúscula, toupper retorna c como uma
95       * letra maiúscula. Caso contrário, toupper retorna o argumento
96       * inalterado.
97       */
98      printf( "%s\n%s%c\n%s%c\n%s%c\n\n", "Usando toupper:",
99              "toupper('a'): ", toupper('a'),
100             "toupper('A'): ", toupper('A'),
101             "toupper('#'): ", toupper('#') );
102
103     /* int isspace( int c )
104      * Retorna um valor verdadeiro se c for um caractere de espaço
105      * em branco: nova linha ('\n'), espaço (' '), avanço de folha
106      * ('\f'), retorno de carro ('\r'), tabulação horizontal ('\t') ou
107      * tabulação vertical ('\v') e 0 caso contrário.
108      */
109     printf( "%s\n%s%s%s\n%s%s%s\n%s%s\n\n", "De acordo com isspace:",
110             "'Nova linha'", isspace('\n') ? " eh um " : " nao eh um ",
111             "caractere de espaco em branco",
112             "'Tab. hor.'", isspace('\t') ? " eh um " : " nao eh um ",
113             "caractere de espaco em branco",
114             isspace('%') ? "% eh um " : "% nao eh um ",
115             "caractere de espaco em branco" );
116
117     /* int isblank( int c )
118      * Retorna um valor verdadeiro se c for um caractere de espaço:
119      * espaço (' ') ou tabulação horizontal ('\t') e 0 caso contrário.
120      */
121     printf( "%s\n%s%s%s\n%s%s%s\n%s%s\n\n", "De acordo com isblank:",
122             "'Nova linha'", isblank('\n') ? " eh um " : " nao eh um ",
123             "caractere em branco",
124             "'Tab. hor.'", isblank('\t') ? " eh um " : " nao eh um ",
125             "caractere em branco",
126             isblank('%') ? "% eh um " : "% nao eh um ",
127             "caractere em branco" );
128
129
130     /* int iscntrl( int c )
131      * Retorna um valor verdadeiro se c for um caractere de controle
```

```
132      * e 0 caso contrário.
133      */
134     printf( "%s\n%s%s%s\n%s%s\n\n", "De acordo com iscntrl:",
135             "'Nova linha'", iscntrl('\n') ? " eh um " : " nao eh um ",
136             "caractere de controle ",
137             iscntrl('$') ? "$ eh um " : "$ nao eh um ",
138             "caractere de controle" );
139
140     /* int ispunct( int c )
141      * Retorna um valor verdadeiro se c for um caractere imprimível
142      * diferente de espaço, um dígito ou uma letra e 0 caso contrário.
143      */
144     printf( "%s\n%s%s\n%s%s\n%s%s\n\n", "De acordo com ispunct:",
145             ispunct(';') ? "; eh um " : "; nao eh um ",
146             "caractere de pontuacao",
147             ispunct('Y') ? "Y eh um " : "Y nao eh um ",
148             "caractere de pontuacao",
149             ispunct('#') ? "# eh um " : "# nao eh um ",
150             "caractere de pontuacao" );
151
152     /* int isprint( int c )
153      * Retorna um valor verdadeiro se c for um caractere imprimível
154      * incluindo espaço (' ') e 0 caso contrário.
155      */
156     printf ( "%s\n%s%s\n%s%s%s\n\n", "De acordo com isprint:",
157             isprint('$') ? "$ eh um " : "$ nao eh um ",
158             "caractere imprimivel",
159             "'Alerta'", isprint('\a') ? " eh um " : " nao eh um ",
160             "caractere imprimivel" );
161
162     /* int isgraph( int c ) Retorna um valor verdadeiro se c for um
163      * caractere imprimível diferente de espaço (' ') e 0
164      * caso contrário.
165      */
166     printf( "%s\n%s%s\n%s%s%s\n", "De acordo com isgraph:",
167             isgraph('Q') ? "Q eh um " : "Q nao eh um ",
168             "caractere imprimivel diferente de um espaco",
169             "Espaco", isgraph(' ') ? " eh um " : " nao eh um ",
170             "caractere imprimivel diferente de um espaco" );
171
172     return 0;
173
```

```
174 }
```

Funções para conversão de Strings em valores numéricos

```c
/*
 * Arquivo: CaracteresStringsConversoes.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>

/* stdlib.h é o include necessário para utilizar as
 * funções de conversão de Strings em valores numéricos.
 */

int main() {

    /* double atof( const char* str )
     * Retorna um double com o valor representado pela string str.
     */
    float vFloat = atof( "12.5" );
    double vDouble = atof( "29.75" );

    /* int atoi( const char *str );
     * long atol( const char *str );
     * long long atoll( const char *str );
     * Retorna um inteiro com o valor representado pela string str.
     */
    int vInt = atoi( "10" );
    long vLong = atol( "248" );
    long long vLongLong = atoll( "1795418" );

    // imprimindo...
    printf( "vFloat: %f\n", vFloat );
    printf( "vDouble: %lf\n", vDouble );
    printf( "vInt: %d\n", vInt );
    printf( "vLong: %li\n", vLong );
    printf( "vLongLong: %lli\n", vLongLong );

    return 0;
```

```
38
39 }
```

Conceitos, entrada e saída de Strings

```c
/*
 * Arquivo: CaracteresStringsConceitosES.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* função imprimeCaixa recebe um ponteiro para
 * char e um inteiro como parâmetro.
 * As mesmas regras vistas para ponteiros se aplicam.
 */
void imprimeCaixa( const char *str, int largura );

void removePuloLinha( char *str );

int main() {

    /* Em C, não existe um tipo específico para Strings,
     * mas sim arrays de char marcados no final com um
     * caractere nulo. Sendo assim as Strings em C são
     * também chamadas de "null-terminated Strings"
     */

    /* array de chars inicializado com um literal
     * de String. Usará 13 posições para armazenar
     * os caracteres e uma posição para armazenar
     * o caractere nulo ('\0').
     */
    char nomeCompleto[30] = "David Buzatto";

    /* perceba a necessidade de inserir o caractere
     * nulo como como último caractere.
     */
    char outraForma[10] = { 'o', 'l', 'a', '\0' };
```

```
37
38      /* um ponteiro de char apontando para a String
39       * criada usando um literal.
40       */
41      char *nomeComPonteiro = "Joao da Silva";
42
43      // 29 caracteres no máximo
44      char string1[30];
45      char string2[30];
46      char string3[30];
47
48      // erro de compilação.
49      // só se pode usar o literal na inicialização
50      //string1 = "abc";
51
52      // cinco strings de 29 caracteres
53      char conjuntoStrings[5][30];
54
55      /* imprimindo as strings usando o especificador
56       * de formato %s
57       */
58      printf( "%s\n%s\n",
59              nomeCompleto,
60              nomeComPonteiro );
61
62      /* uma outra forma de imprimir uma string é
63       * usando a função puts. Ele já pula uma linha.
64       */
65      puts( outraForma );
66
67      /* uma string pode ser "quebrada" para fins de
68       * visibilidade.
69       */
70      printf( "Essa eh uma string que ficou feia no \
71              codigo, pois e muito comprida e \
72              dificulta a leitura, entendeu?\n" );
73
74      // assim é melhor!
75      printf( "Essa eh uma string que ficou feia no "
76              "codigo, pois e muito comprida e "
77              "dificulta a leitura, entendeu?\n" );
78
```

```
79      // a leitura pode ser feita de algumas formas
80
81      /* scanf: termina quando encontra algum
82       * caractere de espaço, ficando o restante no buffer
83       */
84      printf( "Entre com a string 1: " );
85      scanf( "%s", string1 ); // não use &, string1 é um ponteiro
86      getchar(); // descarta o caractere de nova linha do buffer
87
88      /* gets: faz a leitura completa, mas deixa o
89       * pulo de linha no buffer e pode gerar overflow
90       */
91      printf( "Entre com a string 2: " );
92      gets( string2 );
93      getchar(); // descarta o caractere de nova linha do buffer
94
95      /* fgets: consome todo o buffer, inserindo o
96       * pulo de linha antes do caractere nulo e evita
97       * overflow. Vamos usar essa função!
98       */
99      printf( "Entre com a string 3: " );
100
101     /* char *fgets ( char *str, int num, FILE *stream )
102      *        char *str: ponteiro para a string que armazenará
103      *                   a leitura.
104      *          int num: limite de caracteres suportados
105      *                   (incluindo o '\0').
106      *     FILE *stream: um ponteiro para arquivo,
107      *                   usaremos stdin.
108      * Retorna o próprio ponteiro passado caso tenha sucesso
109      * ou o ponteiro NULL caso haja algum erro na leitura.
110      */
111     fgets( string3, 30, stdin );
112     string3[strlen(string3)-1] = '\0'; // remove pulo de linha
113
114     printf( "string1: %s\n", string1 );
115     printf( "string2: %s\n", string2 );
116     printf( "string3: %s\n", string3 );
117
118     for ( int i = 0; i < 5; i++ ) {
119         printf( "string %d: ", i );
120         fgets( conjuntoStrings[i], 30, stdin );
```

```
121     }
122
123     for ( int i = 0; i < 5; i++ ) {
124         removePuloLinha( conjuntoStrings[i] );
125         imprimeCaixa( conjuntoStrings[i], 30 );
126     }
127
128
129     return 0;
130
131 }
132
133 void imprimeCaixa( const char *str, int largura ) {
134
135     int c = largura - 3 - strlen( str );
136
137     printf( "+" );
138     for ( int i = 0; i < largura-2; i++ ) {
139         printf( "-" );
140     }
141     printf( "+\n" );
142
143     printf( "| %s", str );
144     for ( int i = 0; i < c; i++ ) {
145         printf( " " );
146     }
147     printf( "|\n" );
148
149     printf( "+" );
150     for ( int i = 0; i < largura-2; i++ ) {
151         printf( "-" );
152     }
153     printf( "+\n" );
154
155 }
156
157 void removePuloLinha( char *str ) {
158
159     int i = 0;
160     while ( str[i] != '\0' ) {
161         i++;
162     }
```

```
163     str[i-1] = '\0';
164 
165 }
```

Funções para manipulação de Strings

```
1  /*
2   * Arquivo: CaracteresStringsFuncoes.c
3   * Autor: Prof. Dr. David Buzatto
4   */
5  
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <string.h>
9  
10 /* string.h é o include necessário para utilizar as
11  * funções de manipulação de strings na linguagem C.
12  */
13 
14 int main() {
15 
16     char string1[30] = "david";
17     char string2[30] = "fernanda";
18     char string3[30] = "buzatto";
19     char string4[30];
20     int comparacao;
21 
22     /* size_t strlen( const char *str )
23      * Retorna o tamanho da string (quantidade de
24      * caracteres armazenados).
25      */
26     printf( "A string \"%s\" possui %d caracteres.\n",
27             string1, strlen( string1 ) );
28 
29     /* int strcmp( const char *str1, const char *str2 )
30      * Compara str1 com str2 e retorna:
31      *     um valor negativo, caso str1 venha antes de str2;
32      *     zero, caso str1 seja igual a str2;
33      *     um valor positivo, vaso str1 venha após str2.
34      */
35     comparacao = strcmp( string1, string2 );
```

```
36      if ( comparacao < 0 ) {
37          printf( "%s vem antes de %s!\n", string1, string2 );
38      } else if ( comparacao > 0 ) {
39          printf( "%s vem antes de %s!\n", string2, string1 );
40      } else {
41          printf( "%s e %s tem o mesmo conteudo!\n", string1, string2 );
42      }
43
44      /* char *strcat( char *destino, const char *fonte );
45       * Concatena em destino o conteúdo de fonte.
46       */
47      strcat( string1, " " );
48      strcat( string1, string3 );
49      printf( "%s\n", string1 );
50
51      /* char *strcpy( char *destino, const char *fonte );
52       * Copia em destino o conteúdo de fonte.
53       */
54      strcpy( string4, "aurora" );
55      printf( "%s\n", string4 );
56
57      return 0;
58
59  }
```

Formatação de Strings usando a função sprintf

```
1   /*
2    * Arquivo: CaracteresStringsFormatacao.c
3    * Autor: Prof. Dr. David Buzatto
4    */
5
6   #include <stdio.h>
7   #include <stdlib.h>
8
9   int main() {
10
11      char string[30];
12
13      int dia = 25;
14      int mes = 2;
```

```c
    int ano = 1985;

    /* int sprintf( char *buffer, const char *formato, ... );
     *
     * A função sprintf é definida no cabeçalho stdio.h e
     * é usada para realizar o mesmo trabalho que a função
     * printf, só que, ao invés de enviar a String formatada
     * para a saída, ela armazena o resultado em uma String.
     *
     * Todas as regras de formatação aplicadas no printf
     * são aplicadas na sprintf também.
     */
    sprintf( string, "%02d/%02d/%04d", dia, mes, ano );
    printf( "%s", string );

    return 0;

}
```

Processamento de Parâmetros Via Linha de Comando

```c
/*
 * Arquivo: CaracteresStringsArgumentosLinhaComando.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

#include <string.h>
#include <ctype.h>
#include <math.h>


bool ehInteiro( const char *string );

/* A função main aceita 3 assinaturas no padrão da linguagem C.
 * O primeiro já é conhecido: int main().
 * O segundo e o terceiro são análogos. Uma de suas formas é usada
 * abaixo. A outra é int main( int argc, char **argv )
```

```
21  *
22  * Através das formas 2 e 3 permitidas, podemos processar
23  * argumentos passados via linha de comando ao se executar o
24  * programa.
25  *
26  * argc (argument count) é um inteiro que contém a quantidade de
27  * argumentos que foram enviados para a execução. argv (argument
28  * values) é um array de strings que contém o valor de cada argumento.
29  */
30 int main( int argc, char *argv[] ) {
31
32     int n1;
33     int n2;
34
35     if ( argc > 1 ) {
36
37         if ( strcmp( argv[1], "/?" ) == 0 ) {
38             printf( "Programa desenvolvido por David Buzatto." );
39         } else if ( strcmp( argv[1], "/calc" ) == 0 ) {
40
41             if ( argc == 5 ) {
42
43                 if ( ehInteiro( argv[2] ) ) {
44                     n1 = atoi( argv[2] );
45                 } else {
46                     printf( "Voce precisa fornecer numeros inteiros!" );
47                     return 1;
48                 }
49
50                 if ( ehInteiro( argv[4] ) ) {
51                     n2 = atoi( argv[4] );
52                 } else {
53                     printf( "Voce precisa fornecer numeros inteiros!" );
54                     return 1;
55                 }
56
57                 if ( strcmp( argv[3], "+" ) == 0 ) {
58                     printf( "%d + %d = %d", n1, n2, n1 + n2 ) ;
59                 } else if ( strcmp( argv[3], "-" ) == 0 ) {
60                     printf( "%d - %d = %d", n1, n2, n1 - n2 ) ;
61                 } else if ( strcmp( argv[3], "x" ) == 0 ) {
62                     printf( "%d x %d = %d", n1, n2, n1 * n2 ) ;
```

```
63                  } else if ( strcmp( argv[3], "/" ) == 0 ) {
64                      printf( "%d / %d = %d", n1, n2, n1 / n2 ) ;
65                  } else if ( strcmp( argv[3], "pow" ) == 0 ) {
66                      printf( "%d pow %d = %d", n1, n2, (int) pow( n1, n2 ) ) ;
67                  } else {
68                      printf( "Operador invalido!" );
69                      return 1;
70                  }
71
72              }
73
74          }
75
76      }
77
78      return 0;
79
80  }
81
82  bool ehInteiro( const char *string ) {
83
84      for ( int i = 0; i < strlen( string ); i++ ) {
85          if ( !isdigit( string[i] ) ) {
86              return false;
87          }
88      }
89
90      return true;
91
92  }
```

Saiba Mais

A documentação da lista completa de funções para manipulação de Strings da linguagem C pode ser encontrada aqui: **<https://en.cppreference.com/w/c/string/byte>**

Exemplo

Como as strings em C são arrays, elas compartilham do mesmo comportamento ou relacionamento com os ponteiros. No exemplo abaixo é declarado um array

de `char` que será inicializado como uma string e na Figura 9.1 é apresentado o esquema gráfico de como isso se dá na memória.

Declaração e inicialização de uma string

```
1 char s[6] = "david";
```

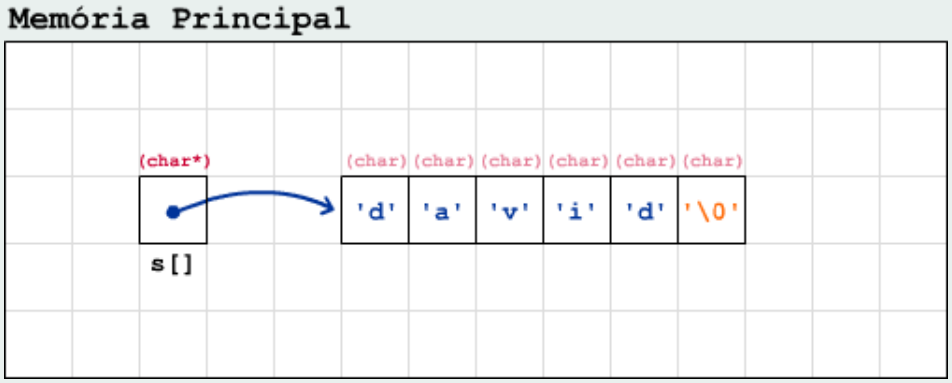


Figura 9.1: Strings na memória

9.2 Exercícios

Atenção! Para todos os exercícios a seguir, considere que as Strings possuem, no máximo, 40 caracteres válidos.

Exercício 9.1: Escreva um programa para ler uma string e apresentar seus quatro primeiros caracteres.

Arquivo com a solução: ex9.1.c

Entrada

```
String: essa eh uma string
```

Saída

```
e, s, s, a.
```

Exercício 9.2: Escreva um programa para ler uma sentença e apresentar:

- O primeiro caractere da sentença;

- O último caractere da sentença;
- O número de caracteres existente na sentença.

Arquivo com a solução: ex9.2.c

Entrada

```
Sentenca: ola, como vai, tudo bem?
```

Saída

```
Primeiro caractere: o
Ultimo caractere: ?
Numero de caracteres: 24
```

Exercício 9.3: Escreva um programa para ler uma sentença e imprimir todos os seus caracteres das posições pares. Se algum caractere for um espaço, imprima-o cercado de aspas simples.

Arquivo com a solução: ex9.3.c

Entrada

```
Sentenca: um dois tres
```

Saída

```
u, ' ', o, s, t, e
```

Exercício 9.4: Escreva um programa para ler uma sentença e imprimir todos os seus caracteres das posições ímpares.

Arquivo com a solução: ex9.4.c

Entrada

```
Sentenca: um dois tres
```

Saída

```
m, d, i, ' ', r, s
```

Exercício 9.5: Escreva um programa para ler um nome e imprimi-lo 5 vezes, um por linha.

Arquivo com a solução: ex9.5.c

Entrada

```
Nome: Fernanda
```

Saída

```
Fernanda
Fernanda
Fernanda
Fernanda
Fernanda
```

Exercício 9.6: Escreva um programa que receba um nome e que o imprima tantas vezes quanto forem seus caracteres.

Arquivo com a solução: ex9.6.c

Entrada

```
Nome: Aurora
```

Saída

```
Aurora
Aurora
Aurora
Aurora
Aurora
Aurora
```

Exercício 9.7: Escreva um programa que leia 5 pares de strings e que imprima:

- IGUAIS, se as strings do par forem iguais;
- ORDEM CRESCENTE, se as strings do par foram fornecidas em ordem crescente;
- ORDEM DECRESCENTE, se as strings do par foram fornecidas em ordem decrescente.

Arquivo com a solução: ex9.7.c

Entrada

```
Par 1, palavra 1: Joao
Par 1, palavra 2: Maria
Par 2, palavra 1: Fernanda
Par 2, palavra 2: David
Par 3, palavra 1: Rafaela
Par 3, palavra 2: Rafaela
Par 4, palavra 1: Renata
Par 4, palavra 2: Cecilia
Par 5, palavra 1: Joana
Par 5, palavra 2: Zelia
```

Saída

```
Joao - Maria: ORDEM CRESCENTE
Fernanda - David: ORDEM DECRESCENTE
Rafaela - Rafaela: IGUAIS
Renata - Cecilia: ORDEM DECRESCENTE
Joana - Zelia: ORDEM CRESCENTE
```

Exercício 9.8: Escreva um programa que leia três strings. A seguir imprimir as 3 strings em ordem alfabética.

Arquivo com a solução: ex9.8.c

Entrada

```
String 1: Luiz
String 2: Everton
String 3: Breno
```

Saída

```
Breno, Everton e Luiz
```

Exercício 9.9: Escreva um programa para ler uma string. A seguir copie para outra string a string informada na ordem inversa e imprima-a. Para isso, implemente e utilize a função `void inverter( char *destino, const char *origem )`.

Arquivo com a solução: ex9.9.c

Entrada

```
String: abacate verde
```

Saída

```
Invertida: edrev etacaba
```

Exercício 9.10: Escreva um programa para ler uma frase e imprimir o número de caracteres dessa frase. Implemente a função `int tamanho( const char *str )` que fará o trabalho de contar a quantidade de caracteres.

⚠ | **Atenção!**

Não use a função `int strlen( const char *str )`!

Arquivo com a solução: ex9.10.c

Entrada

```
Frase: vou comprar um lanche
```

Saída

```
21 caractere(s)!
```

Exercício 9.11: Escreva um programa para ler um caractere e logo após uma frase. Para cada frase informada, imprimir o número de ocorrências do caractere na frase. O programa deve ser encerrado quando a frase digitada for a palavra "fim", que por sua vez não deve ter as ocorrências do caractere informado contadas. A contagem de ocorrências deve ser feita pela função `int contarOcorrencias( const char *str, char c )` implementada por você.

Arquivo com a solução: ex9.11.c

Entrada

```
Caractere: a
Frase: camarao assado
Frase: mas que cabelo sujo!
Frase: fim
```

Saída

```
"camarao assado" tem 5 ocorrencia(s) do caractere 'a'
"mas que cabelo sujo!" tem 2 ocorrencia(s) do caractere 'a'
```

Exercício 9.12: Escreva um programa para ler uma frase e contar o número de ocorrências de cada uma das 5 primeiras letras do alfabeto (tanto maiúsculas quanto minúsculas) e imprimir essas contagens. Você pode usar a função `contarOcorrencias` implementada anteriormente.

Arquivo com a solução: ex9.12.c

Entrada

```
Frase: UI, QUE medo DO white walker!
```

Saída

```
A/a: 1
B/b: 0
C/c: 0
D/d: 2
E/e: 4
```

Exercício 9.13: Escreva um programa para ler uma frase e contar o número de ocorrências das letras A, E, I, O e U (tanto maiúsculas quanto minúsculas) e imprimir essas contagens. Você pode usar a função `contarOcorrencias` implementada anteriormente.

Arquivo com a solução: ex9.13.c

Entrada

```
Frase: UI, QUE medo DO white walker!
```

Saída

```
A/a: 1
E/e: 4
I/i: 2
O/o: 2
U/u: 2
```

Exercício 9.14: Escreva um programa para ler uma frase. A seguir converter todas as letras minúsculas existentes na frase para maiúsculas e apresentar a frase modificada. A conversão deve ser feita por meio da função `void tornarMaiuscula( char *str )`.

Arquivo com a solução: ex9.14.c

Entrada

```
Frase: Fui comprar um COMPUTADOR.
```

Saída

```
FUI COMPRAR UM COMPUTADOR.
```

Exercício 9.15: Escreva um programa para ler uma frase. A seguir converter todas as letras maiúsculas existentes na frase para minúsculas e apresentar a frase modificada. A conversão deve ser feita por meio da função `void tornarMinuscula( char *str )`.

Arquivo com a solução: ex9.15.c

Entrada

```
Frase: Fui comprar um COMPUTADOR.
```

Saída

```
fui comprar um computador.
```

Exercício 9.16: Escreva um programa para ler uma frase e um caractere. A seguir retirar da frase todas as letras iguais (tanto as ocorrências maiúsculas quanto minúsculas) à informada e imprimir a frase modificada. A remoção deve ser feita usando a função `void removerLetra( char *str, char c )`.

Arquivo com a solução: ex9.16.c

Entrada

```
Frase: ja acabou, JESSICA?
Caractere: a
```

Saída

```
j cbou, JESSIC?
```

Exercício 9.17: Escreva um programa para ler uma frase e contar o número de palavras existentes na frase. Considere palavra um conjunto qualquer de caracteres separados por um conjunto qualquer de espaços em branco. A contagem deve ser feita por meio da função `int contarPalavras( const char *str )`.

Arquivo com a solução: ex9.17.c

Entrada

```
Frase: A aranha arranha a ra.
```

Saída

```
Quantidade de palavras: 5
```

Exercício 9.18: Escreva um programa que leia uma string e verifique se a mesma é um palíndromo. Um palíndromo é toda sentença que pode ser lida da mesma forma da esquerda para a direita e vice-versa. Letras maiúsculas e minúsculas não devem ser diferenciadas. Implemente para isso a função `bool ehPalindromo( const char *str )`.

Arquivo com a solução: ex9.18.c

Entrada

```
String: arara
```

Saída

```
"arara" eh um palindromo!
```

Entrada

```
String: Arara
```

Saída

```
"Arara" nao eh um palindromo!
```

Entrada

```
String: ArarA
```

Saída

```
"ArarA" eh um palindromo!
```

Entrada

```
String: Macaco
```

Saída

```
"Macaco" nao eh um palindromo!
```

Exercício 9.19: Escreva um programa que recorte uma string com base em uma posição inicial e uma posição final. O índice inicial é inclusivo e o final é exclusivo. Caso algum índice inválido seja fornecido, a string não deve ser recortada, sendo copiada inteiramente para o destino. Para isso, implemente a função:

```
void substring( char *recorte, const char *origem, int inicio, int fim )
```

Arquivo com a solução: ex9.19.c

Entrada

```
String: Girafa
Inicio: 0
Fim: 3
```

Saída

```
Recorte: Gir
```

Entrada

```
String: Girafa
Inicio: 5
Fim: 3
```

Saída

```
Recorte: Girafa
```

Entrada

```
String: Girafa
Inicio: 5
Fim: 10
```

Saída

```
Recorte: Girafa
```

Entrada

```
String: Girafa
Inicio: 10
Fim: 12
```

Saída

```
Recorte: Girafa
```

Exercício 9.20: Escreva um programa que leia duas strings e que verifique se a segunda está contida na primeira. Considere que letras maiúsculas e minúsculas são diferentes. Para isso, implemente a função:

```
bool contem( const char *fonte, const char *aPesquisar )
```

Arquivo com a solução: ex9.20.c

Entrada

```
String fonte: rabanete
String a pesquisar: bane
```

Saída

```
"bane" esta contida em "rabanete"
```

Entrada

```
String: Hamburguer
String a pesquisar: Hambo
```

Saída

```
"Hambo" nao esta contida em "Hamburguer"
```

Exercício 9.21: Escreva um programa que leia uma string e que a imprima centralizada no terminal, considerando que o terminal tem 80 colunas. Para isso, implemente a função `void imprimirCentralizado( const char *str )`

Arquivo com a solução: ex9.21.c

Entrada

```
String: um texto qualquer
```

Saída

```
                        um texto qualquer
```

Exercício 9.22: Escreva um programa que leia uma string e que a imprima alinhada à direita no terminal, considerando que o terminal tem 80 colunas. Para isso, implemente a função `void imprimirDireita( const char *str )`

Arquivo com a solução: ex9.22.c

Entrada

```
String: um texto qualquer
```

Saída

```
                                                um texto qualquer
```

Exercício 9.23: Escreva um programa que leia uma string e que a imprima dentro de uma caixa desenhada usando os símbolos =, + e |. Para isso, implemente a função `void imprimirCaixa( const char *str )`.

Arquivo com a solução: ex9.23.c

Entrada

```
String: um texto qualquer
```

Saída

```
++====================++
|| um texto qualquer ||
++====================++
```

CAPÍTULO 10

ESTRUTURAS - *Structs*

"Nunca entendi o que significam esses malditos pontos".

Winston Churchill

S estruturas permitem que o programador da linguagem C crie Tipos Abstratos de Dados (TADs), de modo a modelar objetos do mundo real e/ou que sejam necessários para o desenvolvimento do algoritmo desejado. Neste Capítulo você aprenderá como declarar, inicializar e utilizar estruturas.

10.1 Exemplos em Linguagem C

Definição e uso de estruturas

```
1 /*
2  * Arquivo: Estruturas.c
3  * Autor: Prof. Dr. David Buzatto
4  */
5
6 #include <stdio.h>
7 #include <stdlib.h>
```

```c
#include <string.h>

/*
 * struct anônima
 * membros a, b e c, do tipo inteiro
 */
struct {
    int a;
    int b;
    int c;              // inicialização usando designadores
} v1, v2 = { 4, 8, 2 }, vn = { .a = 3, .b = 9, .c = 15 };

/*
 * struct chamada x
 * membros a, b e c do tipo inteiro
 */
struct x {
    int a;
    int b;
    int c;
};

/*
 * definição de tipo (Data) usando
 * uma estrutura
 */
typedef struct {
    int dia;
    int mes;
    int ano;
} Data;

/*
 * definição de tipo (Pessoa) usando
 * uma estrutura
 */
typedef struct {
    char nomeCompleto[40];
    float peso;
    Data dataNascimento;
} Pessoa;

```

```c
void imprimirX( struct x p );
void imprimirData( Data d );
void imprimirDataPonteiro( Data *d );
Pessoa novaPessoa( const char *nc, float p, Data *dn );
void imprimirPessoa( Pessoa *p );

int main() {

    /* declaração de uma variável do tipo struct x e
     * inicialização.
     */
    struct x x1 = { 1, 2, 3 }; // em ordem

    /* declaração de uma variável do tipo struct x e
     * inicialização usando designadores.
     */
    struct x x2 = { .a = 3, .b = 4, .c = 5 }; // a ordem não importa

    /* declaração de uma variável do tipo Data
     * (que deriva de uma estrutura) e inicilização.
     */
    Data d1 = { 23, 5, 2002 };

    /* declaração de uma variável do tipo Data
     * (que deriva de uma estrutura) e inicialização
     * usando designadores.
     */
    Data d2 = { .dia = 25, .mes = 2, .ano = 1985 };

    // declaração de um array com 5 datas
    Data dArray[5];

    /* d2 é copiada, membro a membro para
     * dataNascimento de p1.
     */
    Pessoa p1 = {
        .nomeCompleto = "David Buzatto",
        .peso = 120,
        .dataNascimento = d2
    };

    // p1 é copiado, membro a membro para p2
```

```
92      Pessoa p2 = p1;
93      p2.peso = 90;
94
95      // criando uma nova pessoa através de uma função
96      Pessoa p3 = novaPessoa( "Maria da Silva", 55, &d1 );
97
98      /* declarando uma nova pessoa, que receberá os dados pela
99       * entrada.
100      */
101     Pessoa p4;
102
103     /* inicializando os membros de uma instância da
104      * estrutura anônima.
105      */
106     v1.a = 5;
107     v1.b = 5;
108     v1.c = 5;
109
110     imprimirX( x1 );
111     printf( "\n" );
112
113     imprimirX( x2 );
114     printf( "\n" );
115
116     imprimirData( d1 );
117     printf( "\n" );
118
119     imprimirDataPonteiro( &d2 );
120     printf( "\n" );
121
122     imprimirPessoa( &p1 );
123     printf( "\n" );
124
125     imprimirPessoa( &p2 );
126     printf( "\n" );
127
128     imprimirPessoa( &p3 );
129     printf( "\n" );
130
131     printf( "Entre com o nome: " );
132     fgets( p4.nomeCompleto, 40, stdin );
133     p4.nomeCompleto[strlen(p4.nomeCompleto)-1] = '\0';
```

```
134
135     printf( "Entre com o peso: " );
136     scanf( "%f", &p4.peso );
137
138     printf( "Data de nascimento:\n" );
139     printf( "    dia: " );
140     scanf( "%d", &p4.dataNascimento.dia );
141     printf( "    mes: " );
142     scanf( "%d", &p4.dataNascimento.mes );
143     printf( "    ano: " );
144     scanf( "%d", &p4.dataNascimento.ano );
145
146     imprimirPessoa( &p4 );
147     printf( "\n" );
148
149     for ( int i = 0; i < 5; i++ ) {
150         printf( "Data %d\n", i+1 );
151         printf( "    dia: " );
152         scanf( "%d", &dArray[i].dia );
153         printf( "    mes: " );
154         scanf( "%d", &dArray[i].mes );
155         printf( "    ano: " );
156         scanf( "%d", &dArray[i].ano );
157     }
158
159     printf( "Datas:\n" );
160     for ( int i = 0; i < 5; i++ ) {
161         printf( "    Data %d: ", i+1 );
162         imprimirData( dArray[i] );
163         printf( "\n" );
164     }
165
166     return 0;
167
168 }
169
170 void imprimirX( struct x p ) {
171     printf( "%d %d %d", p.a, p.b, p.c );
172 }
173
174 void imprimirData( Data d ) {
175     printf( "%02d/%02d/%04d", d.dia, d.mes, d.ano );
```

```
176 }
177
178 void imprimirDataPonteiro( Data *d ) {
179
180     // d->dia   =>   (*d).dia, ou seja, o operador -> é um atalho
181     printf( "%02d/%02d/%04d", d->dia, d->mes, (*d).ano );
182
183     /* pode-se usar uma expressão com -> para receber valor, por
184      * exemplo d->dia = 25; (o membro dia da estrutura apontada
185      * por d recebe o valor 25).
186      */
187
188 }
189
190 Pessoa novaPessoa( const char *nc, float p, Data *dn ) {
191
192     Pessoa pe;
193
194     // copia a string
195     strcpy( pe.nomeCompleto, nc );
196
197     // copia o peso
198     pe.peso = p;
199
200     // copia, membro a membro, os dados de dn
201     pe.dataNascimento = *dn;
202
203     return pe;
204
205 }
206
207 void imprimirPessoa( Pessoa *p ) {
208
209     printf( "%s, nascido(a) em ", p->nomeCompleto );
210     imprimirDataPonteiro( &p->dataNascimento );
211     printf( " e tem peso = %.2fkg", p->peso );
212
213 }
```

10.2 Exercícios

Exercício 10.1 (KING, 2008): Escreva um programa que leia duas datas e que as apresente em ordem crescente. Cada data deve ser armazenada usando o tipo Data que contém três membros inteiros: dia, mês e ano. A comparação entre as datas deve ser feita utilizando a função `compararData`. Se `d1` for menor que `d2`, um valor negativo deve ser retornado. Se `d1` for maior que `d2`, um valor positivo deve ser retornado. Se `d1` e `d2` forem a mesma data, 0 deve ser retornado. A impressão das datas deve ser feita utilizando a função `imprimirData`. As definições das funções são:

- `int compararData( const Data *d1, const Data *d2 )`
- `void imprimirData( const Data *data )`

Arquivo com a solução: ex10.1.c

Entrada

```
Data 1
    dia: 1
    mes: 2
    ano: 1990
Data 2
    dia: 1
    mes: 2
    ano: 2000
```

Saída

```
01/02/1990 <= 01/02/2000
```

Entrada

```
Data 1
    dia: 20
    mes: 3
    ano: 2000
Data 2
    dia: 28
    mes: 2
    ano: 2000
```

Saída

```
28/02/2000 <= 20/03/2000
```

Entrada

```
Data 1
    dia: 25
    mes: 2
    ano: 2019
Data 2
    dia: 24
    mes: 2
    ano: 2019
```

Saída

```
24/02/2019 <= 25/02/2019
```

Entrada

```
Data 1
    dia: 30
    mes: 1
    ano: 2000
Data 2
    dia: 30
    mes: 1
    ano: 2000
```

Saída

```
30/01/2000 <= 30/01/2000
```

Exercício 10.2 (KING, 2008): Escreva um programa que leia uma data e que a partir da mesma obtenha o dia do ano correspondente. Cada data deve ser armazenada usando o tipo Data que contém três membros inteiros: dia, mês e ano. O cálculo do dia do ano deve ser feito utilizando a função `diaDoAno`. A definição das função é:

- `int diaDoAno( const Data *data )`

Arquivo com a solução: ex10.2.c

Entrada

```
dia: 1
mes: 4
ano: 2014
```

Saída

```
O dia do ano da data 01/04/2014 eh 91.
```

Entrada

```
dia: 1
mes: 4
ano: 2016
```

Saída

```
O dia do ano da data 01/04/2016 eh 92.
```

Exercício 10.3 (KING, 2008): Escreva um programa que leia uma quantidade de segundos e que, a partir desse valor, gere a quantidade de horas, minutos e segundos correspondentes, gerando como resultado uma instância do tipo Hora, que contém três membros inteiros: hora, minuto e segundo. Esse cálculo deve ser feito através da função `gerarHora`. A impressão da hora deve ser feita utilizando a função `imprimirHora`. As definições das funções são:

- `Hora gerarHora( int quantidadeSegundos )`
- `void imprimirHora( const Hora *hora )`

Arquivo com a solução: ex10.3.c

Entrada

```
Segundos: 254783
```

Saída

```
Hora correspondente: 70:46:23
```

Exercício 10.4 (KING, 2008): Escreva um programa que leia dois números complexos, representados pelo tipo Complexo, e que calcule a soma dos mesmos. O tipo Complexo

contém dois membros decimais: real e imaginário. A soma dos números complexos deve ser feita através da função `somar` que recebe dois números complexos e que retorna um novo número complexo que representa a soma dos números passados. A impressão do valor resultante deve ser feita através da função `imprimirComplexo`. As definições das funções são:

- `Complexo somar( const Complexo *c1, const Complexo *c2 )`
- `void imprimirComplexo( const Complexo *c )`

Arquivo com a solução: ex10.4.c

Entrada

```
Complexo 1
    Parte real: 10
    Parte imaginaria: 5
Complexo 2
    Parte real: 3
    Parte imaginaria: 9
```

Saída

```
(10.00 + 5.00i) + (3.00 + 9.00i) = (13.00 + 14.00i)
```

Exercício 10.5 (KING, 2008): Escreva um programa que leia duas frações, representados pelo tipo Fracao, e que calcule sua soma, subtração, multiplicação e divisão. O tipo Fracao contém dois membros decimais: numerador e denominador. As operações com as frações devem ser feitas através das funções `somar`, `subtrair`, `multiplicar` e `dividir`. A soma e a subtração de frações envolve a obtenção do mínimo múltiplo comum, entretanto, para simplificar a implementação, caso os denominadores sejam diferentes, calcule o denominador comum como a multiplicação dos dois denominadores. Além disso, as frações não precisam ser simplificadas. Essas quatro funções recebem duas frações e retornam uma nova fração com o resultado esperado. A impressão das frações resultantes deve ser feita através da função `imprimirFracao`. As definições das funções são:

- `Fracao somar( const Fracao *f1, const Fracao *f2 )`
- `Fracao subtrair( const Fracao *f1, const Fracao *f2 )`
- `Fracao multiplicar( const Fracao *f1, const Fracao *f2 )`
- `Fracao dividir( const Fracao *f1, const Fracao *f2 )`
- `void imprimirFracao( const Fracao *f )`

Arquivo com a solução: ex10.5.c

Entrada

```
Fracao 1
    Numerador: 1
    Denominador: 2
Fracao 2
    Numerador: 2
    Denominador: 3
```

Saída

```
1.00/2.00 + 2.00/3.00 = 7.00/6.00
1.00/2.00 - 2.00/3.00 = -1.00/6.00
1.00/2.00 * 2.00/3.00 = 2.00/6.00
1.00/2.00 / 2.00/3.00 = 3.00/4.00
```

Exercício 10.6 (KING, 2008): Escreva um programa que leia os componentes vermelho, verde e azul que são usados para representar uma cor e que crie uma instância do tipo Cor através da função `novaCor`. Essa função deve validar a entrada, ou seja, cada um dos componentes da cor deve estar, obrigatoriamente, no intervalo de 0 a 255 inclusive. Caso um valor menor que zero seja fornecido, o valor zero deve ser atribuído ao componente respectivo. Caso um valor maior de 255 seja fornecido, o valor 255 será atribuído. Utilize a função `imprimirCor` para imprimir os dados da cor. As definições das funções são:

- `Cor novaCor( int vermelho, int verde, int azul )`
- `void imprimirCor( const Cor *c )`

Arquivo com a solução: ex10.6.c

Entrada

```
Vermelho: 50
Verde: 60
Azul: 70
```

Saída

```
Cor: rgb( 50, 60, 70 )
```

Entrada

```
Vermelho: -10
Verde: -10
Azul: -10
```

Saída

```
Cor: rgb( 0, 0, 0 )
```

Entrada

```
Vermelho: 260
Verde: 180
Azul: 280
```

Saída

```
Cor: rgb( 255, 180, 255 )
```

Exercício 10.7 (KING, 2008): Com base no exercício anterior, escreva um programa que leia uma cor e que apresente os valores dos seus componentes utilizando as funções `getVermelho`, `getVerde` e `getAzul` que retornam, respectivamente, os componentes vermelho, verde e azul de uma cor. Utilize a função `novaCor` para criar a Cor base. As definições das funções são:

- `int getVermelho( const Cor *c )`
- `int getVerde( const Cor *c )`
- `int getAzul( const Cor *c )`

Arquivo com a solução: ex10.7.c

Entrada

```
Vermelho: 50
Verde: 60
Azul: 70
```

Saída

```
Cor: rgb( 50, 60, 70 )
getVermelho(): 50
getVerde(): 60
getAzul(): 70
```

Entrada

```
Vermelho: -10
Verde: -10
Azul: -10
```

Saída

```
Cor: rgb( 0, 0, 0 )
getVermelho(): 0
getVerde(): 0
getAzul(): 0
```

Entrada

```
Vermelho: 260
Verde: 180
Azul: 280
```

Saída

```
Cor: rgb( 255, 180, 255 )
getVermelho(): 255
getVerde(): 180
getAzul(): 255
```

Exercício 10.8 (KING, 2008): Com base no exercício anterior, escreva um programa que leia uma cor e reconfigure os seus membros utilizando as funções `setVermelho`, `setVerde` e `setAzul`. As mesmas validações da função `novaCor`, já implementada, se aplicam. Utilize a função `novaCor` para criar a Cor base. As definições das funções são:

- `void setVermelho( Cor *c, int vermelho )`

- `void setVerde( Cor *c, int verde )`
- `void setAzul( Cor *c, int azul )`

Arquivo com a solução: ex10.8.c

Entrada

```
Vermelho: 50
Verde: 60
Azul: 70
Novo vermelho: -1
Novo verde: 600
Novo azul: 190
```

Saída

```
Cor: rgb( 50, 60, 70 )
Cor alterada: rgb( 0, 255, 190 )
```

Exercício 10.9 (KING, 2008): Com base no exercício anterior, escreva um programa que leia uma cor e que gere uma versão mais escura dessa cor utilizando a função `escurecer`. A função `escurecer` deve multiplicar cada membro da cor por 0.7, truncando a parte decimal no resultado para a nova cor. Utilize a função `novaCor` para criar a Cor base. A definição da função é:

- `Cor escurecer( const Cor *c )`

Arquivo com a solução: ex10.9.c

Entrada

```
Vermelho: 255
Verde: 180
Azul: 90
```

Saída

```
Cor base: rgb( 255, 180, 90 )
Cor escurecida: rgb( 178, 125, 62 )
```

Exercício 10.10 (KING, 2008): Com base no exercício anterior, escreva um programa que leia uma cor e que gere uma versão mais clara dessa cor utilizando a função `clarear`. A função `clarear` deve dividir cada membro por 0.7, truncando a parte

decimal no resultado para a nova cor. Entretanto, existem alguns casos especiais que devem ser observados: 1) Se o valor de todos os componentes da cor original forem iguais a zero, a nova cor deve ser gerada com todos os componentes valendo 3; 2) Caso algum componente da cor original for maior que 0, mas menor que 3, deve-se configurá-lo como 3 na nova cor antes da divisão por 0,7; 3) Se a divisão por 0,7 de um membro da cor original resultar em algum valor maior que 255, deve-se configurar, na nova cor, esse componente com o valor 255. Utilize a função `novaCor` para criar a Cor base. A definição da função é:

- `Cor clarear( const Cor *c )`

Arquivo com a solução: ex10.10.c

Entrada

```
Vermelho: 10
Verde: 30
Azul: 40
```

Saída

```
Cor base: rgb( 10, 30, 40 )
Cor clareada: rgb( 14, 42, 57 )
```

Entrada

```
Vermelho: 0
Verde: 0
Azul: 0
```

Saída

```
Cor base: rgb( 0, 0, 0 )
Cor clareada: rgb( 3, 3, 3 )
```

Entrada

```
Vermelho: 1
Verde: 2
Azul: 1
```

Saída

```
Cor base: rgb( 1, 2, 1 )
Cor clareada: rgb( 4, 4, 4 )
```

Entrada

```
Vermelho: 100
Verde: 150
Azul: 230
```

Saída

```
Cor base: rgb( 100, 150, 230 )
Cor clareada: rgb( 142, 214, 255 )
```

Exercício 10.11 (KING, 2008): Escreva um programa que leia as coordenadas de dois pontos, armazenadas no tipo Ponto, que contém os membros inteiros: x e y. A partir dos dois pontos lidos, uma instância do tipo Retangulo, que contém dois membros do tipo Ponto (`superiorEsquerdo` e `inferiorDireito`) deve ser criada, usando a função `novoRetangulo`, e sua área deve ser calculada pela função `calcularArea`. Para imprimir o Retangulo, utilize a função `imprimirRetangulo`. Considere que o sistema de coordenadas adotado é o mesmo de um plano cartesiano tradicional. A apresentação de valores inteiros com sinal deve ser feita usando a opção + no especificador de formato %d. Por exemplo, %+02d formata um inteiro, com no mínimo duas casas, preenchendo com zeros se necessário, além de exibir o sinal. As definições das funções são:

- `Retangulo novoRetangulo( const Ponto *sEsq, const Ponto *iDir )`
- `int calcularArea( const Retangulo *r )`
- `void imprimirRetangulo( const Retangulo *r )`

Arquivo com a solução: ex10.11.c

Entrada

```
Ponto superior esquerdo
    x: 10
    y: 40
Ponto inferior direito
    x: 60
    y: 10
```

Saída

```
(+10, +40) =====|
|               |
|               |
|===== (+60, +10)
Area: 1500
```

Entrada

```
Ponto superior esquerdo
    x: -60
    y: -10
Ponto inferior direito
    x: -10
    y: -50
```

Saída

```
(-60, -10) =====|
|               |
|               |
|===== (-10, -50)
Area: 2000
```

Entrada

```
Ponto superior esquerdo
    x: -60
    y: 30
Ponto inferior direito
    x: 60
    y: -30
```

Saída

```
(-60, +30) =====|
|               |
|               |
|===== (+60, -30)
Area: 7200
```

Exercício 10.12 (KING, 2008): Com base no exercício anterior, escreva um programa que crie uma instância do tipo Retangulo e que calcule seu ponto central, utilizando, para isso, a função `obterCentro`. Considere que o sistema de coordenadas adotado é o mesmo de um plano cartesiano tradicional. A definição da função é:

- `Ponto obterCentro( const Retangulo *r )`

Arquivo com a solução: ex10.12.c

Entrada

```
Ponto superior esquerdo
    x: 10
    y: 40
Ponto inferior direito
    x: 60
    y: 10
```

Saída

```
(+10, +40) =====|
|               |
|               |
|===== (+60, +10)
Centro: (+35, +25)
```

Entrada

```
Ponto superior esquerdo
    x: -60
    y: -10
Ponto inferior direito
    x: -10
    y: -50
```

Saída

```
(-60, -10) =====|
|               |
|               |
|===== (-10, -50)
Centro: (-35, -30)
```

Entrada

```
Ponto superior esquerdo
    x: -60
    y: 30
Ponto inferior direito
    x: 60
    y: -30
```

Saída

```
(-60, +30) =====|
|               |
|               |
|===== (+60, -30)
Centro: (+0, +0)
```

Exercício 10.13 (KING, 2008): Com base no exercício anterior, escreva um programa que crie uma instância do tipo Retangulo e a mova em uma quantidade arbitrária de unidades em x e em y, utilizando, para isso, a função `mover`. Considere que o sistema de coordenadas adotado é o mesmo de um plano cartesiano tradicional. A definição da função é:

- `void mover( Retangulo *r, int x, int y )`

Arquivo com a solução: ex10.13.c

Entrada

```
Ponto superior esquerdo
    x: 10
    y: 40
Ponto inferior direito
    x: 60
    y: 10
Mover em x: 50
Mover em y: -10
```

Saída

```
Retangulo original:
(+10, +40) =====|
|               |
|               |
|===== (+60, +10)
Retangulo movido:
(+60, +30) =====|
|               |
|               |
|===== (+110, +0)
```

Exercício 10.14 (KING, 2008): Com base no exercício anterior, escreva um programa que crie uma instância do tipo Retangulo e verifique se cinco pontos, também fornecidos pelo usuário, estão contidos ou não dentro desse retângulo. Para isso, utilize a função `contem`. Considere que o sistema de coordenadas adotado é o mesmo de um plano cartesiano tradicional. A definição da função é:

- `bool contem( const Retangulo *r, const Ponto *p )`

Arquivo com a solução: ex10.14.c

Entrada

```
Retangulo
Ponto superior esquerdo
    x: -60
    y: 30
Ponto inferior direito
    x: 60
    y: -30
Pontos
  Ponto 1
    x: -70
    y: 20
  Ponto 2
    x: -50
    y: 20
  Ponto 3
    x: 60
    y: 30
  Ponto 4
    x: 55
    y: -20
  Ponto 5
    x: 40
    y: -40
```

Saída

```
(-70, +20): nao contido!
(-50, +20): contido!
(+60, +30): contido!
(+55, -20): contido!
(+40, -40): nao contido!
```

Exercício 10.15: Com base no exercício anterior, escreva um programa que crie duas instâncias do tipo Retangulo e verifique se os dois retângulos representados por essas instâncias se interceptam. Para isso, utilize a função `intercepta`. Considere que o sistema de coordenadas adotado é o mesmo de um plano cartesiano tradicional. Dica: você pode utilizar a função contem do exercício anterior. A definição da função é:

- `bool intercepta( const Retangulo *r1, const Retangulo *r2 )`

Arquivo com a solução: ex10.15.c

Entrada

```
Retangulo 1
Ponto superior esquerdo
    x: 10
    y: 40
Ponto inferior direito
    x: 60
    y: 10
Retangulo 2
Ponto superior esquerdo
    x: 30
    y: 50
Ponto inferior direito
    x: 50
    y: 20
```

Saída

```
Os retangulos se interceptam!
```

Entrada

```
Retangulo 1
Ponto superior esquerdo
    x: 10
    y: 40
Ponto inferior direito
    x: 60
    y: 10
Retangulo 2
Ponto superior esquerdo
    x: -30
    y: 60
Ponto inferior direito
    x: 20
    y: -10
```

Saída

```
Os retangulos se interceptam!
```

Entrada

```
Retangulo 1
Ponto superior esquerdo
    x: 10
    y: 40
Ponto inferior direito
    x: 60
    y: 10
Retangulo 2
Ponto superior esquerdo
    x: 30
    y: 100
Ponto inferior direito
    x: 60
    y: 50
```

Saída

```
Os retangulos nao se interceptam!
```

CAPÍTULO 11

UNIÕES E ENUMERAÇÕES

"Mas ainda uma união dividida".

William Shakespeare

AS uniões funcionam de forma parecida com as estruturas, ou seja, permitem que mais de um membro seja agrupado para a definição de um tipo. Ao contrário das estruturas, onde cada membro tem sua região de memória delimitada, nas uniões o compilador aloca a memória necessária para armazenar apenas o maior dos membros, fazendo com que a configuração de membros diferentes interfira nos dados dos outros membros. Apesar de parecer contraintuitivo, o uso de uniões pode ser feito quando há a necessidade de economizar memória. Já as enumerações servem para definir variáveis que terão um conjunto limitado de valores. Veja os exemplos a seguir.

11.1 Exemplos em Linguagem C

Definição e uso de uniões

```
1 /*
2  * Arquivo: Unioes.c
3  * Autor: Prof. Dr. David Buzatto
```

```c
 */

#include <stdio.h>
#include <stdlib.h>

/*
 * union anônima
 * membros a, b e c, do tipo inteiro
 */
union {
    int a;
    int b;
    int c;
} v1, v2 = { 4 }, vn = { .b = 3 };
// na linha acima: só é permitido inicializar um membro!

/*
 * union chamada x
 * membros a, b e c do tipo inteiro
 */
union x {
    int a;
    int b;
    int c;
};

/*
 * definição de tipo (Numero) usando
 * uma união
 */
typedef union {
    int inteiro;
    float decimal;
} Numero;

/*
 * definição de tipo (ItemCatalogo) usando
 * estruturas e uniões
 */
typedef struct {
    int identificador;
    float preco;
```

```
46      int tipoDoItem;
47      union {
48          struct {
49              char titulo[41];
50              char autor[41];
51              int quantidadePaginas;
52          } livro;
53          struct {
54              char formato[41];
55          } caneca;
56          struct {
57              char formato[41];
58              int cor;
59              char tamanho[4];
60          } camiseta;
61      } item;
62  } ItemCatalogo;
63
64  void imprimirX( union x p );
65  void imprimirNumero( Numero n );
66  void imprimirNumeroPonteiro( Numero *n );
67  void imprimirItemCatalogo( ItemCatalogo *i );
68
69  int main() {
70
71      // declaração de uma variável do tipo union x e
72      // inicialização. só é permitido inicializar UM membro.
73      union x x1 = { 1 }; // em ordem
74
75      // declaração de uma variável do tipo union x e
76      // inicialização. só é permitido inicializar UM membro.
77      union x x2 = { .b = 4 }; // ordem não importa
78
79      // declaração de uma variável do tipo Numero
80      // (que deriva de uma união) e inicilização
81      Numero n1 = { 23 };
82
83      // declaração de uma variável do tipo Numero
84      // (que deriva de uma união) e inicilização
85      // usando designadores
86      Numero n2 = { .decimal = 4.5 };
87
```

```
88      // declaração e inicialização de quatro instâncias de ItemCatalogo
89      ItemCatalogo i1 = {
90          .identificador = 104,
91          .preco = 49.99,
92          .tipoDoItem = 1,
93          .item.livro.titulo = "C Programming: a modern approach",
94          .item.livro.autor = "King, K. N.",
95          .item.livro.quantidadePaginas = 805
96      };
97
98      ItemCatalogo i2 = {
99          .identificador = 205,
100         .preco = 20.50,
101         .tipoDoItem = 2,
102         .item.caneca.formato = "cafe"
103     };
104
105     ItemCatalogo i3 = {
106         .identificador = 174,
107         .preco = 29.99,
108         .tipoDoItem = 3,
109         .item.camiseta.formato = "Gola V",
110         .item.camiseta.cor = 20,
111         .item.camiseta.tamanho = "GG"
112     };
113
114     ItemCatalogo i4 = {
115         .identificador = 421,
116         .preco = 34.99,
117         .tipoDoItem = 4,
118         .item.camiseta.formato = "Gola Polo",
119         .item.camiseta.cor = 15,
120         .item.camiseta.tamanho = "M"
121     };
122
123     // inicializando os membros de uma instância da
124     // união anônima
125     v1.c = 5;
126
127     imprimirX( x1 );
128     printf( "\n" );
129
```

```
130     imprimirX( x2 );
131     printf( "\n" );
132 
133     imprimirNumero( n1 );
134     printf( "\n" );
135 
136     imprimirNumeroPonteiro( &n2 );
137     printf( "\n\n" );
138 
139     imprimirItemCatalogo( &i1 );
140     printf( "\n\n" );
141 
142     imprimirItemCatalogo( &i2 );
143     printf( "\n\n" );
144 
145     imprimirItemCatalogo( &i3 );
146     printf( "\n\n" );
147 
148     imprimirItemCatalogo( &i4 );
149     printf( "\n" );
150 
151     return 0;
152 
153 }
154 
155 void imprimirX( union x p ) {
156     printf( "%d %d %d", p.a, p.b, p.c );
157 }
158 
159 void imprimirNumero( Numero n ) {
160     printf( "%d - %.2f", n.inteiro, n.decimal );
161 }
162 
163 void imprimirNumeroPonteiro( Numero *n ) {
164     printf( "%d - %.2f", n->inteiro, n->decimal );
165 }
166 
167 void imprimirItemCatalogo( ItemCatalogo *i ) {
168 
169     printf( "Item do Catalogo: %d (R$%.2f)\n",
170             i->identificador, i->preco );
171 
```

```
172     switch ( i->tipoDoItem ) {
173         case 1:
174             printf( "Livro:\n" );
175             printf( "     Titulo: %s\n", i->item.livro.titulo );
176             printf( "      Autor: %s\n", i->item.livro.autor );
177             printf( "    Paginas: %d", i->item.livro.quantidadePaginas );
178             break;
179         case 2:
180             printf( "Caneca:\n" );
181             printf( "    Formato: %s", i->item.caneca.formato );
182             break;
183         case 3:
184             printf( "Camiseta:\n" );
185             printf( "    Formato: %s\n", i->item.camiseta.formato );
186             printf( "        Cor: %d\n", i->item.camiseta.cor );
187             printf( "    Tamanho: %s", i->item.camiseta.tamanho );
188             break;
189         default:
190             printf( "Tipo de item desconhecido!" );
191             break;
192     }
193
194 }
```

Definição e uso de enumerações

```
1  /*
2   * Arquivo: Enumeracoes.c
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <string.h>
9
10 /* os valores enumerados das enumerações são definidos
11  * normalmente usando somente letras maiúsculas.
12  */
13
14 /*
15  * enum anônima
```

```
16  * valores enumerados: VERMELHO, VERDE e AZUL
17  */
18 enum {
19     VERMELHO,
20     VERDE,
21     AZUL
22 } e1, e2, en;
23
24 /*
25  * enum chamada tamanho
26  * valores enumerados: P, M, G, GG, EG
27  */
28 enum tamanho {
29     P,
30     M,
31     G,
32     GG,
33     EG
34 };
35
36 /*
37  * definição de tipo (Naipe) usando
38  * uma enumeracao
39  */
40 typedef enum {
41     COPAS,
42     OUROS,
43     PAUS,
44     ESPADAS
45 } Naipe;
46
47 /*
48  * definição de tipo (Carta) usando
49  * estrutura e enumeração.
50  */
51 typedef struct {
52     int valor;
53     Naipe naipe;
54 } Carta;
55
56 /*
57  * definição de tipo (Baralho) usando
```

```
 * um array de estrutura.
 */
typedef struct {
    Carta cartas[52];
} Baralho;

Baralho novoBaralho();
void imprimirCarta( Carta *c );
void imprimirBaralho( Baralho *b );

int main() {

    Baralho b = novoBaralho();

    imprimirBaralho( &b );

    return 0;

}

Baralho novoBaralho() {

    Baralho b;
    Naipe naipes[4] = { COPAS, OUROS, PAUS, ESPADAS };
    int valores[13] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };
    int c = 0;

    for ( int i = 0; i < 4; i++ ) {
        for ( int j = 0; j < 13; j++ ) {
            b.cartas[c].naipe = naipes[i];
            b.cartas[c].valor = valores[j];
            c++;
        }
    }

    return b;

}

void imprimirCarta( Carta *c ) {

    char valor[3];
```

```
100     char naipe;
101
102     switch ( c->valor ) {
103         case 1:
104             strcpy( valor, "A" );
105             break;
106         case 2:
107             strcpy( valor, "2" );
108             break;
109         case 3:
110             strcpy( valor, "3" );
111             break;
112         case 4:
113             strcpy( valor, "4" );
114             break;
115         case 5:
116             strcpy( valor, "5" );
117             break;
118         case 6:
119             strcpy( valor, "6" );
120             break;
121         case 7:
122             strcpy( valor, "7" );
123             break;
124         case 8:
125             strcpy( valor, "8" );
126             break;
127         case 9:
128             strcpy( valor, "9" );
129             break;
130         case 10:
131             strcpy( valor, "10" );
132             break;
133         case 11:
134             strcpy( valor, "J" );
135             break;
136         case 12:
137             strcpy( valor, "Q" );
138             break;
139         case 13:
140             strcpy( valor, "K" );
141             break;
```

```
142     }
143
144     switch ( c->naipe ) {
145         case COPAS:
146             naipe = 'C';
147             break;
148         case OUROS:
149             naipe = 'O';
150             break;
151         case PAUS:
152             naipe = 'P';
153             break;
154         case ESPADAS:
155             naipe = 'E';
156             break;
157     }
158
159     printf( "%s(%c)", valor, naipe );
160
161 }
162
163 void imprimirBaralho( Baralho *b ) {
164
165     for ( int i = 1; i <= 52; i++ ) {
166         imprimirCarta( &(b->cartas[i-1]) );
167         printf( " " );
168         if ( i % 13 == 0 ) {
169             printf( "\n" );
170         }
171     }
172
173 }
```

Geração de números pseudoaleatórios

```
1 /*
2  * Arquivo: Rand.c
3  * Autor: Prof. Dr. David Buzatto
4  */
5
6 /* Alguns tipos de programas, jogos por exemplo,
```

```
 * necessitam gerar valores aleatórios para que
 * alguma ação seja executada. Por exemplo, embaralhar
 * uma série de cartas, ou sortear qual a chance
 * de um ataque conectar em um oponente.
 *
 * Neste trecho de código você aprenderá a gerar
 * números pseudoaleatórios em C.
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* time.h é o include necessário para realizar a
 * semeadura do gerador de números pseudoaleatórios.
 */

/* função para gerar números aletatórios
 * dentro de um intervalo fechado.
 */
int randIntervalo( int inicio, int fim );

int main() {

    /* void srand( unsigned semente )
     * int rand()
     *
     * Na linguagem C, usa-se as funções srand e rand
     * para se gerar números inteiros pseudoaletórios/
     * pseudorandômicos.
     * Ambas são definidas no cabeçalho stdlib.h.
     *
     * A função srand é usada para semear o gerador de
     * números pseudoaleatórios. Caso ela não seja
     * invocada ou seja invocada com um valor fixo,
     * a ordem dos números gerados será sempre a mesma.
     *
     * Sendo assim, tentamos simular a diferença de valor
     * de execução passando algum valor que muda a cada
     * execução do programa. Isso pode ser feito
     * utilizando a função time, passando NULL como
     * parâmetro, o que que retornará um valor
     * que representa o tempo atual do sistema.
```

```
49      *
50      * Quando seu programa precisar usar o gerador de
51      * números pseudoaleatórios, lembre-se de invocar
52      * srand APENAS UMA VEZ, normalmente, no início
53      * da função main.
54      */
55     srand( time( NULL ) );
56
57     /* Após a semeadura, cada invocação à rand gerará
58      * um número pseudoaletório no intervalo de 0 a
59      * RAND_MAX, macro definida em stdlib.h que expande
60      * para um inteiro. Esse valor é dependente de
61      * implementação, mas a garantia é que seja, no
62      * mínimo, 32767.
63      * https://en.cppreference.com/w/c/numeric/random/RAND_MAX
64      */
65     printf( "0 a %d: %d\n", RAND_MAX, rand() );
66
67     /* Entretanto, normalmente precisamos que os valores
68      * sejam gerados em um intervalo definido. Para isso
69      * calculamos o resto da divisão inteira do valor
70      * gerado pelo último valor do intervalo desejado.
71      *
72      * No exemplo abaixo, serão gerados 10 valores no
73      * intervalo de 0 a 19.
74      */
75     for ( int i = 0; i < 10; i++ ) {
76         printf( "0 a 19: %d\n", rand() % 20 );
77     }
78
79     /* A geração de valores aleatórios em um intervalo
80      * fechado é feita abaixo, usando a função implementada
81      * a seguir.
82      */
83     for ( int i = 0; i < 10; i++ ) {
84         printf( "5 a 25: %d\n", randIntervalo( 5, 25 ) );
85     }
86
87     return 0;
88
89 }
90
```

```
91 int randIntervalo( int inicio, int fim ) {
92     return inicio + ( rand() % ( fim - inicio + 1 ) );
93 }
```

11.2 Exercícios

Exercício 11.1: Considere os tipos a seguir:

```
typedef enum {
    RETANGULO,
    CIRCULO
} TipoForma;

typedef struct {
    int x;
    int y;
} Ponto;

typedef struct {
    TipoForma tipo;
    Ponto centro;
    union {
        struct {
            int altura;
            int largura;
        } retangulo;
        struct {
            int raio;
        } circulo;
    } geom;
} Forma;
```

Escreva um programa que leia os valores de um Retângulo e de um Círculo e que calcule suas áreas, que os movam, que os redimensione e que imprima os seus dados. Para isso, implemente as funções:

- `int calcularArea( const Forma *f )`
- `void mover( Forma *f, int x, int y )`
- `Forma redimensionar( const Forma *f, float fator )`
- `void imprimirForma( const Forma *f )`

Arquivo com a solução: ex11.1.c

Entrada

```
Dados do retangulo:
  Centro:
    x: 30
    y: 10
  Largura: 20
  Altura: 10
Dados do circulo:
  Centro:
    x: 30
    y: 30
  Raio: 10
Apos a criacao, mover em:
  x: 10
  y: 20
Apos mover, redimensionar pelo fator: 2
```

Saída

```
Original:
===== Retangulo ======
| Centro: (+30, +10) |
| Largura: 20        |
| Altura: 10         |
======================
  Area: 200
======================
====== Circulo =======
| Centro: (+30, +30) |
| Raio: 10           |
======================
  Area: 314
======================

Apos mover:
===== Retangulo ======
| Centro: (+40, +30) |
| Largura: 20        |
| Altura: 10         |
======================
  Area: 200
======================
====== Circulo =======
| Centro: (+40, +50) |
| Raio: 10           |
======================
  Area: 314
======================

Apos redimensionar:
===== Retangulo ======
| Centro: (+50, +35) |
| Largura: 40        |
| Altura: 20         |
======================
  Area: 800
======================
====== Circulo =======
| Centro: (+50, +60) |
| Raio: 20           |
======================
  Area: 1256
======================
```

CAPÍTULO 12

ORGANIZAÇÃO DE CÓDIGO

"*divide et impera*"

"*divide ut regnes*"

"*dividir para conquistar*"

A linguagem de programação C podemos, assim como na maioria, senão em todas as linguagens de programação, dividir nosso código fonte em diversos arquivos, visando organizá-lo de modo a poder reaproveitá-lo. Neste Capítulo não haverá exercício, mas o exemplo a seguir ilustra a divisão de arquivos de um projeto. Vários comentários serão inseridos dentro do código para que você possa entender o que deve ser feito. Em aula será mostrado como fazer com que tal divisão seja adequadamente compilada e executada na ferramenta adotada.

12.1 Exemplos em Linguagem C

Arquivo main.c, contém a função main

```c
/*
 * Arquivo: OrganizacaoDeCodigo/main.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>
#include "geom.h"
/* na linha acima: incluindo o cabeçalho que
 * será usado. Note o uso de aspas duplas,
 * indicando que o arquivo geom.h está no mesmo
 * diretório que o arquivo main.c
 */

int main() {

    Ponto p;
    Linha l;
    Retangulo r;
    Elipse e;
    Circulo c;


    p.x = 10;
    p.y = -30;

    l.ini.x = 20;
    l.ini.y = 30;
    l.fim.x = 40;
    l.fim.y = 20;

    r.canto.x = 10;
    r.canto.y = 20;
    r.largura = 20;
    r.altura = 30;

    e.canto.x = 30;
    e.canto.y = 40;
```

```
39      e.largura = 30;
40      e.altura = 10;
41
42      c.centro.x = 50;
43      c.centro.y = 80;
44      c.raio = 5;
45
46
47      printf( "Ponto: " );
48      imprimirPonto( &p );
49      printf( "\n" );
50
51      printf( "Linha: " );
52      imprimirLinha( &l );
53      printf( "\n" );
54
55      printf( "Retangulo: " );
56      imprimirRetangulo( &r );
57      printf( "\n" );
58
59      printf( "Elipse: " );
60      imprimirElipse( &e );
61      printf( "\n" );
62
63      printf( "Circulo: " );
64      imprimirCirculo( &c );
65      printf( "\n" );
66
67      return 0;
68
69  }
```

Arquivo geom.h, contém a declaração de tipos, funções etc.

```
1  /*
2   * Arquivo: OrganizacaoDeCodigo/geom.h
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  /* as condições de guarda evitam que o mesmo
7   * cabeçalho seja inserido mais de uma vez em
```

```
 * algum arquivo.
 */
#ifndef GEOM_H_INCLUDED
#define GEOM_H_INCLUDED

/* nos arquivos de cabeçalho (extensão .h),
 * usualmente, haverá as seguintes
 * entidades:
 *
 * - definição de macros;
 * - declaração (protótipos) de funções;
 * - declaração estruturas, uniões e enumerações
 * - definião de tipos.
 *
 * ou seja, o cabeçalho é usado, usualmente,
 * para declarações e não implementações.
 */

typedef struct {
    int x;
    int y;
} Ponto;

typedef struct {
    Ponto ini;
    Ponto fim;
} Linha;

typedef struct {
    Ponto canto;
    int largura;
    int altura;
} Retangulo;

typedef struct {
    Ponto canto;
    int largura;
    int altura;
} Elipse;

typedef struct {
    Ponto centro;
```

```
50      int raio;
51  } Circulo;
52
53  void imprimirPonto( Ponto *ponto );
54  void imprimirLinha( Linha *linha );
55  void imprimirRetangulo( Retangulo *retangulo );
56  void imprimirElipse( Elipse *elipse );
57  void imprimirCirculo( Circulo *circulo );
58
59  #endif // GEOM_H_INCLUDED
```

Arquivo geom.c, contém a definição de funções etc.

```
1   /*
2    * Arquivo: OrganizacaoDeCodigo/geom.c
3    * Autor: Prof. Dr. David Buzatto
4    */
5
6   /* cada arquivo de cabeçalho, usualmente,
7    * terá um arquivo de implementação/definição
8    * (extensão .c).
9    */
10  #include <stdio.h>
11  #include <stdlib.h>
12  #include "geom.h"
13  /* na linha acima: incluindo o cabeçalho que
14   * será implementado. Note o uso de aspas duplas,
15   * indicando que o arquivo geom.h está no mesmo
16   * diretório que o arquivo geom.c
17   */
18
19  /* no arquivo de implementação, as funções
20   * declaradas no cabelho e demais entidades
21   * de programação será utilizadas.
22   */
23
24  void imprimirPonto( Ponto *ponto ) {
25      printf( "(%+03d, %+03d)", ponto->x, ponto->y );
26  }
27
28  void imprimirLinha( Linha *linha ) {
```

```
29      imprimirPonto( &linha->ini );
30      printf( " ====== " );
31      imprimirPonto( &linha->fim );
32  }
33
34  void imprimirRetangulo( Retangulo *retangulo ) {
35      printf( "C: " );
36      imprimirPonto( &retangulo->canto );
37      printf( " L: %d, A: %d", retangulo->largura, retangulo->altura );
38  }
39
40  void imprimirElipse( Elipse *elipse ) {
41      printf( "C: " );
42      imprimirPonto( &elipse->canto );
43      printf( " L: %d, A: %d", elipse->largura, elipse->altura );
44  }
45
46  void imprimirCirculo( Circulo *circulo ) {
47      printf( "C: " );
48      imprimirPonto( &circulo->centro);
49      printf( " R: %d", circulo->raio );
50  }
```

CAPÍTULO 13

ARQUIVOS

"A memória dos velhos é menos pronta, porque o seu arquivo é muito extenso".

Marquês de Maricá

NA linguagem C, segundo King (2008), o termo *stream* (fluxo) está associado à ideia de um canal para qualquer fonte de dados de entrada ou qualquer destino para saída. Até agora nossos programas lidaram com um fluxo de entrada, associado ao teclado do computador, e um fluxo de saída, associado ao que vemos na tela do terminal. Em alguns casos, os programas que escrevemos precisam lidar com mais de um tipo de *stream* de entrada e/ou saída. Para acessarmos um *stream* na linguagem C nós utilizamos um ponteiro de arquivo `FILE *` e é disso que se trata esse Capítulo, ou seja, como acessar um *stream* para leitura ou escrita.

Na linguagem C existem três *streams* padrão, descritos na Tabela 13.1.

Ponteiro	*Stream*	Mapeamento padrão
`stdin`	Entrada padrão	Teclado
`stdout`	Saída padrão	Tela
`stderr`	Erro padrão	Tela

Tabela 13.1: *Streams* padrão

As funções de entrada e saída que temos usado até o momento (`printf`, `scanf`, `fgets`, `getchar` etc.) recebem dados ou direcionam dados, respectivamente, para os *streams* `stdin` e `stdout`. Para facilitar o entendimento, os *streams* que acessaremos serão *streams* de/para arquivos de texto cru (*raw*). As funções que utilizaremos para a manipulação de arquivos, bem como o tipo `FILE`, estão definidos nos cabeçalhos `stdio.h` e `stdlib.h`.

13.1 Exemplos em Linguagem C

Abertura de arquivo, leitura de conteúdo e impressão na tela

```c
/*
 * Arquivo: ArquivosAberturaLeitura.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {

    // declaração de um ponteiro para arquivo
    FILE *arquivo;
    char dadosLinha[80];
    int inteiro;
    float decimal;

    /* abrindo o arquivo "arquivoDados.txt", que está no mesmo diretório
     * do programa no modo somente leitura.
     *
     * Os modos de abertura para leitura e escrita de dados de texto são:
     *     - "r"  -> Abre para leitura (read), o arquivo precisa existir.
```

```
23      *     - "w"  -> Abre para escrita (write), o arquivo não
24      *              precisa existir.
25      *     - "a"  -> Abre para anexação (append), o arquivo não
26      *              precisa existir.
27      *     - "r+" -> Abre para leitura e escrita, começando do
28      *              início do arquivo.
29      *     - "w+" -> Abre para leitura e escrita, sobrescrevendo os dados
30      *              do arquivo caso exista.
31      *     - "a+" -> Abre para leitura e escrita, anexando os dados no
32      *              arquivo caso exista.
33      */
34     arquivo = fopen( "arquivoDados.txt", "r" );
35
36     /* adicionalmente, caso se deseje ler ou escrever dados binários
37      * em arquivos, os modos são, respetivamente:
38      * "rb", "wb", "ab",
39      * "r+b" ou "rb+",
40      * "w+b" ou "wb+" e
41      * "a+b" ou "ab+".
42      */
43
44     // verificando se o arquivo foi aberto
45     if ( arquivo != NULL ) {  // aberto
46
47         // lê uma linha do arquivo três vezes e a imprime
48         for ( int i = 0; i < 3; i++ ) {
49             fgets( dadosLinha, 80, arquivo );
50             dadosLinha[strlen(dadosLinha)-1] = '\0';
51             printf( "Linha lida: \"%s\"\n", dadosLinha );
52         }
53
54         // lê um inteiro do arquivo
55         fscanf( arquivo, "%d", &inteiro );
56         printf( "Inteiro lido: %d\n", inteiro );
57
58         // mais uma linha!
59         fgetc( arquivo ); // descarta o pulo de linha
60                           // que sobrou do fscanf!
61         fgets( dadosLinha, 80, arquivo );
62         dadosLinha[strlen(dadosLinha)-1] = '\0';
63         printf( "Linha lida: \"%s\"\n", dadosLinha );
64
```

```
65          // lê um float do arquivo
66          fscanf( arquivo, "%f", &decimal );
67          printf( "Decimal lido: %.2f\n", decimal );
68
69          /* uma forma de verificar se o fim do arquivo foi
70           * alcançado é utilizar a função feof (file
71           * end of file). Essa função retorna 0 (falso) caso
72           * o fim do arquivo ainda não foi encontrado ou
73           * um valor diferente de zero (verdadeiro) caso contrário.
74           */
75          printf( "Fim do arquivo? %s\n", feof(arquivo) ? "sim" : "nao" );
76
77          // mais uma linha!
78          fgetc( arquivo );
79          fgets( dadosLinha, 80, arquivo );
80          dadosLinha[strlen(dadosLinha)-1] = '\0';
81          printf( "Linha lida: \"%s\"\n", dadosLinha );
82
83          /* nesse ponto, devido aos dados do arquivo, não há
84           * mais dados para serem lidos!
85           */
86          printf( "Fim do arquivo? %s\n", feof(arquivo) ? "sim" : "nao" );
87
88      } else { // erro ao abrir
89          printf( "O arquivo nao pode ser aberto!" );
90      }
91
92      // fechando o arquivo (SEMPRE FECHE O(S) ARQUIVO(S) ABERTO(S)!!!)
93      fclose( arquivo );
94
95      return 0;
96
97  }
```

Leitura de um arquivo linha por linha

```
1  /*
2   * Arquivo: ArquivosLeituraTodasLinhas.c
3   * Autor: Prof. Dr. David Buzatto
4   */
5
```

```c
#include <stdio.h>
#include <stdlib.h>

int main() {

    FILE *arquivo;
    char dadosLinha[80];

    arquivo = fopen( "arquivoDados.txt", "r" );

    if ( arquivo != NULL ) {
        // lê o arquivo, linha por linha, e imprime na tela
        while ( !feof( arquivo ) ) {
            fgets( dadosLinha, 80, arquivo );
            printf( "%s", dadosLinha );
        }
    }

    fclose( arquivo );

    return 0;

}
```

Escrita e leitura de um arquivo

```c
/*
 * Arquivo: ArquivosEscrita.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void imprimeConteudo( FILE *file );

int main() {

    FILE *arquivo;
    char dados[80];
```

```
16      int inteiro;
17      float decimal;
18
19      arquivo = fopen( "arquivoEscrita.txt", "w" );
20
21      if ( arquivo != NULL ) {
22
23          printf( "Entre um uma frase: " );
24          fgets( dados, 80, stdin );
25          dados[strlen(dados)-1] = '\0';
26
27          printf( "Escrevendo no arquivo...\n" );
28
29          // escreve no arquivo usando fprintf
30          fprintf( arquivo, "%s", dados );
31
32          // escreve no arquivo um caractere
33          fputc( '\n', arquivo );
34
35          // escreve no arquivo uma string
36          fputs( "mais uma linha para o arquivo!\n", arquivo );
37
38
39          printf( "Entre com um inteiro: " );
40          scanf( "%d", &inteiro );
41
42          printf( "Entre com um decimal: " );
43          scanf( "%f", &decimal );
44
45          // escreve no arquivo usando fprintf
46          fprintf( arquivo, "inteiro escrito: %d\n", inteiro );
47          fprintf( arquivo, "decimal escrito: %f", decimal );
48
49
50      } else {
51          printf( "O arquivo nao pode ser aberto!" );
52      }
53
54      fclose( arquivo );
55
56      // imprimindo dados
57      imprimeConteudo( fopen( "arquivoEscrita.txt", "r" ) );
```

```
58
59
60      return 0;
61
62  }
63
64  void imprimeConteudo( FILE *a ) {
65
66      char dadosLinha[80];
67
68      if ( a != NULL ) {
69          while ( !feof( a ) ) {
70              fgets( dadosLinha, 80, a );
71              printf( "%s", dadosLinha );
72          }
73      }
74
75      fclose( a );
76
77  }
78
79
```

Vamos aos exercícios!

13.2 Exercícios

Os exercícios a seguir lidam com a leitura e escrita de arquivos de texto. Lembre-se de sempre inserir no pacote de código fonte da entrega os arquivos de dados.

Exercício 13.1: Escreva um programa que leia o arquivo de texto "`notas.txt`" e que calcule e exiba a média das notas armazenadas nesse arquivo.

Conteúdo do arquivo "`notas.txt`"

```
6
8
6
9
10
9.5
5
4
2
3.5
9.75
8
```

Arquivo com a solução: ex13.1.c

Saída

```
Media: 6.73
```

Exercício 13.2: Escreva um programa que leia o arquivo de texto "`barras.txt`" e exiba uma representação em barras, usando asteriscos, dos dados lidos.

Conteúdo do arquivo "`barras.txt`"

```
5
6
7
9
4
3
10
5
```

Arquivo com a solução: ex13.2.c

Saída

```
***** (5)
****** (6)
******* (7)
********* (9)
**** (4)
*** (3)
********** (10)
***** (5)
```

Exercício 13.3: Escreva um programa que:

1. Leia o arquivo de texto chamado "`numeros.txt`";
2. Para cada um dos cinco números lidos, o programa deve calcular o valor absoluto da diferença de cada um por 10;
3. Os valores da diferença que foram gerados, devem ser usados para desenhar triângulos de asteriscos;
4. Os dados desses triângulos devem ser armazenados em cinco arquivos diferentes (`tri1.txt`, `tri2.txt`, `tri3.txt`, `tri4.txt` e `tri5.txt`);
5. Por fim, o programa deve ler cada um desses arquivos e exibir os dados na tela.

Conteúdo do arquivo "`numeros.txt`"

```
7
6
5
8
4
```

Arquivo com a solução: ex13.3.c

Saída

```
Numero: 7
Absoluto da diferenca: 3
Gerando arquivo 'tri1.txt'...

Numero: 6
Absoluto da diferenca: 4
Gerando arquivo 'tri2.txt'...

Numero: 5
Absoluto da diferenca: 5
Gerando arquivo 'tri3.txt'...

Numero: 8
Absoluto da diferenca: 2
Gerando arquivo 'tri4.txt'...

Numero: 4
Absoluto da diferenca: 6
Gerando arquivo 'tri5.txt'...

Conteudo do arquivo 'tri1.txt':
*
**
***
Conteudo do arquivo 'tri2.txt':
*
**
***
****
Conteudo do arquivo 'tri3.txt':
*
**
***
****
*****
Conteudo do arquivo 'tri4.txt':
*
**
Conteudo do arquivo 'tri5.txt':
*
**
***
****
*****
******
```

CAPÍTULO 14

RECURSIVIDADE

"Para entender recursão, é preciso entender recursão".

recursividade é uma ferramenta essencial para a solução de diversos tipos de problemas computacionais. Algo que é definido em termos de si mesmo é chamado de recursivo. A seguir diversos exemplos serão apresentados.

14.1 Fatorial

14.1.1 Notação 1

$$n! = \begin{cases} 1, & \text{se } n \leq 1 \\ n(n-1)!, & \text{caso contrário} \end{cases} \quad n \in \mathbb{N}$$

14.1.2 Notação 2

$$fat(n) = \begin{cases} 1, & \text{se } n \leq 1 \\ n * fat(n-1), & \text{caso contrário} \end{cases} \quad n \in \mathbb{N}$$

14.1.3 Exemplo em Linguagem C

```c
int fat( int n ) {
    if ( n <= 1 ) {
        return 1;
    } else {
        return n * fat( n - 1 );
    }
}
```

14.2 Somatório

$$sum(n) = \begin{cases} 0, & \text{se } n = 0 \\ n + sum(n-1), & \text{caso contrário} \end{cases} \quad n \in \mathbb{N}$$

14.2.1 Exemplo em Linguagem C

```c
int sum( int n ) {
    if ( n == 0 ) {
        return 0;
    } else {
        return n + sum( n - 1 );
    }
}
```

14.3 Fibonacci

$$fib(n) = \begin{cases} 1, & \text{se } n \leq 1 \\ fib(n-2) + fib(n-1), & \text{caso contrário} \end{cases} \quad n \in \mathbb{N}$$

14.3.1 Exemplo em Linguagem C

```c
int fib( int n ) {
    if ( n <= 1 ) {
        return 1;
    } else {
        return fib( n - 2 ) + fib( n - 1 );
```

```
    }
}
```

14.4 Adição

Restrição: somar apenas de uma em uma unidade.

14.4.1 Notação 1

$$a+b=\begin{cases} a, & \text{se } b=0 \\ a+(b-1)+1, & \text{caso contrário} \end{cases} \quad a,b \in \mathbb{N}$$

14.4.2 Notação 2

$$add(a,b)=\begin{cases} a, & \text{se } b=0 \\ add(a,b-1)+1, & \text{caso contrário} \end{cases} \quad a,b \in \mathbb{N}$$

14.4.3 Exemplo em Linguagem C

```c
int add( int a, int b ) {
    if ( b == 0 ) {
        return a;
    } else {
        return add( a, b - 1 ) + 1;
    }
}
```

14.5 Subtração

Restrição: Subtrair apenas de uma em uma unidade.

14.5.1 Notação 1

$$a-b=\begin{cases} a, & \text{se } b=0 \\ a-(b-1)-1, & \text{caso contrário} \end{cases} \quad a,b \in \mathbb{N}$$

14.5.2 Notação 2

$$sub(a,b) = \begin{cases} a, & \text{se } b = 0 \\ sub(a, b-1) - 1, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

14.5.3 Exemplo em Linguagem C

```c
int sub( int a, int b ) {
    if ( b == 0 ) {
        return a;
    } else {
        return sub( a, b - 1 ) - 1;
    }
}
```

14.6 Multiplicação

Restrição: Somando qualquer quantidade.

14.6.1 Notação 1

$$a \cdot b = \begin{cases} 0, & \text{se } a = 0 \text{ ou } b = 0 \\ (a-1) \cdot b + b, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

14.6.2 Notação 2

$$mult(a,b) = \begin{cases} 0, & \text{se } a = 0 \text{ ou } b = 0 \\ mult(a-1, b) + b, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

14.6.3 Exemplo em Linguagem C

```c
int mult( int a, int b ) {
    if ( a == 0 || b == 0 ) {
        return 0;
    } else {
        return mult( a - 1, b ) + b;
    }
}
```

14.7 Divisão

Restrição: Subtraindo qualquer quantidade e somando apenas de uma em uma unidade.

14.7.1 Notação 1

$$\frac{a}{b} = \begin{cases} 0, & \text{se } a < b \\ \frac{a-b}{b} + 1, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

14.7.2 Notação 2

$$div(a,b) = \begin{cases} 0, & \text{se } a < b \\ div(a-b,b) + 1, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

14.7.3 Exemplo em Linguagem C

```c
int div( int a, int b ) {
    if ( a < b ) {
        return 0;
    } else {
        return div( a - b, b ) + 1;
    }
}
```

14.8 Resto

Restrição: Subtraindo qualquer quantidade.

14.8.1 Notação 1

$$a \,\%\, b = \begin{cases} a, & \text{se } a < b \\ (a-b) \,\%\, b, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

14.8.2 Notação 2

$$mod(a,b) = \begin{cases} a, & \text{se } a < b \\ mod(a-b,b), & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

14.8.3 Exemplo em Linguagem C

```c
int mod( int a, int b ) {
    if ( a < b ) {
        return a;
    } else {
        return mod( a - b, b );
    }
}
```

14.9 Exponenciação

Restrição: Multiplicando qualquer quantidade.

14.9.1 Notação 1

$$x^n = \begin{cases} 1, & \text{se } n = 0 \\ x \cdot x^{n-1}, & \text{caso contrário} \end{cases} \quad x, n \in \mathbb{N}$$

14.9.2 Notação 2

$$pow(x,n) = \begin{cases} 1, & \text{se } n = 0 \\ x * pow(x, n-1), & \text{caso contrário} \end{cases} \quad x, n \in \mathbb{N}$$

14.9.3 Exemplo em Linguagem C

```c
int pow( int x, int n ) {
    if ( n == 0 ) {
        return 1;
    } else {
        return x * pow( x, n - 1 );
    }
}
```

14.9.4 Usando *squaring*

$2^4 = 2^{(4/2)2} = (2^{4/2})^2 = (2^2)^2 = 4^2 = 16$

$2^5 = 2^{1+(4/2)2} = 2(2^{4/2})^2 = 2(2^2)^2 = 2(4^2) = 32$

$2^6 = 2^{(6/2)2} = (2^{6/2})^2 = (2^3)^2 = 8^2 = 64$

$2^7 = 2^{1+(6/2)2} = 2(2^{6/2})^2 = 2(2^3)^2 = 2(8^2) = 128$

$$pows(x,n) = \begin{cases} 1, & \text{se } n = 0 \\ pows(x, n/2)^2, & \text{se } n > 0 \text{ é par} \\ x * pows(x,(n-1)/2)^2, & \text{se } n > 0 \text{ é ímpar} \end{cases} \quad x, n \in \mathbb{N}$$

14.9.5 Exemplo em Linguagem C

```c
int pows( int x, int n ) {
    if ( n == 0 ) {
        return 1;
    } else if ( n % 2 == 0 ) {   // par
        int y = pows( x, n / 2 );
        return y * y;
    } else {                     // ímpar
        int y = pows( x, ( n - 1 ) / 2 );
        return x * y * y;
    }
}
```

14.10 Exercícios

Exercício 14.1: Escreva um programa que leia dois inteiros e que calcule o resultado da função `ackermann`. O protótipo dela é:

- `int ackermann( int m, int n )`

E ela é definida como:

$$ackermann(m,n) = \begin{cases} n+1, & \text{se } m = 0 \\ ackermann(m-1,1), & \text{se } m > 0 \text{ e } n = 0 \\ ackermann(m-1, ackermann(m,n-1)), & \text{se } m > 0 \text{ e } n > 0 \end{cases}$$

Saiba Mais

Quer saber mais sobre a função Ackermann? Siga o link **<http://dan-scientia.blogspot.com/2009/08/funcao-ackermann.html>**.

Arquivo com a solução: ex14.1.c

Entrada

```
Entre com o valor de m: 2
Entre com o valor de n: 1
```

Saída

```
ackermann( 2, 1 ) = 5
```

Entrada

```
Entre com o valor de m: 0
Entre com o valor de n: 1
```

Saída

```
ackermann( 0, 1 ) = 2
```

Entrada

```
Entre com o valor de m: 2
Entre com o valor de n: 3
```

Saída

```
ackermann( 2, 3 ) = 9
```

Exercício 14.2: Escreva um programa que leia dois inteiros e que calcule o máximo divisor comum entre dois números. O cálculo deve ser feito utilizando a função `mdc`. Seu protótipo é:

- `int mdc( int a, int b )`

E ela é definida como:

$$mdc(a,b) = \begin{cases} a, & \text{se } b = 0 \\ mdc(b, a\%b), & \text{se } a \geq b \text{ e } b > 0 \end{cases}$$

Arquivo com a solução: ex14.2.c

Entrada

```
Entre com o valor de a: 5
Entre com o valor de b: 20
```

Saída

```
mdc( 5, 20 ) = 5
```

Entrada

```
Entre com o valor de a: 15
Entre com o valor de b: 225
```

Saída

```
mdc( 15, 225 ) = 15
```

Entrada

```
Entre com o valor de a: 149
Entre com o valor de b: 7
```

Saída

```
mdc( 149, 7 ) = 1
```

Exercício 14.3: Escreva um programa que leia uma String e que a inverta, armazenando o resultado em outra String. Para realizar a inversão, deve-se usar a função `inverter`, que por sua vez, deve fazer a inversão de forma recursiva. Seu protótipo é:

- `void inverter( char *destino, const char *base, int tamanho )`

Arquivo com a solução: ex14.3.c

Entrada

```
String: abracadabra
```

Saída

```
Invertida: arbadacarba
```

CAPÍTULO 15

FUNÇÕES COM LISTA DE ARGUMENTOS VARIÁVEIS

"It is the user who should parameterize procedures, not their creators".

(PERLIS, 1982)

S funções em C podem também possuir uma quantidade arbitrária de parâmetros, recebendo assim uma quantidade arbitrária de argumentos. Você já usou algumas funções disponíveis no cabeçalho `stdio.h`, por exemplo, as funções `printf` e `scanf`, que têm essa natureza. Para isso, há a necessidade de se declarar tais funções e lidar com esses argumentos de uma forma um pouco diferente. Essas funções são chamadas de funções com lista argumentos variáveis ou simplesmente *Variadic functions*. Neste Capítulo são apresentados alguns exemplos de como realizar tal ação, não havendo exercícios.

15.1 Exemplos em Linguagem C

Exemplo de prototipação e implementação de funções com argumentos variáveis

```c
/*
 * Arquivo: FuncoesComArgumentosVariaveis.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>

/* stdarg.h é o cabeçalho necessário para a declaração
 * e definição de funções com argumentos variáveis.
 */

/* declaração da função max que possui um
 * parâmetro inteiro (n) e um parâmetro variável.
 *
 * o padrão da linguagem C exige que exista pelo menos
 * um parâmetro normal antes do parâmetro variável,
 * além do que deve haver apenas um o parâmetro variável
 * e ele deve estar obrigatoriamente no final da lista
 * de parâmetros.
 */
int max( int quantidade, ... );

/* declaração da função min que possui um
 * parâmetro inteiro (n) e um parâmetro variável.
 */
int min( int quantidade, ... );

int main() {

    int maior = max( 10, 4, 5, 9, 7, 8, 4, 5, 3, 2, 1 );
    int menor = min( 10, 4, 5, 9, 7, 8, 4, 5, 3, 2, 1 );

    printf( "Maior: %d\n", maior );
    printf( "Menor: %d", menor );

```

```
38      return 0;
39
40  }
41
42  int max( int quantidade, ... ) {
43
44      /* variável p do tipo va_list
45       * que vai ser usada para obter
46       * os argumentos do parâmetro variável.
47       */
48      va_list dados;
49
50      int atual;
51      int maior;
52
53      // inicia a captura de dados
54      va_start( dados, quantidade );
55
56      // obtém o primeiro item, um inteiro
57      maior = va_arg( dados, int );
58
59      for ( int i = 1; i < quantidade; i++ ) {
60
61          // obtém o próximo item, um inteiro
62          atual = va_arg( dados, int );
63
64          if ( atual > maior ) {
65              maior = atual;
66          }
67
68      }
69
70      // termina a captura de dados
71      va_end( dados );
72
73      return maior;
74
75  }
76
77  int min( int quantidade, ... ) {
78
79      va_list dados;
```

```
80      int atual;
81      int menor;
82
83      va_start( dados, quantidade );
84      menor = va_arg( dados, int );
85
86      for ( int i = 1; i < quantidade; i++ ) {
87
88          atual = va_arg( dados, int );
89
90          if ( atual < menor ) {
91              menor = atual;
92          }
93
94      }
95
96      va_end( dados );
97
98      return menor;
99
100 }
```

CAPÍTULO 16

USO AVANÇADO DE PONTEIROS

"One can only display complex information in the mind. Like seeing, movement or flow or alteration of view is more important than the static picture, no matter how lovely".

(PERLIS, 1982)

Á aprendemos a utilizar ponteiros anteriormente e estamos os utilizando desde então. Além do que já aprendemos, neste Capítulo veremos mais três tópicos que envolvem o uso de ponteiros. O primeiro será a capacidade de alocarmos quantidades arbitrárias de memória utilizando uma função específica para isso, ao invés de ter esse tipo de funcionalidade disponível apenas durante a declaração de variáveis. O segundo tema será a declaração e a utilização de ponteiros que são capazes de armazenar endereços de ponteiros! Imagine, se por algum motivo, você precise mudar o endereço que um ponteiro aponta de forma indireta. Os ponteiros para ponteiros são empregados com esse objetivo. Por fim, veremos como criar ponteiros que apontam para funções, permitindo, entre outras coisas, que possamos passar endereços de funções como argumento para outras funções!

16.1 Alocação Dinâmica de Memória

16.1.1 Exemplos em Linguagem C

Exemplo de código de alocação dinâmica de memória

```c
/*
 * Arquivo: UsoAvancadoPonteirosAlocacaoDinamica.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>

void imprimir( int *p, int n );
void configurar( int *p, int n, int valor );

int main() {

    int t = 10;

    /* void* malloc( size_t tamanhoTotal )
     *
     * A função malloc é utilizada para alocar dinamicamente
     * uma quantidade de memória em bytes.
     *
     * Retorna um ponteiro para o início da região alocada
     * caso haja espaço livre ou NULL caso não seja possível
     * alocar espaço. O ponteiro retornado é um ponteiro do
     * tipo void*, chamado de ponteiro genérico. Para a
     * atribuição desse retorno a um ponteiro, há a necessidade
     * de se realizar um cast (coerção explícita) para o tipo
     * de ponteiro necessário.
     */
    int *p1 = (int *) malloc( t * sizeof(int) );

    /* void* calloc( size_t quantidade, size_t tamanhoItem )
     *
     * A função calloc é utilizada para alocar dinamicamente
     * uma quantidade de memória em bytes e recebe dois
     * parâmetros: o primeiro é a quantidade de elementos
     * a serem alocados e o segundo é o tamanho de cada
```

```
37       * elemento
38       *
39       * Retorna um ponteiro para o início da região alocada
40       * caso haja espaço livre ou NULL caso não seja possível
41       * alocar espaço.
42       */
43      int *p2 = (int *) calloc( t, sizeof(int) );
44
45      if ( p1 != NULL && p2 != NULL ) {
46
47          imprimir( p1, t );
48          imprimir( p2, t );
49
50          configurar( p1, t, 1 );
51          configurar( p2, t, 2 );
52
53          imprimir( p1, t );
54          imprimir( p2, t );
55
56      }
57
58      /* liberando o espaço alocado
59       * SEMPRE LIBERE O ESPAÇO ALOCADO DINAMICAMENTE!!!
60       */
61      free( p1 );
62      free( p2 );
63
64      /* cuidado! a partir daqui, a região apontada por p1 e p2
65       * não é mais válida!!!
66       *
67       * cuidado! não invoque free mais de uma vez para o mesmo
68       * ponteiro!
69       *
70       * cuidado! se alguma operação de aritmética de ponteiros
71       * for feita, a invocação de free não fará o desejado.
72       */
73
74      return 0;
75
76  }
77
78  void imprimir( int *p, int n ) {
```

```
79
80      for ( int i = 0; i < n; i++ ) {
81          printf( "%d ", p[i] );
82      }
83      printf( "\n" );
84
85  }
86
87  void configurar( int *p, int n, int valor ) {
88
89      for ( int i = 0; i < n; i++ ) {
90          p[i] = valor;
91      }
92
93  }
```

16.2 Ponteiros para Ponteiros

16.2.1 Exemplos em Linguagem C

Exemplo de declaração e uso de ponteiros para ponteiros

```
1  /*
2   * Arquivo: UsoAvancadoPonteirosPonteirosParaPonteiros.c
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  void naoAltera( int *a, int *b );
10 void alterar( int **a, int *b );
11
12 int main() {
13
14     /* ponteiros para ponteiros são usados
15      * para alterar, indiretamente, o valor de
16      * um ponteiro.
17      */
18     int n1 = 10;
```

```
19      int n2 = 12;
20      int n3 = 14;
21
22      int *p1 = &n1;
23      int *p2 = &n2;
24      int *p3 = &n3;
25
26      int **pp = &p1;
27
28      printf( "p1: %x\n", p1 );
29      printf( "p2: %x\n", p2 );
30      printf( "p3: %x\n\n", p3 );
31
32      p1 = p2;
33      printf( "p1 = p2;\n" );
34      printf( "p1: %x\n", p1 );
35      printf( "p2: %x\n", p2 );
36      printf( "p3: %x\n\n", p3 );
37
38      naoAltera( p1, p3 );
39      printf( "naoAltera( p1, p3 );\n" );
40      printf( "p1: %x\n", p1 );
41      printf( "p2: %x\n", p2 );
42      printf( "p3: %x\n\n", p3 );
43
44      alterar( pp, p3 );
45      printf( "alterar( pp, p3 );\n" );
46      printf( "p1: %x\n", p1 );
47      printf( "p2: %x\n", p2 );
48      printf( "p3: %x", p3 );
49
50      return 0;
51
52  }
53
54  void naoAltera( int *a, int *b ) {
55      a = b;
56  }
57
58  void alterar( int **a, int *b ) {
59      *a = b;
60  }
```

16.3 Ponteiros para Funções

16.3.1 Exemplos em Linguagem C

Exemplo de declaração e uso de ponteiros para funções

```c
/*
 * Arquivo: UsoAvancadoPonteirosPonteirosParaFuncoes.c
 * Autor: Prof. Dr. David Buzatto
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void ordenar( int *valores, int n );

/* A função ordenarUsandoFuncao tem como terceiro parâmetro
 * um ponteiro para uma função, chamado pFuncao (indicado
 * por (*pFuncao), sendo que esse ponteiro é capaz de apontar
 * para funções que retornam inteiros e que têm dois parâmetros
 * inteiros.
 */
void ordenarUsandoFuncao( int *valores, int n,
                          int (*pFuncao)( int n1, int n2 ) );
void embaralhar( int *valores, int n );

void imprimir( int *p, int n );
int sortear( int inicio, int fim );

int crescente( int n1, int n2 );
int decrescente( int n1, int n2 );
void imprimirMensagem();

int main() {

    int v[10] = { 3, 5, 1, 4, 3, 9, 4, 2, -1, -3 };
    int t = 10;

    /* declaração e inicialização de um ponteiro para
     * uma função que não retorna nada e não possui
     * parâmetros.
```

```
37      */
38     void (*pFV)() = imprimirMensagem;
39
40     /* declaração de um ponteiro para uma função que
41      * retorna um inteiro e recebe dois inteiros
42      * como parâmetro
43      */
44     int (*pFDec)( int, int );
45     pFDec = decrescente;
46
47
48     srand( time( NULL ) );
49
50
51     imprimirMensagem();
52     printf( "(invocacao de imprimirMensagem)\n" );
53     pFV();
54     printf( "(invocacao indireta via pFV)\n\n" );
55
56     printf( "Embaralhando...\n" );
57     embaralhar( v, t );
58     imprimir( v, t );
59
60     printf( "Ordenado:\n" );
61     ordenar( v, t );
62     imprimir( v, t );
63
64
65     printf( "\n\nEmbaralhando...\n" );
66     embaralhar( v, t );
67     imprimir( v, t );
68
69     printf( "Ordenado crescente usando 'int crescente(int, int)':\n" );
70     ordenarUsandoFuncao( v, t, crescente );
71     imprimir( v, t );
72
73
74     printf( "\n\nEmbaralhando...\n" );
75     embaralhar( v, t );
76     imprimir( v, t );
77
78     printf( "Ordenado decrescente usando pFDec:\n" );
```

```
79      ordenarUsandoFuncao( v, t, pFDec );
80      imprimir( v, t );
81
82      return 0;
83
84  }
85
86  void ordenar( int *valores, int n ) {
87
88      int t;
89
90      for ( int i = 0; i < n; i++ ) {
91          for ( int j = 0; j < n-1; j++ ) {
92              if ( valores[j] > valores[j+1] ) {
93                  t = valores[j];
94                  valores[j] = valores[j+1];
95                  valores[j+1] = t;
96              }
97          }
98      }
99
100 }
101
102 void ordenarUsandoFuncao( int *valores, int n,
103                           int (*pFuncao)( int n1, int n2 ) ) {
104
105     int t;
106     int c;
107
108     for ( int i = 0; i < n; i++ ) {
109         for ( int j = 0; j < n-1; j++ ) {
110
111             // invocando a função passada como parâmetro
112             c = pFuncao( valores[j], valores[j+1] );
113
114             if ( c > 0 ) {
115                 t = valores[j];
116                 valores[j] = valores[j+1];
117                 valores[j+1] = t;
118             }
119
120         }
```

```
121     }
122
123 }
124
125 void embaralhar( int *valores, int n ) {
126
127     int p;
128     int t;
129
130     for ( int i = 0; i < n; i++ ) {
131         p = sortear( 0, n-1 );
132         t = valores[i];
133         valores[i] = valores[p];
134         valores[p] = t;
135     }
136
137 }
138
139 void imprimir( int *p, int n ) {
140
141     for ( int i = 0; i < n; i++ ) {
142         printf( "%d ", p[i] );
143     }
144     printf( "\n" );
145
146 }
147
148 int sortear( int inicio, int fim ) {
149     return inicio + ( rand() % ( fim - inicio + 1 ) );
150 }
151
152 int crescente( int n1, int n2 ) {
153     return n1 - n2;
154 }
155
156 int decrescente( int n1, int n2 ) {
157     return n2 - n1;
158 }
159
160 void imprimirMensagem() {
161     printf( "Comecando..." );
162 }
```

CAPÍTULO 17

TRATAMENTO DE ERROS

"There are two ways to write error-free programs; only the third one works".

(PERLIS, 1982)

maioria das linguagens de programação, senão todas, disponibilizam ao programador funcionalidades específicas para a recuperação de erros durante a execução do programa. Por exemplo, calcular a raiz quadrada de um número negativo deve gerar um erro, mas como um programa pode se recuperar de tal ação não permitida? Neste Capítulo veremos como realizar tais tipos de tratamento na linguagem de programação C.

17.1 Exemplos em Linguagem C

Tratamento de erros na linguagem de programação C

```
1 /*
2  * Arquivo: TratamentoDeErros.c
3  * Autor: Prof. Dr. David Buzatto
4  */
```

```c

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <errno.h>

/* no cabeçalho errno.h é declarada a variável errno
 * que conterá algum código de erro depois da execução
 * de alguma função que a usa para sinalizar um
 * possível erro.
 *
 * a maioria das funções que modificam errno são as do
 * cabeçalho math.h.
 */

void exemploBasico();
void exemploDominio();
void exemploIntervalo();
void exemploMensagem();

int main() {

    exemploBasico();
    exemploDominio();
    exemploIntervalo();
    exemploMensagem();

    return 0;

}

void exemploBasico() {

    float v;

    // zerando a variável errno
    errno = 0;

    // raiz quadrada de um número negativo
    v = sqrt( -9 );

```

```
47      if ( errno != 0 ) {
48          fprintf( stderr, "erro na funcao sqrt, valor negativo\n" );
49      }
50
51  }
52
53  void exemploDominio() {
54
55      float v;
56
57      // zerando a variável errno
58      errno = 0;
59
60      /* um valor negativo está fora do domínio
61       * dos valores permitidos para a função sqrt
62       */
63      v = sqrt( -9 );
64
65      if ( errno == EDOM ) {
66          fprintf( stderr, "erro na funcao sqrt, fora do dominio\n" );
67      }
68
69  }
70
71  void exemploIntervalo() {
72
73      double v;
74
75      // zerando a variável errno
76      errno = 0;
77
78      /* 1000 é um valor muito grande, ou seja
79       * fora do intervalo, permitido para a execução
80       * da função exp, que deveria calcular para 1000
81       * o valor de e^1000
82       */
83      v = exp( 1000 );
84
85      if ( errno == ERANGE ) {
86          fprintf( stderr, "erro na funcao exp, " );
87          fprintf( stderr, "fora do intervalo permitido\n" );
88      }
```

```
89
90  }
91
92  void exemploMensagem() {
93
94      double v;
95      char mensagem[80];
96
97      // zerando a variável errno
98      errno = 0;
99
100     /* qual é o logarítimo base 10 de 0? ou seja
101      * qual número deve elevar 10 para resultar em 0?
102      */
103     v = log10( 0 );
104
105     if ( errno != 0 ) {
106
107         /* a função perror, declarada em stdio.h
108          * exibe uma mensagem de erro padronizada
109          * para o erro que foi gerado.
110          */
111         perror( "Erro em log10" );
112
113         /* a função strerror, declarada em string.h
114          * recebe um inteiro como parâmetro e gera
115          * a mensagem de erro baseada nesse valor.
116          * é usada como base dentro da função perror
117          * apresentada acima.
118          */
119         strcpy( mensagem, strerror( errno ) );
120         fprintf( stderr, "Mensagem: %s\n", mensagem );
121
122         // ou...
123         fprintf( stderr, "Ou: %s\n", strerror( errno ) );
124
125     }
126
127
128 }
```

CAPÍTULO 18

CLASSES DE ARMAZENAMENTO, QUALIFICADORES E INICIALIZAÇÃO

"Making something variable is easy. Controlling duration of constancy is the trick".

(PERLIS, 1982)

NESTE Capítulo veremos os últimos detalhes pertinentes à linguagem de programação C que nos interessa nesse momento. Serão apresentadas as palavras chave que podem ser empregadas para se mudar o comportamento padrão de como variáveis são armazenadas na memória etc.

Na linguagem C os especificadores de declaração são:

- Classes de armazenamento:
  - `auto`: Variáveis declaradas em blocos tem armazenamento automático, ou seja, são gerenciadas de modo que ao terminar a execução de um bloco, a espaço alocado para a variável é liberado, fazendo com que as mesmas tenham escopo de bloco e sem linkagem, ou seja, não é compartilhada com nenhuma outra parte do programa;
  - `static`: Variáveis de armazenamento estático não perdem seu valor ao fim da execução um bloco. Quando declaradas fora de blocos, a linkagem

da variável se torna interna, ou seja, só é enxergada dentro do arquivo em que reside. Quando declaradas dentro de blocos, não tem linkagem, mas a propriedade de manter o valor após o fim de bloco se mantém. Pode ser usada em funções, fazendo com que a função só possa ser invocada dentro do arquivo em que for declarada;
  - `extern`: A classe de armazenamento externa permite que vários arquivos de código fonte tenham acesso à variável declarada além de terem a característica de variáveis de armazenamento estático. Pode ser usada em funções, sendo o comportamento padrão, que permite que a função seja usada em outros arquivos;
  - `register`: Esse tipo de armazenamento sugere ao compilador que a variável seja armazenada em algum registrador da CPU ao invés de ser armazenada na memória principal. É ideal para ser utilizado quando a variável é acessada tanto para leitura, quanto para modificação, constantemente. Por exemplo, a variável de controle de uma estrutura de repetição *for*.
- Qualificadores de tipo:
  - `const`: Faz com que uma variável possa ser apenas lida após sua inicialização, ou seja, seu valor não pode ser alterado. Um ponteiro `const` indica que não é permitido alterar o valor da variável apontada pelo ponteiro;
  - `volatile`: Utilizado em programação de baixo nível. Usada para indicar ao compilador, normalmente ao se usar ponteiros, que o ponteiro aponta para uma região de memória volátil, ou seja, que é alterada constantemente durante a execução do programa sem que necessariamente seja alterada pela influência direta do programa, por exemplo, um fluxo de entrada de algum dispositivo;
  - `restrict`: Utilizado apenas em ponteiros. Indica que o ponteiro aponta para uma região não compartilhada de memória.
- Especificadores de tipo (já aprendemos):
  - `void`;
  - `char`, `int`, `float` e `double`;
  - `short` e `long`;
  - `signed` e `unsigned`;
  - Espeficadores de tipos abstratos de dados:
    * `struct`: Especificação de estruturas;
    * `union`: Especificação de uniões;
    * `enum`: Especificação de enumerações;
    * Nomes de tipos criados usando `typedef` são também especificadores de tipo.
- Especificador de função:

– `inline`: Sugere ao compilador que a invocação da função seja substituída/expandida pelo código de sua definição.

CAPÍTULO 19

CONCLUSÃO

"O que vale na vida não é o ponto de partida e sim a caminhada. Caminhando e semeando, no fim terás o que colher".

Cora Coralina

ASSIM finalizamos nosso livro! Vale a pena salientar que o que vimos durante o mesmo corresponde a uma pequena parte das funcionalidades disponíveis na linguagem de programação C, sendo que, para que você possa esgotar o assunto, será necessário o estudo de outras fontes, além de anos de prática com a linguagem em um ambiente de desenvolvimento real.

Nos cursos de computação utilizamos a linguagem C como linguagem de programação inicial para o aprendizado dos conceitos básicos de programação, mas dificilmente você trabalhará com ela no mundo real. Isso, no entanto, não é impossível, pois há vagas para esse tipo de programador.

Caso queira se especializar mais na linguagem, recomendo as obras:

- DEITEL, P. M.; DEITEL, H. M. **C: como programar**. 6. ed. São Paulo: Pearson, 2011. 846 p.
- KING, K. N. **C Programming: a modern approach**. 2. ed. New York: W. W. Norton & Company, 2008. 805 p.

O trabalho de King (2008) talvez seja a melhor e mais completa obra da linguagem C disponível no mercado.

Espero que este livro tenha sido útil!

Um grande abraço a todos!

Até mais!

BIBLIOGRAFIA

BEECROWD. 2023. Disponível em: <https://www.beecrowd.com.br/>. Acesso em: 06 de janeiro de 2023.

DEITEL, P. M.; DEITEL, H. M. **C: como programar**. 6. ed. São Paulo: Pearson, 2011. 846 p.

DEITEL, P. M.; DEITEL, H. M. **Java: como programar**. 10. ed. São Paulo: Pearson, 2016. 968 p.

KING, K. N. **C Programming: a modern approach**. 2. ed. New York: W. W. Norton & Company, 2008. 805 p.

PERLIS, A. J. Special feature: Epigrams on programming. **SIGPLAN Not.**, ACM, New York, NY, USA, v. 17, n. 9, p. 7–13, set. 1982. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/947955.1083808>.

ISBN: 978-65-00-60543-3