Estudo Dirigido de Linguagem C

José Augusto N. G. Manzano

Estudo Dirigido de Linguagem C

17ª Edição Revisada e Atualizada

www.editoraerica.com.br

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

Manzano, José Augusto Navarro Garcia, 1965

Estudo Dirigido de Linguagem C / José Augusto N. G. Manzano.-- 17. ed. rev. -- São Paulo : Érica, 2013.

Bibliografia.
ISBN 978-85-365-1151-1

1. C (Linguagem de Programação) I. Título.

13-10535 CDD-005.133

Índice para catálogo sistemático

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

Copyright © 2002 da Editora Érica Ltda.

Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida por qualquer meio ou forma sem prévia autorização da Editora Érica. A violação dos direitos autorais é crime estabelecido na Lei nº 9.610/98 e punido pelo Artigo 184 do Código Penal.

Coordenação Editorial:	Rosana Arruda da Silva
Capa:	Edson Antonio dos Santos
Editoração e Finalização:	Tatiana Cardoso Gonçalves
	Dalete Regina de Oliveira
	Marlene T. Santin Alves
	Carla de Oliveira Morais
	Rosana Ap. A. dos Santos

O Autor e a Editora acreditam que todas as informações aqui apresentadas estão corretas e podem ser utilizadas para qualquer fim legal. Entretanto, não existe qualquer garantia, explícita ou implícita, de que o uso de tais informações conduzirá sempre ao resultado desejado. Os nomes de sites e empresas, porventura mencionados, foram utilizados apenas para ilustrar os exemplos, não tendo vínculo nenhum com o livro, não garantindo a sua existência nem divulgação. Eventuais erratas estarão disponíveis para download no site da Editora Érica.

Conteúdo adaptado ao Novo Acordo Ortográfico da Língua Portuguesa, em execução desde 1º de janeiro de 2009.

A Ilustração de capa e algumas imagens de miolo foram retiradas de <www.shutterstock.com>, empresa com a qual se mantém contrato ativo na data de publicação do livro. Outras foram obtidas da Coleção MasterClips/MasterPhotos© da IMSI, 100 Rowland Way, 3rd floor Novato, CA 94945, USA, e do CorelDRAW X5 e X6, Corel Gallery e Corel Corporation Samples. Copyright© 2013 Editora Érica, Corel Corporation e seus licenciadores. Todos os direitos reservados.

Todos os esforços foram feitos para creditar devidamente os detentores dos direitos das imagens utilizadas neste livro. Eventuais omissões de crédito e copyright não são intencionais e serão devidamente solucionadas nas próximas edições, bastando que seus proprietários contatem os editores.

Seu cadastro é muito importante para nós

Ao preencher e remeter a ficha de cadastro constante no site da Editora Érica, você passará a receber informações sobre nossos lançamentos em sua área de preferência.

Conhecendo melhor os leitores e suas preferências, vamos produzir títulos que atendam suas necessidades.

Contato com o editorial: editorial@editoraerica.com.br

Editora Érica Ltda. | Uma Empresa do Grupo Saraiva
Rua São Gil, 159 - Tatuapé
CEP: 03401-030 - São Paulo - SP
Fone: (11) 2295-3066 - Fax: (11) 2097-4060
www.editoraerica.com.br

Fabricante

Software: GCC - GNU Compiler Collection

Copyright (C) Free Software Foundation, Inc.

Contato pelo e-mail: florian@freepascal.org (em inglês)

Sítio: http://gcc.gnu.org/

Requisitos Básicos de Hardware e de Software

Hardware

- Microcomputador IBM-PC com processador mínimo, padrão Intel Pentium ou superior (incluindo-se a família AMD);
- 1 Gbyte de memória RAM (mínimo);
- 210 Mbytes de disco rígido para a instalação completa do compilador.

Software

MS-Windows/MS-DOS

- Sistema Operacional MS-Windows 2000/XP ou superior em modo de operação de 32 *bits*.

Linux

- Sistema Operacional Linux (Fedora 12).

Dedicatória

À minha esposa Sandra, Sol do meu dia, Lua da minha noite, vento que me abraça em seu amor, brisa que refresca minha alma, chuva que acalenta o meu ser. Te amo, te amo, te amo.

À minha filha Audrey, encanto de pessoa, flor que enfeita meu caminho e o de minha esposa.

Aos meus amigos, alunos e leitores cujas dúvidas, perguntas e comentários me incentivam e fazem com que eu procure me aprimorar cada vez mais, com um grande abraço a todos.

Agradecimentos

Aos amigos e funcionários da Editora Érica, que, desde o primeiro contato, confiaram em mim estendendo essa atitude até os dias de hoje, dando-me a oportunidade de continuar divulgando meus trabalhos. A toda a equipe da Editora que, com atenção e apreço, concretiza cada obra que publico.

A todos que direta ou indiretamente colaboram com meu trabalho, principalmente aos amigos professor Carlos Takeo Akamine, professor Roberto Yamamoto, Carlos Gomes da Fitcor, engenheiro Ricardo L. M. Martinez da Mercedes-Benz do Brasil e professor Goya pelas sugestões para conclusão e continuidade do aprimoramento deste livro a cada nova edição.

"Pedi e será dado; buscai e achareis; batei e será aberto."

Mateus - 7,7

Sumário

Prefácio

Este não se trata de mais um livro de linguagem C que surge no mercado. Ele tem dois objetivos básicos, sendo um didático para que o aluno (leitor) aprenda de forma tranquila a utilizar os conceitos iniciais de programação existentes na linguagem C; o outro é que sirva de apoio às aulas de linguagens e técnicas de programação, cumprindo os objetivos gerais da Coleção P. D. Estudo Dirigido.

Este livro é indicado para alunos iniciantes no estudo da linguagem. Assim sendo, não é abordado neste material assuntos considerados avançados. Mostra-se apenas recursos introdutórios para conhecimento inicial da linguagem.

O livro apoia um roteiro de estudos baseado em uma estrutura padronizada de trabalho já consagrada em outros livros de linguagens de programação, pertencentes à Coleção P. D. Estudo Dirigido, como é o caso dos títulos Estudo Dirigido de Algoritmos (base central para o estudo de lógica de programação), Estudo Dirigido de C#, Estudo Dirigido de Delphi e Estudo Dirigido de Visual Basic.

O programa de estudo desta obra está dividido em oito capítulos, sendo o primeiro destinado a uma rápida introdução para situar o aluno (leitor) no contexto do surgimento da linguagem C e também fornecer algumas regras adotadas na obra.

O segundo capítulo apresenta o ambiente de trabalho do compilador GCC, (linha de comando e Code::Blocks). No entanto o professor pode trabalhar com outro ambiente de programação, pois o código escrito está em formato ISO/IEC 9899:2011, desde que o ambiente, escolhido pelo professor, suporte esta norma. A sugestão de uso do compilador GCC é por ser um produto conhecido e estar disponível para *download* gratuitamente a partir do sítio do Projeto GNU para diversos sistemas operacionais.

O terceiro capítulo aborda os tipos de dados, variáveis, constantes, operadores aritméticos, expressões aritméticas, utilização prática de entrada e saída. Este capítulo permeia o uso das técnicas de programação sequencial.

Tomada de decisões é assunto do quarto capítulo. É realizado um estudo sobre decisões simples, decisões compostas, operadores relacionais e lógicos, além de abordar o uso do arquivo de cabeçalho ISO 646 como suporte

ao uso de operadores lógicos e apresentar as regras para operação com divisibilidade de valores numéricos.

O quinto capítulo mostra o uso da técnica de programa com laços para a repetição de trechos de programas, também conhecidos como malhas de repetição ou *loopings*.

A utilização de tabelas (matrizes) e listas (vetores) em linguagem C é tema do sexto capítulo. Apresenta os conceitos de trabalho com uma (vetores) e duas dimensões (matrizes) para dados numéricos e do tipo caractere (*strings*).

No sétimo capítulo são apresentados os trabalhos com funções internas e externas definidas pelo próprio programador, bem como a criação da própria biblioteca particular de funções. Esse capítulo mostra como trabalhar com alguns códigos de controle para o modo Terminal ANSI e Windows API para a realização de ações de limpeza (tela/linha) e posicionamento de cursor no vídeo.

O oitavo e último capítulo faz uma introdução à utilização de arquivos em disco. São fornecidos exemplos de utilização de arquivos em formato texto e binário. Ao final do capítulo é apresentado um exemplo com o máximo de recursos utilizados em todos os exercícios de aprendizagem abordados.

Os livros técnicos de programação, voltados para a sala de aula, periodicamente necessitam ser reformulados no sentido de estarem sempre de acordo com as necessidades que surgem ou com as mudanças ocorridas no mercado ou nas ferramentas usadas. Nesta décima sétima edição, foram feitos diversos ajustes ao longo da obra em praticamente todos os capítulos, destacando-se as mudanças do Capítulo 2; que teve seu texto reescrito; do Capítulo 3, em que foram acrescidas melhorias nas informações relacionadas ao uso dos tipos de dados da linguagem C; e do Capítulo 7, em que se ampliou o uso de bibliotecas definidas pelo programador com o emprego do modo Windows API, para uso dos sistemas operacionais da Microsoft de 32 e 64 *bits*.

Após a conclusão do estudo deste livro, espera-se que o aluno, leitor deste trabalho, tenha adquirido base sólida para prosseguir o aprendizado dessa maravilhosa e uma das mais usadas linguagens do mercado computacional. Para o aprimoramento desse conhecimento, recomenda-se continuar o estudo com obras avançadas sobre o tema.

Todos os programas exemplificados neste texto foram escritos respeitando-se o padrão recomendado pela norma ISO/IEC 9899:2011[1], portanto, acredita-se que podem ser executados em outros compiladores de linguagem C sem problemas, desde que esses compiladores sejam compatíveis com a norma internacional da linguagem. Apesar do cuidado em manter o código escrito dos programas em formato-padrão, não é possível garantir compatibilidade com todos os compiladores existentes, pois alguns podem não identificar corretamente determinadas funções. Quando isso ocorre, é necessário substituir por outra função que possua uma ação compatível com a utilizada no momento.

A todos um grande abraço e um bom aprendizado.

O Autor

[1] *O livro* Estudo Dirigido de Linguagem C *da primeira até a quinta edição dava ênfase à programação em linguagem C, padrão Turbo 2.01. Da sexta até a décima segunda edição, foi adotado o critério de escrever os programas de forma mais genérica, permitindo usar maior número de compiladores. Da décima terceira até a décima sexta edição, foi adotada a forma de escrita recomendada na norma ISO/IEC 9899:1999. A décima sétima edição usa a norma ISO/IEC 9899:2011 como referência para os códigos em uso.*

Objetivo da Coleção

Esta obra faz parte da *Série Estudo Dirigido (Coleção P.D.)* e tem por objetivo principal servir de material de apoio ao ensino das disciplinas relacionadas às áreas de computação, desenvolvimento e tecnologia da informação. Deseja-se fornecer um conteúdo programático de aula que seja útil ao professor e também ao aluno.

Desta forma, os exercícios de fixação apresentados ao final de alguns capítulos devem ser trabalhados com a supervisão de um professor, preferencialmente responsável pela matéria, pois ele terá condições técnicas necessárias para proceder às devidas correções e apresentação de seu gabarito, além de poder criar a sua própria bateria de exercícios.

Procurou-se fazer um trabalho que não fosse muito extenso e pudesse ser usado em sua totalidade, principalmente dentro do período letivo, sem com isso tomar um tempo demasiado do leitor/aluno. Outro ponto de destaque da obra é a linguagem, pois foi o objetivo, desde o princípio, manter um estilo agradável e de fácil leitura, dosando a utilização de termos técnicos e de conceitos, normalmente considerados de grande complexidade.

Coordenador da Coleção:

Prof. Mestre José Augusto N. G. Manzano

Sobre o Autor

José Augusto N. G. Manzano

Brasileiro, nascido no Estado de São Paulo, capital, em 26 de abril de 1965, é professor e mestre com licenciatura em matemática. Atua na área de Tecnologia da Informação (ensino, desenvolvimento de software e treinamento) desde 1986. Participou do desenvolvimento de aplicações computacionais para áreas de telecomunicações e comércio. Na carreira docente iniciou suas atividades em cursos livres, passando por empresas de treinamento e atuando nos ensinos técnico e superior. Atuou em empresas na área de TI como: ABAK Informática Ltda. (funcionário), SERVIMEC S.A. (funcionário), CEBEL, SPCI, BEPE, ORIGIN, OpenClass, entre outras, como consultor e instrutor freelancer.

Atualmente é professor efetivo do IFSP (Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, antiga Escola Técnica Federal). Em sua carreira docente, possui condições técnicas de ministrar componentes curriculares de Algoritmos, Lógica de Programação, Estrutura de Dados, Técnicas de Programação, Microinformática, Informática Básica, Linguagens de Programação Estruturada, Linguagens de Programação Orientada a Objetos, Engenharia de Software, Tópicos Avançados em Processamento de Dados, Sistemas de Informação, Engenharia da Informação, Arquitetura de Computadores e Tecnologias Web. Possui conhecimento de uso e aplicação das linguagens de programação BASIC CLASSIC, COMAL, Assembly, LOGO, PASCAL, FORTRAN, C, C++, C++/CLI, JAVA, MODULA-2, STRUCTURED BASIC, C#, Lua, HTML, XHTML, JavaScript, VBA e ADA. Possui mais de sessenta obras publicadas, além de artigos técnicos publicados no Brasil e no exterior.

Sobre o Material Disponível na Internet

O material disponível na Internet contém os programas-fonte utilizados no livro. Para abrir o arquivo, é necessário que o usuário possua instalado em sua máquina um compilador de linguagem C compatível com a estrutura da obra. Neste caso o compilador GCC.

fontes.exe - 96,6KB

Procedimento para Download

Acesse o site da Editora Érica Ltda.: www.editoraerica.com.br. A transferência do arquivo disponível pode ser feita de duas formas:

- **Por meio do módulo pesquisa**. Localize o livro desejado, digitando palavras-chave (nome do livro ou do autor). Aparecem os dados do livro e o arquivo para download. Com um clique o arquivo executável é transferido.
- **Por meio do botão "Download"**. Na página principal do site, clique no item "Download". É exibido um campo no qual devem ser digitadas palavras--chave (nome do livro ou do autor). Aparecem o nome do livro e o arquivo para download. Com um clique o arquivo executável é transferido.

Procedimento para Descompactação

Primeiro passo: após ter transferido o arquivo, verifique o diretório em que se encontra e dê um duplo clique nele. Aparece uma tela do programa WINZIP SELF-EXTRACTOR que conduz ao processo de descompactação. Abaixo do Unzip To Folder há um campo que indica o destino do arquivo que será copiado para o disco rígido do seu computador.

C:\ED_LINGUAGEM_C

Segundo passo: prossiga a instalação, clicando no botão Unzip, o qual se encarrega de descompactar o arquivo. Logo abaixo dessa tela aparece a barra de status que monitora o processo para que você acompanhe. Após o término, outra tela de informação surge, indicando que o arquivo foi descompactado com sucesso e está no diretório criado. Para sair dessa tela, clique no botão OK. Para finalizar o programa WINZIP SELF-EXTRACTOR, clique no botão Close.

1

capítulo

Antes de Começar

Objetivos

Este capítulo apresenta uma introdução ao histórico da linguagem C e da notação que será utilizada nesta obra para o aprendizado dos recursos da linguagem.

1.1 Introdução

A evolução do processamento de dados de forma automática tem como ponto de partida o século XIX. Neste período ocorreram diversas tentativas para construir aparelhos mecânicos capazes de efetuar operações matemáticas. Atualmente, esses aparelhos não são mais mecânicos, mas eletrônicos. Os computadores ganharam grande espaço, os quais não são mais usados apenas para realizar operações matemáticas, como nos primórdios de sua invenção.

Atualmente é grande o número de pessoas que se interessam pelos recursos computacionais, não só como usuários, mas também como profissionais da área de desenvolvimento de *software* do setor de Tecnologia da Informação, os quais buscam soluções computacionais como suporte para problemas específicos de cada área profissional existente.

Este trabalho tem por finalidade apresentar os conceitos preliminares da programação de computadores com a linguagem de programação C. Leva-se em consideração que o leitor não possui conhecimento da sintaxe dessa linguagem, mas já tem familiaridade com algoritmos e lógica de programação em nível básico (espera-se que o aluno saiba utilizar programação sequencial, tomada de decisões, laços, matrizes e sub-rotinas). Caso não tenha esse conhecimento, sugere-se consultar a obra:

Estudo Dirigido de Algoritmos
Autores: José Augusto Manzano e Jayr Figueiredo
Editora Érica

1.2 Breve Histórico da Linguagem C

Em 1972, no laboratório da empresa Bell Telephone Labs. Inc. (atual Alcatel--Lucent), foi projetada a linguagem de programação de computadores C por Dennis M. Ritchie que a desenvolveu com o objetivo de utilizá-la na codificação da segunda versão do sistema operacional UNIX para a equipe de trabalho chefiada por Ken Thompson. Tempo depois, Brian W. Kernighan juntou-se ao projeto de ampliação da linguagem C..

A primeira versão do sistema operacional UNIX foi escrita em linguagem *Assembly* para o computador DEC PDP-11, precursor dos atuais *minicomputadores*. A partir da segunda versão até hoje o UNIX é escrito com a linguagem C.

A linguagem de programação C é derivada da ALGOL 68 escrita pelo Prof. Niklaus Wirth (criador da linguagem Pascal) e baseada na linguagem B de Ken Thompson que era uma evolução da linguagem BCPL. Alguns autores acharam que possivelmente a próxima linguagem baseada na estrutura C seria chamada de P, mas basicamente isso não ocorreu, pois a linguagem seguinte à C foi chamada de C++ (Linguagem C com suporte à Programação Orientada a Objetos), desenvolvida por Bjarne Stroustrup.

A linguagem C nasceu da necessidade de escrever programas que utilizem os recursos internos de máquina de uma forma mais fácil que a linguagem de montagem Assembly. C permite a integração direta entre alto nível e baixo nível, permitindo a escrita de código *Assembly* dentro do código em alto nível.

A grande aceitação da linguagem decorre da elegância em conciliar seu poder de programação em baixo nível com o seu alto grau de portabilidade, ou seja, um programa escrito em C pode teoricamente ser rodado em qualquer plataforma. C é uma linguagem de propósito geral, estruturada do tipo imperativa com suporte a programação estruturada, por possibilitar o trabalho de programação com base no paradigma procedural.

Desde seu lançamento a linguagem C passou por algumas mudanças. Entre os anos de 1983 e 1989, o instituto de normas técnicas ANSI iniciou um trabalho de padronização, de forma que a linguagem fosse realmente portável entre diversas plataformas, pois a linguagem C passou a ser usada em diversos tipos de computadores. O padrão definido a partir do trabalho do instituto ANSI é conhecido como ANSI C (norma ANSI X3.159-1989). Após o ano de 1990, o padrão ANSI C passou por pequenas mudanças e foi assim adotado pela Organização Internacional de Padrões, conhecida como ISO na norma ISO/IEC 9899:1990, também conhecido como C89 ou C90.

No final da década de 1990, a linguagem C passa por uma nova revisão, levando à publicação da norma ISO/IEC 9899:1999, também referenciada como C99, tema desta obra.

No ano de 2007, uma nova revisão foi iniciada, conhecida como C1X, no entanto até o momento esse trabalho não foi concluído.

1.3 Notação Utilizada

Procurou-se manter ao máximo o padrão de notação adotado para a linguagem C baseada na norma ISO/IEC 9899:2012 de maneira que os programas desta obra possam ser executados em diversos sistemas operacionais, porém para facilitar a visualização dos códigos de programa apresentados nos exemplos de aprendizagem, foi necessária a definição de alguns parâmetros gráficos:

- Todas as variáveis utilizadas em um programa nesta obra são escritas em caracteres maiúsculos e em itálico; as instruções (comandos e funções) sempre em caracteres minúsculos e em negrito.
- No momento em que o leitor for escrever os códigos de programa exemplificados no programa compilador, deve escrever as variáveis apenas com caracteres maiúsculos (o formato itálico é impossível de ser conseguido) e as instruções devem ser escritas apenas em caracteres minúsculos (o formato negrito também não pode ser formatado).
- As chamadas de função são indicadas em caracteres minúsculos.
- Na indicação das sintaxes, o leitor encontra alguns termos escritos entre os símbolos "<" (menor que) e ">" (maior que), que são informações obrigatórias que devem ser substituídas por algum elemento da linguagem C.
- O que for mencionado entre os símbolos "[" (colchete esquerdo) e "]" (colchete direito) é uma informação opcional, a qual pode ser omitida para a utilização do recurso indicado. Exceção no que se refere a tabelas em memória.
- Toda menção feita a uma condição utilizada é definida entre parênteses "(" e ")".
- Nesta obra os programas são escritos com espaço dois de endentação. Normalmente programas escritos em linguagem C possuem um espaço maior. Esta restrição se faz necessária para que os programas caibam adequadamente no tamanho horizontal da página.

Há ainda um detalhe adicional a ser considerado. Todos os programas utilizados na obra são desenvolvidos para a execução em *prompt* de comando (modo console).

2

capítulo

O Ambiente de Trabalho

Objetivos

Para estudar a linguagem de programação C abordada no livro, é importante possuir instalado em seu microcomputador um compilador da linguagem de programação C, preferencialmente o GCC. Este capítulo orienta o uso nos sistemas operacionais Linux e Microsoft Windows e sua aquisição, principalmente para o sistema operacional da Microsoft.

2.1 GNU Compiler Collection

A coleção de compiladores GNU, conhecidos como **GCC** (GNU Compiler Collection) é uma ferramenta voltada ao desenvolvimento de *software*, sendo distribuída pelo projeto GNU (**http://www.gnu.org**) em regime de *software livre*, para vários sistemas operacionais como MS-DOS, Microsoft Windows, Linux, entre outros. Trata-se de um ambiente de programação que contém *front-ends* para diversas linguagens para computadores, como C, C++, Objective-C, FORTRAN, Java, Assembly, Cobol e Ada entre outras. A sigla **GNU** significa **G**NU **N**ot **U**nix (**G**NU **N**ão é **U**nix), sendo esta recursiva, segundo Richard Matthew Stallman, criador do projeto e também do compilador **GCC**, sendo o arauto do movimento *software livre* no mundo. Outro detalhe em relação ao compilador **GCC** é sua sintaxe normatizada internacionalmente, tornando seu uso idêntico aos compiladores usados no sistema operacional UNIX.

Apesar de algumas poucas críticas a esse compilador, é um dos poucos produtos a respeitar plenamente os padrões normatizados pelo órgão ISO, principalmente para as linguagens C e C++. Por essa razão, é o compilador de linguagem C/C++ mais utilizado e respeitado.

Só para dar uma noção de seu poder, basta citar que é nesse compilador que Linus Torvalds e equipe desenvolvem o sistema operacional Linux.

A quase totalidade dos programas-exemplo desta obra pode ser executada sem nenhum tipo de problema, tanto no sistema operacional Linux como no Windows, exceto alguns códigos particulares ao sistema operacional Linux não puderem ser executados por algum motivo no Windows. Nesse caso, será apresentada uma versão compatível para o Windows.

O GCC é normalmente utilizado em modo de linha de comando (console texto), tanto no sistema operacional Microsoft Windows como no Linux. Os programas devem ser escritos em um editor de textos. No sistema operacional Microsoft Windows pode-se utilizar o programa **edit** em modo texto, o **Bloco de notas** em modo gráfico, ou usar o programa livre **Notepad++**. No sistema operacional Linux há grande variedade de programas de edição de textos, entre eles destacam-se **emacs**, **vi**, **joe**, **gedit** etc.

Após escrever o programa e gravá-lo com o editor de texto escolhido em um dos sistemas operacionais de sua escolha, deve-se compilar o programa utilizando, respectivamente, um dos seguintes comandos indicados:

- `gpp [<fonte>] -o [<executável>]`
- `g++ -o [<executável>] [<fonte>]`

A primeira forma é usada caso esteja instalado o compilador DJGPP e a segunda quando em uso o sistema operacional Linux ou quando instalado no Windows o compilador MinGW.

Se o programa-fonte estiver escrito sem erro, ele será compilado sem que nenhuma mensagem seja apresentada. No sistema operacional Microsoft Windows (modo texto) a chamada do programa é efetuada pelo nome atribuído no parâmetro **executável**. No sistema operacional Linux o processo é semelhante, no entanto é necessário colocar à frente do nome do programa executável os caracteres **./** (ponto e barra).

O GCC é uma ferramenta de trabalho que opera em 100% em relação a conformidade com a norma internacional. Esta ferramenta supera em qualidade muitos produtos consagrados comercialmente.

O compilador DJGPP pode apresentar efeitos visuais indesejáveis quando executado no sistema operacional Microsoft Windows 7 de 32 *bits* ou Microsoft Vista 64 *bits* (não é executado no Windows 7 de 64 *bits*). Nesse caso, aconselha-se a fazer uso do compilador **TDM-GCC** que opera com o MinGW ou o ambiente **Code::Blocks** com o compilador MinGW embutido. Para uso nos modos de 64 *bits* do Microsoft Windows, sugere-se o compilador **TDM-GCC** com o ambiente **Code::Blocks** sem o compilador MinGW embutido. O DJGPP é mais adequado para uso no sistema operacional Microsoft Windows XP.

Para efetuar o estudo da linguagem de programação de computadores C++ abordado nesta obra, é de fundamental importância possuir instalado em seu computador um compilador[2] da linguagem de programação em uso. Nesse caso, um compilador que consiga trabalhar com o código da linguagem C++ e, de preferência, que respeite o padrão ANSI em cem por cento. O compilador

---

[2] *Um programa compilador é uma ferramenta que transforma um programa-fonte (escrito em linguagem de alto nível) em um programa-objeto (em linguagem de baixo nível), anexando este às bibliotecas de run-time do sistema operacional, tornando-o um programa executável. Em ambiente MS-DOS/Windows, um programa executável possui normalmente a extensão .EXE. Em ambientes UNIX (Linux, BSD etc.), um programa executável não possui obrigatoriamente nenhuma extensão de identificação.*

GCC é a melhor ferramenta para este estudo, pois além de atender à norma, foco deste trabalho, é distribuído em regime de *software livre*.

Um compilador para uma linguagem de programação pode ser comprado no mercado especializado em produtos para a área de Tecnologia da Informação, ou ser baixado diretamente da Internet, o qual pode ser encontrado em versões *trial* (o produto possui todos os recursos disponíveis por espaço de tempo predeterminado), *demo* (possui alguns de seus recursos habilitados)*, freeware* (possui todos os recursos disponíveis sem nenhum custo para o seu usuário) ou *shareware* (possui todos os recursos disponíveis e pode ser utilizado a título de experimentação por um determinado tempo, após o que deve ser comprado).

2.2 GCC (DJGPP)

Usuários do sistema operacional Microsoft Windows XP, Microsoft Windows Vista - 32 *bits* e Microsoft Windows 7 - 32 *bits* que façam uso do modo console por intermédio da janela do *prompt do MS-DOS*, podem utilizar diretamente o compilador **DJGPP**, distribuído em regime *freeware*. O *DJGPP* (é uma versão do compilador GCC para plataforma Microsoft em 32 *bits*) é acessado em linha de comando, o que exige certo grau de conhecimento da manipulação dos comandos do modo console.

Para fazer a aquisição do compilador *DJGPP* para o sistema operacional Microsoft Windows XP*,* basta acessar o sítio **http://www.delorie.com/djgpp/** (a operação aqui descrita foi realizada em novembro de 2012. Não se pode garantir que em data futura os procedimentos descritos irão ocorrer da mesma forma ou que o referido sítio esteja disponível para acesso).

O compilador *DJGPP* possui uma extensa relação de arquivos, o que muitas vezes pode confundir usuários leigos, pois esses arquivos não possuem um programa de instalação que faça esse trabalho. Todo o processo deve ser executado manualmente. Apesar de parecer difícil para alguns, é uma tarefa muito simples de ser realizada com a vantagem de não afetar o arquivo de registro do Microsoft Windows.

Para efetuar *download* dos arquivos necessários, selecione no menu lateral esquerdo a opção **Zip Picker**, situada na área **Introduction**. Esse recurso permite, por meio do selecionamento de algumas opções, saber quais arquivos serão copiados para posterior instalação em seu computador.

No campo "**Select a suitable FTP site**", selecione o local de onde os arquivos serão copiados. No campo "**Pick one of the following**", mantenha a seleção da opção **Build and rum programs with DJGPP**; no campo "**Which operating system will you be using?**", mantenha a seleção da opção **Windows 2000/XP**; e, para o campo "**Do you want to be able to read the on-line documentation?**", mantenha a seleção **Yes**.

Para a pergunta "**Which programming languages will you be using?**", mantenha a seleção **C** e marque pelo menos a opção **C++**. Assim, terá acesso aos doze arquivos mínimos para a instalação e execução do compilador. Em seguida, acione o botão "**Tell me which files I need**" e será apresentada a relação de arquivos a serem baixados.

Faça o *download* de todos os arquivos indicados para o diretório **djgpp** que deve ser previamente criado em seu computador, depois faça a descompactação dos arquivos com o programa **unzip32.exe** por meio da sintaxe **unzip32 –o *.zip**. É aconselhável que essa ação seja realizada no *prompt* da linha de comando do sistema.

A próxima etapa da instalação é estabelecer a configuração das variáveis de ambiente para a execução do compilador. Assim sendo, no *desktop* do sistema, dê um clique com o *mouse* no ícone **Meu computador**, no menu de contexto apresentado selecione a opção **Propriedades**. Selecione a guia **Avançado** e acione neste instante o botão **Variáveis de ambiente**.

Na caixa de diálogo **Variáveis de ambiente**, observe na área **Variáveis do usuário para <Nome do usuário ativo>** a lista de variáveis apresentada. Selecione a variável **PATH** e acione o botão **Editar**. Inclua no final da lista um caractere ponto e vírgula, após informe **C:\DJGPP\BIN** e acione o botão **OK**. Em seguida, acione o botão **Novo**, informe no campo **Nome da variável** a nova variável de ambiente chamada **DJGPP** e no campo **Valor da variável** informe **C:\DJGPP\DJGPP.ENV**. Acione o botão **OK** para fechar a caixa de diálogo de criação de variáveis e retornar à caixa de diálogo **Variáveis de ambiente** e então acione o botão **OK** para sair dessa caixa de diálogo. O ambiente está configurado e pronto para ser usado.

No caso de instalações em modo MS-DOS, o processo de configuração é mais simples, pois basta acrescentar as variáveis de ambiente ao arquivo **autoexec.bat** ou criar um arquivo **.bat** para ser executado antes de usar o

compilador. Por exemplo, a partir de um arquivo criado dentro do diretório **C:\DJGPP** com o nome **meugcc.bat**.

```
@echo off
set PATH=c:\djgpp\bin;%PATH%
set DJGPP=c:\djgpp\djgpp.env
cd c:\djgpp\fontes
```

No caso de menção ao diretório **fontes**, presume-se sua criação no diretório **C:\DJGPP** com o comando **md fontes**.

A partir desse ponto, basta entrar no diretório do programa com o comando **cd C:\DJGPP** e executar o arquivo **meugcc.bat** para preparar o ambiente para o devido uso.

O usuário do sistema operacional Linux não precisa se preocupar em fazer a aquisição do compilador, pois o GCC é fornecido junto com as muitas distribuições de Linux existentes, desde que, no momento da instalação do sistema, se faça também a instalação do compilador GCC.

No caso do sistema operacional Linux, o compilador é executado a partir da linha de comando do *prompt* do sistema e possui os *front-ends* de acesso aos recursos das linguagens de programação C, C++ e FORTRAN. A instalação em Linux é mais simples que a instalação no sistema operacional Microsoft Windows. Esta pode ser realizada a partir dos comandos **yum** ou **apt-get** como superusuário, dependendo do padrão em que o sistema está baseado **Red Hat** ou **Debian**.

```
su
yum install gcc-c++ (Fedora)

sudo
apt-get install gcc-c++ (Debian)
```

Após a execução dos comandos basta aguardar a conclusão da instalação e da configuração que ocorre de forma automática.

2.3 TDM-GCC

O compilador **TDM-GCC** é um programa de linha de comando composto pelo compilador GCC e pelo recurso MinGW para os modos 32/64 *bits* dos sistemas operacionais Windows. Pode ser usado por todas as edições do Microsoft Windows a partir da versão XP.

Caracteriza-se por ser um ambiente integrado de desenvolvimento do tipo IDE (Integrated Development Environment) multiplataforma (Microsoft Windows, Linux e Mac OS), criado para ser usado em conjunto com diversos compiladores de linguagem C e C++, entre eles o compilador **GCC**. Essa ferramenta de trabalho pode ser usada tanto no sistema operacional Microsoft Windows como no sistema operacional Linux. Em ambos os casos, o programa deve ser baixado e instalado.

Para fazer a aquisição do compilador TDM-GCC para os sistemas operacionais Microsoft Windows Vista, 7 e 8, ambos de 32/64 bits, basta acessar o sítio **http://tdm-gcc.tdragon.net/** (a operação aqui descrita foi realizada em agosto de 2013. Não se pode garantir que em data futura os procedimentos descritos irão ocorrer da mesma forma ou que o referido sítio esteja disponível para acesso).

Selecionar no menu superior da página a opção **Download** e quando esta for apresentada selecionar a opção **tdm64-gcc-4.7.1-3** e aguarde o arquivo ser copiado para seu computador.

Após ter concluído a cópia do programa, vá até a pasta onde o mesmo foi copiado e execute-o com um duplo clique do ponteiro do *mouse*, escolha o botão **Create** e siga as instruções apresentadas. Atente para o momento em que é solicitada a escolha da versão do compilador a ser instalada. Opte pela opção que menciona o modo 64 *bits* se estiver em uso um Windows de 64 *bits*. Siga as demais instruções até concluir a instalação do programa.

A próxima etapa da instalação é estabelecer a configuração das variáveis de ambiente para execução do compilador. Assim sendo, no *desktop* do sistema dê um clique com o *mouse* no ícone **Computador** (Microsoft Windows Vista ou 7), e selecione **Propriedades** no menu de contexto apresentado. Em seguida selecione a opção **Configurações avançadas do sistema** que apresentará a guia **Avançado** já selecionada. Acione neste instante o botão **Variáveis de ambiente**.

Na caixa de diálogo **Variáveis de ambiente**, observe na área **Variáveis do usuário para <Nome do usuário ativo>** a lista de variáveis apresentada. Selecione a variável **PATH** e acione o botão **Editar**. Inclua no final da lista um caractere ponto e vírgula, após informe **C:\MINGW64\BIN** e acione o botão **OK**. Acione o botão **OK** para fechar a caixa de diálogo de criação de variáveis e retornar à caixa de diálogo **Variáveis de ambiente** e então acione o botão **OK** para sair dessa caixa de diálogo. O ambiente está configurado e pronto para ser usado.

2.4 IDE: Code::Blocks

O programa **Code::Blocks** caracteriza-se por ser um ambiente integrado de desenvolvimento do tipo IDE (Integrated Development Environment) múltipla-taforma (Microsoft Windows, Linux e Mac OS) criado para ser usado em conjunto com diversos compiladores de linguagem C e C++, entre eles o compilador **GCC**.

Essa ferramenta de trabalho pode ser usada tanto no sistema operacional Microsoft Windows como no sistema operacional Linux. Em ambos os casos, o programa deve ser baixado e instalado.

O **Code::Blocks** é uma ferramenta de trabalho distribuída sob a licença GPL, ou seja, assim como o **GCC**, essa ferramenta de trabalho também pode ser adquirida de forma livre, sem nenhum ônus, a menos que o usuário deseje fazer alguma doação aos desenvolvedores do programa.

Para a aquisição do programa **Code::Blocks**, acesse o endereço **http://www.codeblocks.org**, como mostra a figura 2.1 e selecione no menu lateral esquerdo a opção **Downloads**.

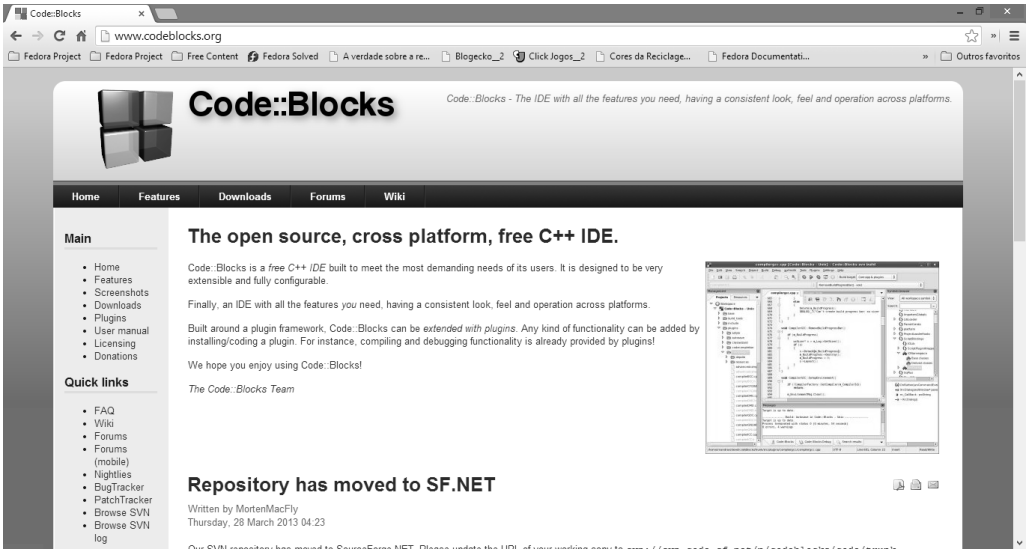

Figura 2.1 - Página Code::Blocks.

Na página *Downloads*, selecione o *link* **Download the binary release**, em seguida selecione o arquivo de tamanho maior (**codeblocks-10.05mingw-setup.exe**) se estiver usando o sistema operacional Microsoft Windows 7 ou 8 de 32 *bits* ou o sistema operacional Microsoft Windows Vista de 32/64 *bits*. Após obter o arquivo desejado, basta proceder com sua instalação de acordo com as regras do sistema operacional. No entanto, se estiver utilizando o sistema operacional **Fedora Linux** basta na janela **Terminal** executar o comando **su -c 'yum install codeblocks'**, entre a senha de superusuário e efetue a instalação dos recursos indicados como dependências, lembrando que para essa ação deverá o computador estar conectado à Internet.

Caso esteja usando o sistema operacional Microsoft Windows 7 ou 8 de 64 *bits*, escolha o arquivo de tamanho menor **codeblocks-10.05-setup.exe**, desde que já tenha adquirido e instalado o compilador TDM-GCC.

No sistema operacional Microsoft Windows, vá à pasta que fora usada para baixar o programa e selecione o arquivo **codeblocks-10.05mingw-setup.exe** e confirme com o botão do *mouse* as etapas apresentadas até o término da operação.

Após a instalação do programa **Code::Blocks**, para fazer uso deste execute uma das seguintes ações:

- No sistema operacional Microsoft Windows: selecione o botão **Menu Iniciar**, na opção **Todos os Programa** selecione a pasta **CodeBlocks** e escolha **CodeBlocks**. A figura 2.2 mostra a tela do programa no sistema operacional **Microsoft Windows**;
- No sistema operacional Fedora Linux com ambiente GNOME: selecione a opção principal de menu **Aplicativos**, selecione a opção **Desenvolvimento** escolha **Code::Blocks IDE**. A figura 2.3 mostra a tela do programa no sistema operacional **Fedora Linux**.

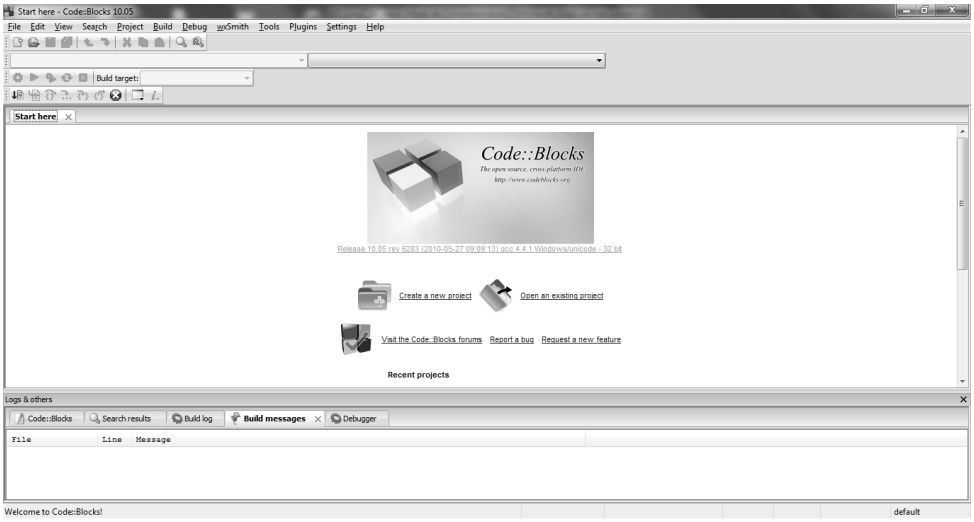

Figura 2.2 - Tela do programa Code::Blocks no Microsoft Windows.

Assim que o ambiente é executado pela primeira vez, após sua tela de *splash* é apresentada a caixa de diálogo de autodetecção, indicando o compilador detectado, nesse caso **GNU GCC Compiler**. Nesta etapa, acione no botão **OK** e será apresentada a tela do programa com caixa de diálogo **Tip of the Day** (dica do dia) e a janela **Scripting console**.

Para **Tip of the Day** desmarque a opção **Show tips at startup** (mostrar dicas ao entrar) e feche-a. Feche a janela **Scripting console** e ao lado esquerdo feche a janela **Management**.

Para uso básico e essencial do ambiente é necessário conhecer os seguintes comandos de operação:

- **File/New/Empty file** - para criar uma folha nova de trabalho onde um código fonte deverá ser escrito;

- **File/Save file** - para gravar o arquivo atual em edição, use **.cpp** no nome do arquivo para identificar um arquivo de programa em linguagem C++;
- **File/Save file as** - para gravar o arquivo atual em edição com outro nome;
- **File/Print** - para imprimir o arquivo de programa em edição;
- **Build/Build and run** - para compilar e executar o programa em edição;
- **File/Quit** - para sair do programa.

Além desses comandos, existem outros comuns em editores de texto, tais como: **Edit/Copy** (copiar), **Edit/Paste** (colar), **Edit/Cut** (cortar), **Search/Find** (procurar) e **Search/Replace** (trocar).

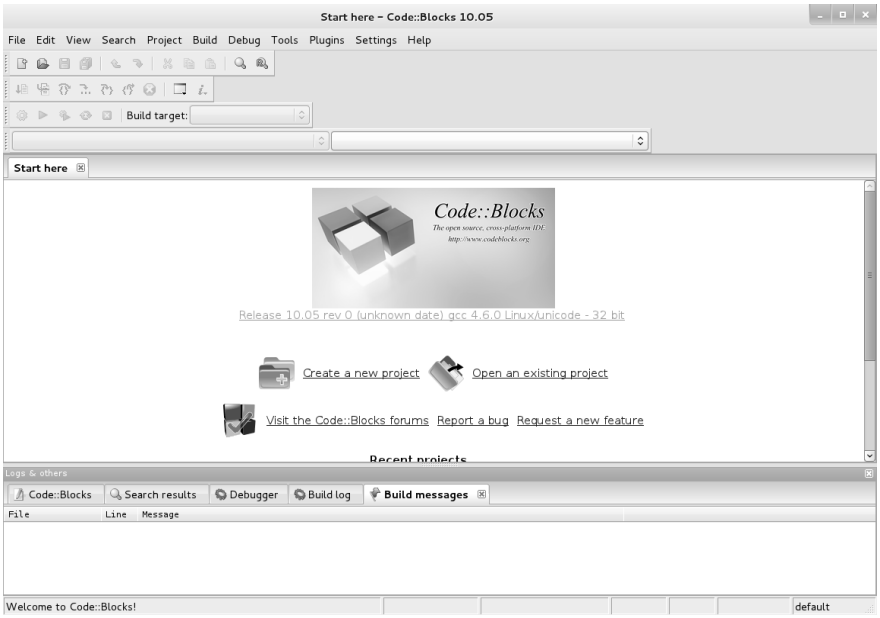

Figura 2.3 - Tela do programa Code::Blocks no Fedora Linux.

O **Code::Blocks**, quando usado para compilar e executar um programa no sistema operacional Microsoft Windows, apresenta o resultado da execução na janela de **Prompt de comando** com fundo preto e letras em tom branco. Já, no sistema operacional Linux, apresenta o resultado da execução na janela **xterm** com fundo branco e letras em preto (dependendo da distribuição Linux em uso ou versão, essa ocorrência poderá ser diferente). Caso queira ver no sistema operacional Linux a janela **xterm** com fundo preto e letras brancas, é necessário alterar uma configuração do ambiente. Nesse caso,

selecione no menu **Settings** a opção **Enviroment** e no campo **Shell to run commands in** altere a linha "**xterm -T $TITLE -e**" para "**xterm -bg black -fg white -T $TITLE –e**" e acione em seguida o botão **OK**.

A ferramenta **Code::Blocks** nesta obra é apenas uma indicação de uso e não será tratada como ferramenta de trabalho dos programas apresentados. Toda a execução de compilação será realizada a partir da linha de comando. Fica a critério do leitor fazer ou não uso dessa ferramenta, sendo que não será dada nenhuma indicação de funcionalidade e/ou configuração desta ferramenta nesta obra. Vale salientar que alguns programas apresentados nesta obra não serão executados adequadamente dentro da ferramenta **Code::Blocks**, pois irão necessitar ajustes e configurações que fogem do escopo deste trabalho.

No caso da instalação do arquivo com o GCC embutido, não há a necessidade de configurar. No entanto, em relação ao arquivo de tamanho menor para o sistema operacional Microsoft Windows 7 ou 8 de 64 *bits* (ou se instalou o TDM-GCC em outra versão do sistema operacional da Microsoft), é necessário proceder, conforme alguns passos de configuração.

A próxima etapa da instalação é estabelecer a configuração das variáveis de ambiente para execução do compilador. Assim sendo, no *desktop* do sistema dê um clique com o *mouse* no ícone **Computador**, e selecione **Propriedades** no menu de contexto apresentado. Em seguida, selecione a opção **Configurações avançadas do sistema** que apresentará a guia **Avançado** já selecionada. Acione neste instante o botão **Variáveis de ambiente**.

Na caixa de diálogo **Variáveis de ambiente**, observe na área **Variáveis do usuário para <Nome do usuário ativo>** a lista de variáveis apresentada. Selecione a variável **PATH** e acione o botão **Editar**. Inclua no final da lista um caractere ponto e vírgula, após informe **C:\MINGW64\BIN** e acione o botão **OK**. Acione o botão **OK** para fechar a caixa de diálogo de criação de variáveis e retornar à caixa de diálogo **Variáveis de ambiente** e então acione o botão **OK** para sair dessa caixa de diálogo. O ambiente está configurado e pronto para ser usado.

Em seguida, é necessário configurar o programa **Code::Blocks**. Para tanto, selecione o menu **Settings** e escolha a opção **Compiler and debugger**. Será apresentada a janela **Compiler and debugger settings**, selecione primeiramente a guia **Toolchain executables**, informe para o campo **Compiler's**

install directory o local do compilador **C:\MINGW64**, acione em seguida o botão **Auto-detect**. Se for apresentada uma caixa de mensagem com uma advertência de alteração acione o botão **Não** (**No**).

Verifique se, na guia **Program Files**, são apresentadas as seguintes informações para cada um dos campos existentes:

- C compiler: x86_64-w64-mingw32-gcc.exe
- C++ compiler: x86_64-w64-mingw32-g++.exe
- Linker for dynamic libs: x86_64-w64-mingw32-g++.exe
- Linker for static libs: ar.exe
- Debugger: gdb.exe
- Resource compiler: windres.exe
- Make program: make.exe ou mingw32-make.exe

Caso essas informações não sejam apresentadas, será necessário informá-las manualmente. Para tanto, use o botão **[...]** existente ao lado direito de cada campo e localize na pasta **C:\MINGW64\BIN** os arquivos indicados que não estejam apresentados e acione o botão **OK** e a configuração estará pronta.

Se ocorrer algum erro de compilação após os procedimentos anteriores, talvez seja necessário acrescentar algumas configurações adicionais. Assim sendo, execute o comando de menu **Settings/Compiler and debugger**, na guia **Search directories** selecione a aba **Compiler**.

Apague com a tecla **<Del>** qualquer informação existente na aba **Compiler** e acione o botão **Add** informando em seguida o diretório **C:\MINGW64\INCLUDE**. Na aba **Linker**, proceda de maneira semelhante à aba **Compiler** e informe o diretório **C:\MINGW64\LIB**; em seguida, na aba **Resource compiler**, informe o diretório **C:\MINGW64\INCLUDE** e acione o botão **OK**.

Anotações

3

capítulo

Programação Sequencial

Objetivos

Este capítulo traz informações iniciais sobre o uso preliminar da linguagem de programação C, como tipos de dados, variáveis, constantes, operadores aritméticos, expressões aritméticas, instruções de entrada e de saída, além do processo de formatação dos dados manipulados pelos programas.

São utilizados os comandos:

- *char*
- *double*
- *float*
- *int*
- *long*
- *unsigned*
- *signed*

São utilizadas as funções:

- *main()*
- *printf()*
- *scanf()*

Operadores aritméticos:

- *adição (+)*
- *subtração (-)*
- *multiplicação (*)*
- *divisão (/)*
- *atribuição (=)*

A diretiva de cabeçalho:

- *#include*

3.1 Instruções

Para desenvolver um programa que seja executável em um computador, é necessário ter em mente que o trabalho de gerenciamento de dados ocorre em três níveis:

1. entrada dos dados;
2. processamento (transformação) dos dados em informação;
3. saída de informações.

O trabalho de gerenciamento de dados em um computador executa instruções que devem estar ordenadas de forma lógica para realizar uma determinada ação e atender certa necessidade.

As instruções de operação na linguagem C são obtidas a partir de comandos e funções. A seguir apresentam-se alguns dos comandos e funções utilizados nesta obra.

- Os comandos são definidos por palavras reservadas da linguagem escritas de forma simples, podendo-se destacar break, case, char, default, do, double, else, FILE, float, if, include, int, long, return, signed, stdin, struct, switch, unsigned e while.
- As funções são definidas por palavras reservadas da linguagem seguidas de parênteses, podendo-se destacar fclose(), fflush(), fgets(), fgets(), fopen(), for(), fprintf(), fread(), fscanf(), fseek(), fwrire(), printf(), puts(), scanf(), strcmp() e strcpy().

Tanto comandos como funções da linguagem C são escritos com caracteres em formato minúsculo, exceto o comando FILE, e assim devem ser definidos, pois C é uma linguagem considerada *case sensitive*, ou seja, suas instruções levam em conta a diferenciação dos caracteres. Caso escreva comandos ou funções em maiúsculo, obterá erro de compilação, exceto o comando FILE que deve sempre ser grafado em maiúsculo.

3.2 Tipos de Dados

Os **dados** são representados por elementos brutos a serem processados por um computador, a fim de se transformarem em informação. A linguagem C tem a capacidade de trabalhar com tipos de dados predefinidos com **char**, **int**, **float** ou **double**. Além dos tipos de dados, a linguagem C possui para formatá-los

os chamados *modificadores*, que orientam como o espaço de um dado é alocado na memória, destacando-se: **signed** (sinalizado), **unsigned** (não sinalizado), **long** (longo) e **short** (curto).

Os modificadores *signed*, *unsigned*, *long* e *short* são usados em conjunto com os tipos de dados **char**, **int** e **double**, podendo ser combinados entre si, desde que respeitadas suas abrangências de operação como podem ser observadas nos subtópicos a seguir.

3.2.1 Dados Inteiros

São inteiros os dados numéricos positivos ou negativos, excluindo qualquer número fracionário (números do tipo real). Em linguagem C esse tipo de dado pode ser referenciado por um dos seguintes identificadores:

Tabela 3.1 - Tipos de dados inteiros

Tipo de dado inteiro	Faixa de abrangência	Tamanho
unsigned short int	de 0 a 65.535	16 bits
short / short int / signed int / signed short int	de -32.768 a 32.767	16 bits
unsigned int / unsigned long int	de 0 a 4.292.967.295	32 bits
int / long / long int / signed long int	de -2.147.483.648 a 2.147.483.647	32 bits

Os valores numéricos **int** ocupam 2 *bytes* (16 *bits*) quando utilizados em sistemas-padrão operacionais MS-DOS. Em sistemas operacionais de 32 *bits*, como o Microsoft Windows ou Linux, seu tamanho equivale a 4 *bytes* (32 *bits*). O termo **unsigned** representa a existência de um valor numérico sem o sinal negativo, enquanto os inteiros sem a definição de **unsigned** aceitam valores negativos. Na maior parte dos programas-exemplo desta obra, será utilizado o dado **int** ou **long int**.

Observação

O tipo de dado inteiro **long**, quando em uso no sistema operacional Linux x86-64, opera em 64 bits. No Linux i386 ou qualquer sistema operacional Microsoft Windows, opera em 32 bits.

3.2.2 Dados Reais

São reais os dados numéricos positivos, negativos, números fracionários e também inteiros. Em linguagem C, esse tipo de dado pode ser referenciado por um dos seguintes identificadores:

Tabela 3.2 - Tipos de dados reais

Tipo de dado real	Faixa de abrangência	Tamanho
float	de -3.4^{38} a 3.4^{38}	32 bits
double	de -1.7^{308} a 1.7^{308}	64 bits
long double	de -3.4^{4932} a 1.1^{4932}	96 bits

O tipo de dado real permite trabalhar com uma representação de valores em ponto flutuante que consiste em uma parte inteira e uma mantissa (parte fracionária).

3.2.3 Dados Caracteres

São classificados como caracteres os dados inteiros que representam números e símbolos especiais delimitados com aspas simples (' '). Esse tipo de dado é referenciado pelo identificador **char**, podendo-se armazenar esses caracteres numa variável simples, com base na estrutura da tabela de caracteres do computador, chamada tabela ASCII (American Standard Code Information Interchange).

Tabela 3.3 - Tipos de dados caracteres

Tipo de dado inteiro	Faixa de abrangência	Tamanho
char / signed char	de –128 até 127	8 bits
unsigned char	de 0 até 255	8 bits
char	Pode ser considerado *signed char* ou *unsigned char*, dependendo do sistema	8 bits

Enquanto caracteres são delimitados entre aspas simples, a definição de uma sequência de caracteres (os *strings*) deve ser feita entre aspas inglesas (" "), erroneamente chamadas de aspas duplas. Este tema será tratado adiante em capítulo específico, pois trata-se de uso de variáveis compostas.

3.2.4 Dados Lógicos

Na linguagem C não existe um dado lógico ou booleano propriamente dito, ou seja, não existirão os valores lógicos *Falso* ou *Verdadeiro* para um determinado dado, de forma predefinida. Nessa linguagem qualquer valor igual a **0** (zero) é considerado um valor lógico **falso**, e qualquer valor diferente de zero, que será representado pelo número **1** (um) inteiro, será considerado um valor lógico **verdadeiro**. Não se preocupe com este detalhe agora, pois mais à frente você terá a oportunidade de usá-lo.

3.3 O Uso de Variáveis

Variável, do ponto de vista de programação, é uma região de memória previamente identificada que tem por finalidade armazenar os dados ou informações de um programa por um determinado tempo. Uma variável limita-se a armazenar apenas um valor por vez. Sendo considerado como valor o conteúdo de uma variável, desta forma, um valor está relacionado ao tipo de dado de uma variável, que pode ser numérico, lógico ou caractere.

O nome de uma variável é utilizado para sua identificação e posterior uso em um programa. Sendo assim, é necessário estabelecer algumas regras de utilização de variáveis:

- Nomes de variáveis podem ser atribuídos com um ou mais caracteres.
- O primeiro caractere do nome de uma variável não pode ser, em hipótese nenhuma, um número; sempre deve ser uma letra ou, no caso da linguagem C, pode-se iniciar o nome de uma variável com o caractere *underline* "_".
- O nome de uma variável não pode ter espaços em branco.
- Não pode ser nome de uma variável uma palavra reservada a uma instrução ou identificador de uma linguagem de programação.
- Não podem ser utilizados outros caracteres em nomes de variáveis a não ser letras, números e o caractere *underline*, utilizado normalmente para simular a separação de palavras compostas como nomes de variáveis, como, por exemplo, NOME_ALUNO.
- Na linguagem C, os 32 primeiros caracteres de um nome de variável são significativos, porém se o sistema operacional em uso for o MS-DOS, somente serão considerados os primeiros oito caracteres. Este fator limita um pouco o uso das variáveis.

- Outro detalhe a ser considerado na linguagem C é a diferença entre caracteres maiúsculos e minúsculos. Sendo assim, as variáveis NOME, nome, Nome, noMe são diferentes.

É necessário ainda considerar que em um programa uma variável pode exercer dois papéis de trabalho. Um de ação, quando é modificada ao longo de um programa para apresentar um determinado resultado, e o outro de controle, em que ela pode ser "vigiada" e controlada durante a execução de um programa (esse tipo de variável será estudado nos capítulos que abordam a tomada de decisões e o uso de laços).

Todo dado a ser armazenado na memória de um computador por meio de uma variável utilizando a linguagem C deve ser previamente declarado, ou seja, primeiramente é necessário saber o seu tipo para depois fazer o seu armazenamento. Armazenado o dado, ele pode ser utilizado e manipulado a qualquer momento durante a execução do programa.

Observação

Há ainda o tipo de dado **void** (vácuo ou vazio) utilizado para declarar funções que não retornam valor. A linguagem C é montada com base em um grande conjunto de funções. Ao longo desta obra este conceito ficará claro, pois é usado intensamente.

3.4 O Uso de Constantes

Constante é tudo aquilo que é fixo ou estável, e em vários momentos este conceito deve estar em uso. Por exemplo, o valor 1.23 da fórmula seguinte é uma constante: *RESULTADO* = *ENTRADA* * 1.23.

Em um programa, uma constante pode também fazer referência ao contexto implícito. Uma constante é implícita quando é declarada em um programa codificado em linguagem de programação C, por meio da cláusula **define** (**#define** <rótulo de identificação> <valor constante>), a qual será utilizada bem mais adiante, nesta obra.

3.5 Os Operadores Aritméticos

Tanto variáveis como constantes podem ser utilizadas na elaboração de cálculos matemáticos com a utilização de operadores aritméticos. Os operadores aritméticos são classificados em duas categorias, sendo **binários** ou

unários. São binários quando atuam em operações de exponenciação, multiplicação, divisão, adição e subtração. São unários quando atuam na inversão de um valor, atribuindo a este o sinal positivo ou negativo.

A tabela seguinte apresenta um resumo dos operadores aritméticos utilizados na linguagem C:

Tabela 3.4 - Operadores aritméticos

Operador	Operação	Tipo	Resultado
+	Manutenção de sinal	Unário	-
-	Inversão de sinal	Unário	-
%	Resto de divisão (Módulo)	Binário	Inteiro
/	Divisão	Binário	Inteiro ou Real
*	Multiplicação	Binário	Inteiro ou Real
+	Adição	Binário	Inteiro ou Real
–	Subtração	Binário	Inteiro ou Real
+ +	Incremento	Unário	Inteiro ou Real
– –	Decremento	Unário	Inteiro ou Real
+ =	Adição cumulativa	Unário	Inteiro ou Real
– =	Subtração cumulativa	Unário	Inteiro ou Real
* =	Multiplicação cumulativa	Unário	Inteiro ou Real
/ =	Divisão cumulativa	Unário	Inteiro ou Real
% =	Módulo cumulativo	Unário	Inteiro ou Real
Pow (base expoente)	Exponenciação	Binário	Real
Sqrt (valor numérico)	Raiz quadrada	Unário	Real

Observação

Para utilizar as funções **pow()** e **sqrt(),** é necessário que a biblioteca **math.h** seja inicialmente incluída no programa com o comando **#include** antes da função principal **main()**, utilizando a sintaxe **#include <math.h>**.

3.6 As Expressões Aritméticas

É muito comum a necessidade de trabalhar com expressões aritméticas, uma vez que, na sua maioria, todo trabalho computacional está relacionado e envolve a utilização de cálculos matemáticos. Essas expressões são definidas pelo relacionamento existente entre variáveis e constantes numéricas com a utilização dos operadores aritméticos.

Considere a fórmula **AREA = π . RAIO^2** para o cálculo da área de uma circunferência, em que estão presentes as variáveis *AREA* e *RAIO*, a constante π (pi = 3.14159) e os operadores aritméticos de multiplicação e também a operação de potência, elevando o valor da variável *RAIO* ao quadrado.

As expressões aritméticas escritas para a execução em um computador seguem um formato um pouco diferente do conhecido em matemática. Por exemplo, a expressão **X = {43 . [55 : (30 + 2)] }** será escrita em C++ como **X = (43 * (55 / (30 + 2)))**. Perceba que as chaves e os colchetes são abolidos, utilizando em seu lugar apenas os parênteses. No caso da fórmula para o cálculo da área de uma circunferência, seria escrita como **AREA = 3.14159 * RAIO * RAIO**.

E se a fórmula a ser utilizada fosse para calcular a área de um triângulo, em que é necessário efetuar a multiplicação da base pela altura e em seguida dividir pela constante 2, como ficaria? Observe a fórmula-padrão:

$$\mathbf{A} = \frac{\mathbf{b} \cdot \mathbf{h}}{\mathbf{2}}$$

Ela deveria ser escrita como **A = (B * H) / 2**.

3.7 Estrutura de um Programa em C

Todo programa escrito em C consiste em uma ou mais funções, tendo como particularidade a possibilidade de construir programas modulares e estruturados. O programa principal escrito em C é uma função. Os nomes, programas, rotinas, sub-rotinas, procedimentos, instruções podem confundir-se em C (uma vez que C é uma linguagem extremamente estruturada e exige do programador domínio adequado de tal conceito) com o termo função. Veja a seguir o menor programa possível de ser escrito em C.

```
int main(void)
{
  return 0;
}
```

em que:

int - definição obrigatória do valor de retorno da função **main()**, principal função de operação de um programa C;

main - a primeira e principal função a ser executada;

() - parênteses usados para indicar parâmetros a serem passados para a função **main()**, podendo ser **void** quando se trata de parâmetros inexistentes (às vezes omitido, mas não nesta obra) ou o uso dos parâmetros formais **argc** e **argv** a partir de **int argc, char *argv[]**, usado quando se deseja estabelecer interação do programa C com a linha de comandos do sistema operacional (será visto mais adiante);

void - indica lista vazia de parâmetros/argumentos;

{ - inicia ou abre o corpo da função (programa);

return - definição do retorno de valor para o tipo **int** de **int** main(void), sendo esse retorno normalmente configurado com valor **0** para a finalização da função **main()**;

} finaliza ou fecha o corpo da função.

Observação

A linguagem C não possui uma forma obrigatória de escrita, o que pode causar alguns problemas de legibilidade. No livro você encontra uma forma de endentação que certamente é de fácil leitura.

A função **main()** é a principal instrução a ser considerada em um programa escrito na linguagem C, ou seja, ela deve estar presente em algum lugar do programa, pois marca o ponto de inicialização do processo de execução do programa. É apresentado em seguida o modelo de escrita de um programa em C com a definição e comentário de cada um dos elementos básicos do programa.

```
[<definições de pré-processamento - cabeçalhos>]
[<declaração das variáveis globais>]
int main([<parâmetros>])
{

  /*
   Este trecho é reservado para o corpo da função, com a declaração
   de suas variáveis locais, seus comandos e funções de trabalho
  */
  return 0;
}
[
  [<tipo>] funçãonome([<parâmetros>])
  [<declaração de parâmetros>]
  {
    /*
    Este trecho é reservado para o corpo da funçãonome, com a
    declaração de suas variáveis locais, seus comandos e
    funções de trabalho
    */
    [return[valor]]
  }
]
```

Por se tratar **main()** de uma função, por característica-padrão ANSI, ela devolve um valor do tipo inteiro ao programa chamador (no caso o programa principal, representado pelo sistema operacional). Normalmente esse valor é considerado **0** por grande maioria dos compiladores de linguagem C (na prática, para evitar problemas, ao final da função **main()** utiliza-se **return(0)** para forçar o retorno do valor como zero e cumprir as exigências do padrão ANSI C).

A função **main()** pode ainda ser declarada com uma lista de parâmetros dentro de seus parênteses. Na maior parte dos programas desenvolvidos isso não acontece. Nestes casos normalmente se declara o parâmetro como **void** (indefinido), utilizando a sintaxe **int main(void)**.

Informações situadas entre colchetes "[" e "]" podem ou não estar presentes em um programa. Já qualquer informação entre os símbolos de maior que "<" e menor que ">" é obrigatória. Os símbolos de chaves "{" e "}" indicam, como já mencionado, início e fim da rotina.

Todas as informações escritas entre os delimitadores /* e */ são comentários para documentação do programa e não são compiladas quando da execução do programa pelo compilador.

Normalmente, um programa em C é iniciado com as definições de pré-processamento e com a declaração das variáveis globais, lembrando que

essas variáveis podem ser usadas em qualquer parte do programa, seguidas da declaração de todas as funções a serem utilizadas no programa.

As funções em C são formadas inicialmente pela declaração de seu tipo, informação esta opcional, e de seu nome seguida da lista de parâmetros entre parênteses. A lista de parâmetros pode estar vazia, sendo a finalidade de um parâmetro servir como um ponto de comunicação bidirecional entre as várias funções de um programa.

3.7.1 Programa Adição

Desenvolva em linguagem C um programa que leia dois valores numéricos inteiros. Faça a operação de adição dos dois valores e apresente o resultado obtido.

Sempre que o programador estiver diante de um problema, primeiramente deve ser resolvido por ele, para que depois seja resolvido por um computador após implementação de um programa.

Para transformar o problema em um programa de computador, é necessário entender de forma adequada o problema que é apresentado, para depois buscar a solução e assim implementar um programa no computador. Desta forma, o segredo de uma boa lógica de programação está na compreensão adequada do problema a ser solucionado.

Com relação à leitura dos dois valores inteiros (que não são previamente conhecidos), será necessário definir uma variável para cada valor. Assim sendo, serão definidas as variáveis A e B (que representam os dois valores desconhecidos e incógnitos). Em seguida é necessário também definir a variável que guardará a resposta da adição dos dois valores incógnitos. No caso define-se a variável X. O programa em questão deve executar as seguintes ações:

Algoritmo

1. Ler um valor inteiro para a variável A.
2. Ler outro valor inteiro para a variável B.
3. Efetuar a adição dos valores das variáveis A e B e implicar o resultado obtido na variável X.
4. Apresentar o valor da variável X após a operação de adição dos dois valores fornecidos.

No exemplo estão as três fases de trabalho de um programa: a fase de entrada de dados retratada nos passos 1 e 2, o processamento de dados no passo 3 e por fim a saída da informação mostrada no passo 4.

Uma entrada e uma saída podem ocorrer no computador de diversas formas. Por exemplo, uma entrada pode ser feita via teclado, modem, leitores ópticos, disco, entre outros. Uma saída pode ser feita em vídeo, impressora, disco, entre outras formas. Devido à grande variedade, os programas desta obra utilizam na sua maioria as funções **scanf()** (para a entrada de dados) e **printf()** (para a saída de dados) pertencentes ao arquivo de cabeçalho (biblioteca-padrão externa de funções) **stdio.h**, mencionado no programa por meio da diretiva de cabeçalho **#include**.

Considera-se que os programas desta obra sempre vão usar o teclado e o vídeo para efetuar, respectivamente, as operações de entrada e saída.

Completada a fase de interpretação do problema e a definição das variáveis a serem utilizadas, por meio de um algoritmo, passa-se para a fase de elaboração do diagrama de blocos (modo tradicional) ou a utilização do diagrama de quadros (modo NS-Chapin)[3].

Não é o intuito deste trabalho ensinar a confecção de diagramas de blocos ou de quadros, tampouco desenvolver lógica de programação (o leitor deve possuir esse conhecimento), sendo assim será tratada de forma direta a codificação dos programas em linguagem C, tema do livro.

Após considerar os passos anteriores, realiza-se a codificação do programa. São utilizadas no exemplo três variáveis, A, B e X (que neste caso estão sendo trabalhadas no estado de variáveis locais ou privadas), as quais devem ser declaradas segundo o tipo, no caso como inteiras.

```
/* Programa Adiciona Numeros */
#include <stdio.h>
int main(void)
{
  int A;
  int B;
  int X;
```

[3] *Sobre elaboração de diagrama de blocos o leitor pode consultar as obras "Estudo Dirigido de Algoritmos" ou "Algoritmos - Lógica para o Desenvolvimento de Programação de Computadores". Sobre elaboração de diagrama de quadros pode consultar a obra "Lógica Estruturada para Programação de Computadores". As três são publicações da Editora Érica.*

A primeira linha do programa com a indicação **/* Programa Adiciona Numeros */** apresenta uma mensagem, a qual elucida a ação do programa. Essa linha não é obrigatória, porém é aconselhável sempre fazer algum tipo de identificação da ação do programa.

Na segunda linha encontra-se a citação do arquivo de cabeçalho (biblioteca) **stdio.h** por meio da linha de comando **#include <stdio.h>**, a qual estabelece o vínculo do programa com a biblioteca externa (arquivo de cabeçalho) **stdio.h** (situada no diretório **\INCLUDE** do compilador) que possui o suporte ao uso das funções (comandos) de entrada (**scanf**) e saída (**printf**). Todo arquivo com extensão **.h** em linguagem C é de cabeçalho **(.h** de *header*).

Na terceira linha está a função **main()** declarada como **int** (tipo-padrão da principal função em C), a qual estabelece a estrutura do bloco principal do programa. Observe que entre os parênteses da função está sendo utilizado um parâmetro **void** para determinar que nenhum parâmetro está sendo passado.

Observação

Uma boa parte dos compiladores de linguagem C permite declarar a função **main()** sem o uso do **void**, porém há outros compiladores que exigem esta característica. O modo de programação em ANSI C determina que, independentemente da característica do compilador, a função **main()** seja usada com o tipo **void**, quando ela não retornar nenhum valor.

Na função **main()** entre os símbolos "**{**" e "**}**" estão sendo definidas as variáveis de trabalho, bem como as ações de entrada, processamento e saída.

Assim que as variáveis são relacionadas com seus respectivos tipos (no caso variáveis **int**), passa-se para a fase de montagem da ação do programa, em que se efetuam a solicitação da entrada dos dados numéricos, a efetivação do processamento da adição e a apresentação do resultado obtido. As variáveis podem ser também relacionadas em uma única linha **int** *A, B, X*;.

Observe que o bloco de instruções entre as chaves "{" e "}" está deslocado um pouco para a direita. Esse estilo de escrita deve ser obedecido para facilitar a leitura de um bloco de programa, recebendo o nome de endentação.

```
  scanf("%d", &A);
  scanf("%d", &B);
  X = A + B;
  printf("%d", X);
  return 0;
}
```

Após a leitura dos valores para as variáveis A e B com a função **scanf()**, eles são adicionados e implicados (atribuídos) à variável X por meio da linha de programa **X = A + B;**. Em seguida, após efetuar o cálculo, por intermédio da função **printf()** apresenta o resultado da variável X. Note que todas as instruções utilizadas na função **main()** são precedidas de ponto e vírgula, indicando desta forma o fim da instrução naquela linha.

Por último se encontra a instrução **return 0** informando que a função **main()** está retornando o valor zero (valor-padrão para a função). A seguir, é apresentado o programa completo:

```
/* Programa Adiciona Numeros */
#include <stdio.h>
int main(void)
{
  int A;
  int B;
  int X;
  scanf("%d", &A);
  scanf("%d", &B);
  X = A + B;
  printf("%d", X);
  return 0;
}
```

A função **scanf()** possibilita efetuar entrada de dados pelo teclado. A sintaxe dessa função é uma expressão de controle seguida de uma lista de argumentos separados por vírgula, sendo seus argumentos endereços de variáveis.

```
scanf("expressão de controle", lista de argumentos);
```

A expressão de controle contém códigos de formatação para o tipo de dado a ser processado, precedidos pelo sinal de porcentagem **%**. Veja em seguida a tabela com os códigos de formatação para a função **scanf()**.

Tabela 3.5 - Códigos de formatação para função scanf()

Código	Função
%c	Permite que seja efetuada a leitura de apenas um caractere.
%d	Permite fazer a leitura de números inteiros decimais.
%e	Permite a leitura de números em notação científica.
%f	É feita a leitura de números reais (ponto flutuante).
%l	Realiza-se a leitura de um número inteiro longo.
%o	Permite a leitura de números octais.
%s	Permite a leitura de uma série de caracteres.
%u	Efetua-se a leitura de um número decimal sem sinal.
%x	Permite que seja feita a leitura de um número hexadecimal.
%[código]	Permite que seja feita uma entrada formatada pelo código.

Alguns dos códigos de formatação indicados na tabela anterior serão usados ao longo desta obra, destacando-se os códigos **%f**, **%d**, **%s** e **%[]**.

A lista de argumentos é a indicação dos endereços das variáveis em uso por meio do operador de endereço **&**, que possibilita retornar o conteúdo da variável. Caso não seja usado o operador de endereço **&**, será retornado o endereço de memória em que se encontra a variável.

A função **printf()** possibilita a saída de informações no vídeo. A sintaxe dessa função é semelhante à sintaxe da função **scanf()**.

```
printf("expressão de controle", lista de argumentos);
```

A expressão de controle possui códigos de formatação para o tipo de dado a ser processado, precedidos pelo sinal de porcentagem **%**. Veja em seguida a tabela com os códigos de formatação para a função **printf()**.

Tabela 3.6 - Códigos de formatação para função printf()

Código	Função
%c	Permite a escrita de apenas um caractere.
%d	Permite a escrita de números inteiros decimais.
%e	Realiza-se a escrita de números em notação científica.
%f	É feita a escrita de números reais (ponto flutuante).
%g	Permite a escrita de %e ou %f no formato mais curto.
%o	Permite que números octais sejam escritos.
%s	Efetua-se a escrita de uma série de caracteres.
%u	Escreve-se um número decimal sem sinal.
%x	Permite a escrita de um número hexadecimal.
%n[]	Permite determinar entre colchetes quais caracteres podem ser ou não aceitos na entrada de uma sequência de caracteres, sendo "n" um valor opcional que determina o tamanho da sequência de caracteres.

Daqui para frente o processo de montagem de um programa fica mais fácil; basta que sejam seguidas as orientações anteriores.

Observação

Nesta obra as instruções da linguagem C são escritas em caracteres minúsculos (exigência obrigatória por parte dos compiladores de linguagem C), enquanto as variáveis são escritas sempre em caracteres maiúsculos.

Para que o programa desenvolvido seja executado, é necessário que você tenha instalado em seu computador um compilador de linguagem C. Existem diversas maneiras de conseguir um compilador, conforme orientações fornecidas no capítulo 2.

Em seguida escreva o programa anteriormente indicado com todo o cuidado para evitar erros de sintaxe no programa de edição de textos favorito, grave-o com o nome **adicao.c** no diretório (pasta) **estudo** e efetue a compilação com o comando **gpp adicao.c –o adiçao** no MS-Windows e **g++ adicao.c –o adicao** no Linux, depois realize a chamada do programa **adicao** de acordo com a particularidade de cada um dos sistemas operacionais.

Observação

A partir deste instante todas as vezes em que for solicitado escrever no editor de textos e gravar um programa, considerar-se-á, mesmo implicitamente, que o leitor vai executar os passos descritos. Desta forma, a indicação não será explicitamente mencionada.

Se houver algum erro de sintaxe, o compilador não efetua a compilação do programa. Neste caso, é necessário primeiramente corrigir o programa para depois realizar nova tentativa.

3.7.2 Programa Salário

Desenvolver em linguagem C um programa que calcule o salário líquido de um profissional que trabalhe por hora.

Para fazer esse programa, é necessário possuir alguns dados básicos, tais como valor da hora de trabalho, número de horas trabalhadas no mês e o percentual de desconto do INSS.

O programa em questão deve apresentar o valor do salário bruto, o valor descontado e o valor do salário líquido. Depois de escrever o programa no editor de textos, grave-o com o nome **salario.c**.

Algoritmo

1. Estabelecer a leitura da variável HT (horas trabalhadas no mês).
2. Estabelecer a leitura da variável VH (valor hora trabalhada).
3. Estabelecer a leitura da variável PD (percentual de desconto).
4. Calcular o salário bruto (SB), sendo este a multiplicação das variáveis HT e VH.
5. Calcular o total de desconto (TD) com base no valor de PD dividido por 100.
6. Calcular o salário líquido (SL), subtraindo o desconto do salário bruto.
7. Apresentar os valores dos salários bruto e líquido: SB, TD e SL.

Programa

```
/* Calculo de Salario */
#include <stdio.h>
int main(void)
{
  float HT, VH, PD, TD, SB, SL;
  scanf("%f", &HT);
  scanf("%f", &VH);
```

```
  scanf("%f", &PD);
  SB = HT * VH;
  TD = (PD/100) * SB;
  SL = SB - TD;
  printf("%f", SB);
  printf("%f", TD);
  printf("%f", SL);
  return 0;
}
```

Ao ser executado o programa anterior, ele apresenta o cursor piscando do lado esquerdo da tela. Note que essa ocorrência pode ocasionar uma sensação de não saber o que se deve exatamente fazer. Perceba que fica difícil saber o momento certo para informar os valores para o programa. Isso ocorre porque o programa não possui nenhuma indicação para quem está digitando os dados de entrada, podendo ser melhorado com mensagens que façam o programa "conversar" com o usuário, além de poder apresentar os valores de saída de uma forma melhor. Assim sendo, considere a seguinte versão do programa.

```
/* Calculo de Salario */
#include <stdio.h>
int main(void)
{
  float HT, VH, PD, TD, SB, SL;
  printf("Quantas horas de trabalho? "); scanf("%f", &HT);
  printf("Qual o valor da hora? "); scanf("%f", &VH);
  printf("Qual o percentual de desconto? "); scanf("%f", &PD);
  SB = HT * VH;
  TD = (PD/100) * SB;
  SL = SB - TD;
  printf("Salario Bruto ....: %f\n", SB);
  printf("Desconto .........: %f\n", TD);
  printf("Salario liquido ..: %f\n", SL);
  return 0;
}
```

A função **printf()**, além da expressão de controle e da lista de argumentos, pode conter um conjunto de códigos especiais com o símbolo \ barra invertida, com a finalidade de melhorar o efeito visual dos dados no vídeo.

```
printf("Augusto Manzano\nteste@teste.com.br");
```

Observe neste exemplo a utilização do código **\n** no meio da mensagem. O código informa à função **printf()** que a impressão do restante da linha após o código **\n** deve ser feita em uma nova linha, tendo como resultado:

```
Augusto Manzano
teste@teste.com.br
```

A seguir é exibida uma tabela contendo os códigos especiais de formatação de saída de dados em vídeo.

Tabela 3.7 - Códigos de formatação de saída

Caractere especial	Finalidade
\"	Apresenta o símbolo de aspas no ponto em que é indicado.
\?	Apresenta o símbolo de ponto de interrogação no ponto em que é indicado.
\\	Apresenta o símbolo de barra no ponto em que é indicado.
\'	Apresenta o símbolo de apóstrofo no ponto em que é indicado.
\0	Gera um caractere nulo (zero).
\a	Gera um sinal de áudio (beep) no alto-falante.
\b	Executa um retrocesso de espaço do ponto que é indicado.
\f	Gera um salto de página de formulário (uso de uma impressora).
\n	Gera uma nova linha a partir do ponto que é indicado.
\r	Gera um retorno de carro sem avanço de linha.
\t	Gera um espaço de tabulação do ponto que é indicado.
\v	Gera a execução da tabulação vertical.

O uso dos códigos especiais permite ao programador obter caracteres que não podem ser conseguidos diretamente do teclado para dentro do programa.

Faça as alterações propostas e grave o programa com o nome **salario2.c**, executando-o novamente. Observe a melhora que as pequenas alterações proporcionaram. Note que, apesar de ter melhorado, é necessário ainda melhorar a apresentação dos dados numéricos, definindo o número de inteiros e a precisão dos números com ponto flutuante. Observe o programa em seguida:

```
/* Calculo de Salario */
#include <stdio.h>
int main(void)
```

```
{
  float HT, VH, PD, TD, SB, SL;
  printf("Quantas horas de trabalho? "); scanf("%f", &HT);
  printf("Qual o valor da hora? "); scanf("%f", &VH);
  printf("Qual o percentual de desconto? "); scanf("%f", &PD);
  SB = HT * VH;
  TD = (PD/100) * SB;
  SL = SB - TD;
  printf("Salario Bruto ....: %7.2f\n", SB);
  printf("Desconto .........: %7.2f\n", TD);
  printf("Salario liquido ..: %7.2f\n", SL);
  return 0;
}
```

São colocados junto com o código da expressão de controle dois valores separados por ponto: **%7.2f**. O valor numérico 7 indica que será estabelecido um campo para a parte do valor inteiro com a capacidade de apresentar valores tabulados até 9999, pois o valor 2 depois do ponto indica que serão utilizadas apenas duas casas decimais para valores com ponto flutuante, utilizando três dos sete espaços separados. Após proceder à alteração, grave o programa com o nome **salario3.c**.

3.8 Recomendações do Padrão ISO/IEC C

O padrão ISO/IEC C estabelece que toda função escrita deve sempre retornar um valor (por ser este o conceito mais puro desse tipo de estrutura). Desta forma sugere-se que, ao escrever uma função do tipo **int**, seja colocada sempre no seu final (antes do símbolo fecha-chave) a instrução **return**. O capítulo 7 aborda com maior profundidade as funções.

O fato de não colocar a instrução de retorno não implica em erro, mas o padrão ISO/IEC C recomenda que o faça. Desta forma, compiladores de linguagem C que utilizam o formato ISO/IEC C, ao compilarem o programa, apresentam diversas mensagens de advertência, indicando as funções do programa que não estabelecem um retorno.

Um programa bem compilado (politicamente correto) é aquele em que o compilador no máximo apresenta apenas uma mensagem **"compiled successfully"**, **"built successfully"**, entre outras de operação bem-sucedida.

Há aqueles compiladores que, quando a operação é bem-sucedida, não apresentam nenhuma mensagem.

Para fazer um programa politicamente correto em linguagem C, é preciso seguir à risca as definições da norma ISO/IEC C. Observe em seguida um exemplo básico de um programa escrito de forma incorreta e de forma correta (de acordo com o modo ISO/IEC C).

Exemplo incorreto (apesar de poder ser compilado)

```
#include <stdio.h>
main()
{
  printf("Alo mundo");
}
```

Apesar de o programa estar escrito sintaticamente de forma correta, ao ser compilado (será realmente gerado seu executável), dependendo do compilador em uso será apresenta mensagens de advertência, tais como:

- Function should return a value;
- Missing return value;
- No type specified, default to int;
- Entre outras.

Exemplo correto (respeitando o padrão ISO)

```
#include <stdio.h>
int main(void)
{
  printf("Alo mundo");
  return 0;
}
```

Foi estabelecido o tipo da função **main()** como **int main()**. Desta forma, a função **main()** retorna para o programa chamador (no caso o próprio sistema operacional) um valor inteiro. É passado um parâmetro para a função **main()** como **void**, ou seja, nada é transferido para a função **main(void)**.

Ao final, após a conclusão do programa principal, foi colocada a instrução **return 0;** para forçar o retorno do valor zero para a função **main()**. Ao ser

compilado esse programa (que está escrito em padrão ISO), nenhuma mensagem de advertência será apresentada.

Observação

Para funções do tipo **int**, como é o caso da função **main(),** é ideal incluir na penúltima linha do código da função, antes de fechar a chave, a instrução **return 0**. Desta forma, evitar-se-á a apresentação de mensagens de advertência por parte de alguns compiladores no momento da compilação do programa.

Para funções do tipo **void**, deve-se utilizar apenas a instrução **return** sem nenhum parâmetro de retorno.

Exercícios

Desenvolva os algoritmos e a codificação em linguagem C dos seguintes programas. Não se esqueça de ir gravando cada programa. Como sugestão, você pode gravar o exercício "a" como EXERC03A, o exercício "b" como EXERC03B e assim por diante.

a. Ler quatro valores numéricos inteiros e apresentar o resultado dois a dois da adição e multiplicação entre os valores lidos, baseado-se na utilização do conceito de propriedade distributiva. Sugestão: se forem lidas as variáveis A, B, C e D, devem ser somados e multiplicados os valores de A com B, A com C e A com D; depois B com C, B com D e por último C com D. Note que para cada operação são utilizadas seis combinações. Assim sendo, devem ser realizadas doze operações de processamento, sendo seis para as adições e seis para as multiplicações.

b. Efetuar o cálculo da quantidade de litros de combustível gasta em uma viagem, utilizando um automóvel que faz 12 Km por litro. Para obter o cálculo, o usuário deve fornecer o tempo gasto na viagem e a velocidade média. Desta forma, será possível obter a distância percorrida com a fórmula DISTANCIA = TEMPO * VELOCIDADE. Tendo o valor da distância, basta calcular a quantidade de litros de combustível utilizada na viagem com a fórmula LITROS_USADOS = DISTANCIA / 12. O programa deve apresentar os valores da velocidade média, tempo gasto, a distância percorrida e a quantidade de litros utilizada na viagem. Sugestão: trabalhe com valores reais.

c. Ler uma temperatura em graus Celsius e apresentá-la convertida em graus Fahrenheit. A fórmula de conversão de temperatura a ser utilizada é F = (9 * C + 160) / 5, em que a variável F representa a temperatura em graus Fahrenheit e a variável C representa a temperatura em graus Celsius.

d. Ler uma temperatura em graus Fahrenheit e apresentá-la convertida em graus Celsius. A fórmula de conversão de temperatura a ser utilizada é C = ((F - 32) * 5) / 9, em que a variável F é a temperatura em graus Fahrenheit e a variável C é a temperatura em graus Celsius.

e. Calcular e apresentar o valor do volume de uma lata de óleo, utilizando a fórmula V = 3.14159 * R * R * A, em que as variáveis V, R e A representam, respectivamente, o volume, o raio e a altura.

f. Ler dois valores inteiros para as variáveis A e B, efetuar a troca dos valores de modo que a variável A passe a possuir o valor da variável B, e a variável B passe a possuir o valor da variável A. Apresentar os valores trocados.

Anotações

4

capítulo

A Tomada de Decisões

Objetivos

Este capítulo mostra os recursos relacionados ao processo de tomadas de decisão simples, decisão composta, operadores relacionais, operadores lógicos e arquivo de cabeçalho ISO 646.

São utilizados os comandos:

- *if*
- *else*

São utilizados os operadores relacionais:

- *igual a (==)*
- *diferente de (!=)*
- *maior que (>)*
- *menor que (<)*
- *maior ou igual a (>=)*
- *menor ou igual a (>=)*

São utilizados os operadores lógicos:

- *conjunção (and ou &&)*
- *disjunção (or ou ||)*
- *negação (not ou !)*

4.1 Condição e Decisão

Para que seja possível um programa de computador tomar decisões, é necessário primeiramente imputar uma condição. Não só os computadores agem segundo este critério. Os seres humanos também tomam decisões baseados em condições apresentadas a eles. Para tomar uma decisão, é necessário existir uma condição.

Imagine como exemplo um programa de computador que leia quatro notas bimestrais de um determinado aluno, calcule a média e apresente-a. Até aqui, muito simples, mas além de calcular e apresentar a média, o programa deve também informar se o aluno foi *aprovado* ou *reprovado* segundo a verificação de sua média (considere a média para aprovação quando ela for maior ou igual a cinco).

O programa deve verificar o valor da média (que será a condição, no caso, média maior ou igual a cinco) do aluno para então tomar uma decisão no sentido de apresentar apenas uma das mensagens indicadas: *Aprovado* ou *Reprovado*. Em hipótese nenhuma podem ser apresentadas ambas as mensagens.

Neste contexto de apresentar apenas uma das duas mensagens estabelecidas está uma condição (média maior ou igual a cinco) que implica na tomada de uma decisão e apresenta a mensagem mais adequada para o contexto avaliado. Se a condição de verificação da média (média maior ou igual a cinco) for verdadeira, deve ser apresentada a mensagem *Aprovado*; caso contrário, deve ser exibida a mensagem *Reprovado*.

4.2 Decisão Simples

Para solucionar o problema, é necessário trabalhar com uma nova instrução, denominada **if**, que tem por finalidade tomar uma decisão e efetuar um desvio no processamento do programa, dependendo, é claro, de a condição atribuída ser *Verdadeira* ou *Falsa*.

Sendo a condição *Verdadeira,* será executada a instrução que estiver escrita após a instrução **if**. Caso seja necessário executar mais de uma instrução para uma condição verdadeira, elas devem estar escritas dentro de um bloco. Lembre-se de que um bloco é definido com os símbolos de chaves "{" e "}".

Sendo a condição *Falsa*, executam-se as instruções que estejam após a instrução considerada verdadeira, ou as instruções após o bloco considerado verdadeiro. Desta forma a instrução **if** pode ser escrita:

```
if <(condição)>
  <instrução para condição verdadeira>;
<instrução para condição falsa ou após condição ser verdadeira>;
```

Caso exista a necessidade de processar mais de uma instrução para uma mesma condição verdadeira, elas devem estar inseridas em um bloco, como indicado em seguida.

```
if <(condição)>
  {
    <instrução1 para condição verdadeira>;
    <instrução2 para condição verdadeira>;
    <instrução3 para condição verdadeira>;
    <instruçãoN para condição verdadeira>;
  }
<instrução para condição falsa ou após condição ser verdadeira>;
```

Para exemplificar o uso prático da instrução **if**, considere em seguida os dois exemplos de problemas propostos, observando-os cuidadosamente e prestando bastante atenção em suas características operacionais.

4.2.1 Programa Adição de Números

Escrever um programa em linguagem C que faça a leitura de dois valores numéricos, efetuar a adição e apresentar o seu resultado caso o valor somado seja maior que 10.

Algoritmo

1. Conhecer dois valores incógnitos (estabelecer variáveis *A* e *B*).
2. Efetuar a soma dos valores incógnitos *A* e *B,* implicando o valor da soma na variável *X*.
3. Apresentar o valor da soma contido na variável *X*.

Programa

```
/* Adiciona Numeros */
#include <stdio.h>
int main(void)
{
  int A, B, X;
  printf("Informe um valor para a variavel A: "); scanf("%d", &A);
  printf("Informe um valor para a variavel B: "); scanf("%d", &B);
  X = A + B;
  if (X > 10)
    printf("\no valor da variavel X equivale: %d", X);
  return 0;
}
```

No programa anterior, após a definição dos tipos de variáveis, é solicitada a leitura dos valores para as variáveis *A* e *B*. Depois esses valores são implicados na variável *X*, a qual passa a possuir o resultado da adição dos dois valores.

Após calcular a soma e implicar o resultado na variável *X,* o programa verifica uma condição com a instrução **if** que permite imprimir o resultado da adição caso seja maior que 10. Não sendo, o programa é encerrado sem apresentar o referido resultado, uma vez que a condição é falsa.

A mensagem da condição **O valor da variável X equivale:** é precedida do símbolo especial **\n**. Lembre-se de que este símbolo faz com que a mensagem indicada após o seu uso seja escrita na próxima linha. Grave o programa anterior com o nome **adic_nr1.c**.

4.2.2 Programa Ordena

Escrever um programa em linguagem C que leia dois valores inteiros e independentemente da ordem em que foram entrados, eles devem ser impressos na ordem crescente, ou seja, se forem fornecidos 5 e 3, respectivamente, devem ser apresentados 3 e 5.

O programa proposto necessita verificar se um dos valores é maior que o outro. Se for constatado que esta condição é verdadeira, o programa deve efetuar a troca dos valores entre as duas variáveis, para então apresentar os valores ordenados.

Algoritmo

1. Conhecer dois valores inteiros e incógnitos (estabelecer variáveis *A* e *B*).
2. Verificar se o valor de *A* é maior que o valor de *B*.
   a. Se for verdadeiro, efetuar a troca (*) de valores entre as variáveis.
   b. Se for falso, pedir para executar o que está estabelecido no passo 3.
3. Apresentar os valores das duas variáveis.

(*) Para efetuar a troca, utilizar uma terceira variável, no caso *X,* por exemplo, de forma que *X* seja igual ao valor de *A*, liberando *A* para receber o valor de *B* e em seguida *B* estar livre para receber o valor que está em *X*, que anteriormente era o valor de *A*.

Programa

```
/* Ordena */
#include <stdio.h>
int main(void)
{
  int A, B, X;
  printf("Informe um valor para a variavel A: "); scanf("%d", &A);
  printf("Informe um valor para a variavel B: "); scanf("%d", &B);
  if (A > B)
    {
      X = A;
      A = B;
      B = X;
    }
  printf("\nos valores ordenados sao: %d e %d", A, B);
  return 0;
}
```

Note neste exemplo o uso das chaves após a utilização da instrução **if**. Isso é necessário, pois a troca de valores é conseguida com a execução de três operações, as quais devem acontecer quando a condição for *Verdadeira*. Sendo a condição *Falsa,* são apresentados os valores como entrados, uma vez que já estão em ordem.

Observe na linha **printf(**"\nos valores ordenados são: %d e %d", *A*, *B*)**;**, a colocação do código **%d** para indicar o local de posicionamento do valor da variável. Grave esse programa com o nome **ordena.c**.

4.3 Operadores Relacionais

No exemplo anterior foi utilizado o sinal de > (maior que) para verificar o estado da variável quanto ao seu valor lógico. Sendo assim, uma condição também pode ser verificada, como diferente de, igual a, menor que, maior ou igual que e menor ou igual que. As verificações são efetuadas com a utilização dos chamados operadores relacionais, conforme tabela seguinte:

Tabela 4.1 - Operadores relacionais

Símbolo	Significado
==	igual a
!=	diferente de
>	maior que
<	menor que
>=	maior ou igual que
<=	menor ou igual que

Observação

Ao utilizar o operador relacional **"=="** que representa o valor de igualdade de uma comparação entre dois elementos, deve-se tomar o cuidado para não confundi-lo com o sinal **"="**, que na linguagem C é utilizado como símbolo de atribuição. Este erro é comum para os estudantes que utilizam outros ambientes de programação como Turbo Pascal, Visual Basic, Delphi, Quick Basic. É preciso estar bem atento.

4.4 Decisão Composta

Já foi estudado como usar a instrução **if**. Agora você vai aprender a usar a instrução **if..else**. Sendo a condição *Verdadeira*, será executada a instrução que estiver posicionada entre as instruções **if** e **else**. Sendo a condição *Falsa,* será executada a instrução que estiver posicionada logo após a instrução **else**.

Caso seja necessário considerar mais de uma instrução para as condições *Verdadeira* ou *Falsa,* usam-se blocos delimitados por chaves. A instrução **if...else** possui a seguinte estrutura:

```
if <(condição)>
  <instruções para condição verdadeira>;
```

```
else
  <instruções para condição falsa>;
```

Caso exista mais de uma instrução verdadeira ou falsa para uma determinada condição, elas devem estar inseridas em um bloco, como indicado em seguida:

```
if <(condição)>
  {
    <instrução1 para condição verdadeira>;
    <instruçãoN para condição verdadeira>;
  }
else
  {
    <instrução1 para condição falsa>;
    <instruçãoN para condição falsa>;
  }
```

Para exemplificar o uso prático da instrução **if...else**, considere em seguida os dois exemplos de problemas propostos, observando-os cuidadosamente e prestando bastante atenção em suas características operacionais.

4.4.1 Programa Faixa Numérica

Escrever um programa em linguagem C que faça a leitura de dois valores numéricos e efetuar a adição. Caso o valor somado seja maior ou igual a 10, deve ser apresentado somando a ele mais 5. Caso o valor somado não seja maior ou igual a 10, deve ser apresentado subtraindo dele 7.

Algoritmo

1. Conhecer dois valores (variáveis *A* e *B*).
2. Efetuar a soma dos valores *A* e *B* e implicar o valor da soma em *X*.
3. Verificar se *X* é maior ou igual 10; caso sim, mostre *X*+5, senão mostre *X*-7.

Programa

```
/* Adiciona Numeros Versao 2 */
#include <stdio.h>
int main(void)
```

```
{
  int A, B, X;
  printf("Informe um valor para a variavel A: "); scanf("%d", &A);
  printf("Informe um valor para a variavel B: "); scanf("%d", &B);
  X = A + B;
  printf("\nO resultado equivale a: ");
  if (X >= 10)
    printf("%d", X + 5);
  else
    printf("%d", X - 7);
  return 0;
}
```

Após a definição dos tipos de variáveis é solicitada a leitura dos valores para as variáveis *A* e *B*, depois esses valores são implicados na variável *X*, a qual possui o resultado da adição dos dois valores.

Em seguida é apresentada a mensagem **O resultado equivale a:**, que não posiciona o cursor na próxima linha. Desta forma, qualquer que seja o resultado avaliado pela condição, ele será apresentado do lado direito da mensagem. Depois é executada a linha de programa com a instrução **if** (*X* >= 10), a qual questiona a condição do programa.

Se a condição for verdadeira, é apresentado o valor de *X* somado com mais 5. Não sendo a condição verdadeira, o programa apresenta o valor de *X* subtraindo 7. Grave o programa com o nome **adic_nr2.c**.

4.4.2 Programa Média Escolar

Escrever um programa em linguagem C que faça a leitura de quatro valores numéricos referentes a quatro notas bimestrais de um aluno, efetuar o cálculo da média e apresentá-la junto com uma das seguintes mensagens: caso o valor da média seja maior ou igual a 5, apresentar "aluno aprovado"; caso contrário, apresentar "aluno reprovado".

Algoritmo

1. Entrar quatro valores para as notas (estabelecer variáveis *N1*, *N2*, *N3* e *N4*).
2. Calcular a média: *MD* = (*N1* + *N2* + *N3* + *N4*) / 4.

3. Verificar se o valor da média é maior ou igual a 5. Caso seja a média verdadeira, apresentar a mensagem "Aluno aprovado"; senão, apresentar a mensagem "Aluno reprovado".

4. Apresentar também junto com cada mensagem o valor da média obtida.

Programa

```
/* Calcula Media Escolar */
#include <stdio.h>
int main(void)
{
  float N1, N2, N3, N4, MD;
  printf("\nEntre a Nota 1 ...: "); scanf("%f", &N1);
  printf("Entre a Nota 2 ...: "); scanf("%f", &N2);
  printf("Entre a Nota 3 ...: "); scanf("%f", &N3);
  printf("Entre a Nota 4 ...: "); scanf("%f", &N4);
  MD = (N1 + N2 + N3 + N4) / 4;
  if (MD >= 5)
    printf("Aluno aprovado com media = ");
  else
    printf("Aluno reprovado com media = ");
  printf("%5.2f\n", MD);
  return 0;
}
```

Observe que, se a condição da média for verdadeira, é apresentada a mensagem "Aluno aprovado". Não sendo a condição verdadeira, o programa apresenta a mensagem "Aluno reprovado". Grave o programa com o nome **calcmed.c**.

4.5 Operadores Lógicos

Existem ocasiões em que é necessário trabalhar com o relacionamento de duas ou mais condições ao mesmo tempo na mesma instrução **if**, efetuando testes múltiplos. Para estes casos é necessário trabalhar com os operadores lógicos, também conhecidos como operadores booleanos.

Já foi comentado anteriormente que a linguagem C não usa valores lógicos, tais como *Verdadeiro* ou *Falso*, porque esses dois valores não existem. Em uma expressão condicional ou no uso de operadores lógicos, é considerado

como valor *Verdadeiro* o resultado da expressão que for não zero (será considerado para efeito de avaliação lógica o valor numérico um: 1). Sendo o resultado da expressão zero, será considerado como valor *Falso*.

Os operadores lógicos comumente usados na atividade de programação de computadores são quatro: E (and), OU Inclusivo (or), OU Exclusivo (xor) e NÃO (not). Em alguns casos, os operadores lógicos evitam a utilização de muitas instruções **if** encadeadas. Dos quatro operadores lógicos a linguagem C faz uso direto de três, sendo E, OU Inclusivo e NÃO. O operador OU Exclusivo na linguagem C existe para operações em nível binário, as quais estão fora do escopo deste trabalho. Assim sendo, o operador lógico OU Exclusivo será tratado nesta obra, mas não na espera binária.

4.5.1 Operador Lógico de Conjunção

O operador **&&** é utilizado quando dois ou mais relacionamentos lógicos de uma determinada condição necessitam ser verdadeiros. Em seguida é apresentada a tabela-verdade para esse tipo de operador:

Tabela 4.2 - Tabela-verdade para operador &&

Condição 1	Condição 2	Resultado Lógico
0 (Falsa)	0 (Falsa)	0 (Falso)
1 (Verdadeira)	0 (Falsa)	0 (Falso)
0 (Falsa)	1 (Verdadeira)	0 (Falso)
1 (Verdadeira)	1 (Verdadeira)	1 (Verdadeiro)

O operador **&&** faz com que somente seja executada uma determinada operação se todas as condições mencionadas forem simultaneamente verdadeiras, gerando um resultado lógico verdadeiro. A seguir, é apresentado um exemplo com a utilização do operador lógico **&&**.

Desenvolver em linguagem C um programa que aceite valores numéricos inteiros entre 0 (zero) e 9 (nove). Se o valor estiver dentro da faixa, o programa deve apresentar a mensagem "valor válido". Caso o valor esteja fora da faixa, o programa deve apresentar a mensagem "valor inválido".

Algoritmo

1. Entrar um valor numérico inteiro (usar a variável NUMERO).

2. Verificar por meio de condição se o valor fornecido está na faixa de 0 (zero) a 9 (nove).

3. Caso o valor fornecido esteja dentro da faixa, apresentar a mensagem "valor válido"; caso contrário, apresentar a mensagem "valor inválido".

Programa

```
/* Operador AND (E) - && */
#include <stdio.h>
int main(void)
{
  int NUMERO;
  printf("\nEntre um numero entre 0 e 9: ");
  scanf("%d", &NUMERO);
  if (NUMERO >= 0 && NUMERO <= 9)
    printf("Valor valido");
  else
    printf("Valor invalido");
  return 0;
}
```

Observe que qualquer valor fornecido dentro da faixa estabelecida pela linha de comando **if** (NUMERO >= 0 **&&** NUMERO <= 9) apresenta a mensagem "valor válido". Qualquer valor acima de 9 ou abaixo de 0 resulta na apresentação da mensagem "valor inválido". Grave o programa com o nome **testand.c**.

4.5.2 Operador Lógico de Disjunção Inclusiva

O operador lógico **||** (sentido de **OU** - **or** em inglês) é utilizado quando pelo menos um dos relacionamentos lógicos (quando houver mais de um relacionamento) de uma condição necessita ser verdadeiro. Em seguida é apresentada a tabela-verdade para esse tipo de operador:

Tabela 4.3 - Tabela-verdade para operador ||

Condição 1	Condição 2	Resultado Lógico
0 (Falsa)	0 (Falsa)	0 (Falso)
1 (Verdadeira)	0 (Falsa)	1 (Verdadeiro)
0 (Falsa)	1 (Verdadeira)	1 (Verdadeiro)
1 (Verdadeira)	1 (Verdadeira)	1 (Verdadeiro)

O operador **||** faz com que seja executada uma determinada operação, se pelo menos uma das condições mencionadas gerar um resultado lógico verdadeiro. A seguir é apresentado um exemplo com o operador lógico **||**.

Escrever um programa em linguagem C que leia um valor numérico inteiro referente aos códigos 1, 2 ou 3. Qualquer outro valor deve apresentar a mensagem "código inválido". Se o valor estiver correto, apresentar o valor do código escrito por extenso".

Algoritmo

1. Entrar um valor numérico inteiro (usar a variável CODIGO).
2. Verificar se o código fornecido é igual a 1, igual a 2 ou igual a 3. Caso não seja, apresentar a mensagem "código inválido".
3. Ao ser verificado o código e constatado que é um valor válido, o programa deve verificar cada código em separado para determinar seu valor por extenso.

Programa

```
/* Operador OR (OU) - || */
#include <stdio.h>
int main(void)
{
  int CODIGO;
  printf("\nEntre o codigo de acesso: ");
  scanf("%d", &CODIGO);
  if (CODIGO == 1 || CODIGO == 2 || CODIGO == 3)
    {
      if (CODIGO == 1)
        printf("um\n");
      if (CODIGO == 2)
        printf("dois\n");
      if (CODIGO == 3)
        printf("tres\n");
    }
  else
    printf("codigo invalido\n");
  return 0;
}
```

Qualquer valor fornecido dentro da faixa estabelecida pela linha de comando **if** (CODIGO == 1 **||** CODIGO == 2 **||** CODIGO == 3) é em seguida verificado em separado para determinar a apresentação do seu valor escrito por extenso. Qualquer valor que não seja 1, 2 ou 3 resulta na apresentação da mensagem "código inválido". Grave o programa com o nome **testor.c**.

4.5.3 Operador Lógico de Negação

O operador lógico **!** (sentido de **NÃO** - **not** em inglês) é utilizado quando se necessita estabelecer que uma determinada condição deve não ser verdadeira ou deve não ser falsa. O operador **!** se caracteriza por inverter o estado lógico de uma condição.

Em seguida é apresentada a tabela-verdade para esse tipo de operador:

Tabela 4.4 - Tabela-verdade para operador !

Condição	Resultado Lógico
1 (Verdadeira)	0 (Falso)
0 (Falsa)	1 (Verdadeira)

O operador **!** faz com que seja executada uma determinada operação, invertendo o resultado lógico da condição. A seguir, é apresentado um exemplo com a utilização do operador lógico **!**.

*Escrever um programa em linguagem C que leia um valor numérico inteiro que **não seja** negativo. Qualquer outro valor deve apresentar a mensagem "valor inválido". Se o valor estiver correto, apresentar a mensagem "valor válido, você informou" e colocar junto com a mensagem o valor fornecido.*

Algoritmo

1. Entrar um valor numérico inteiro (usar a variável VALOR).
2. Verificar se o valor fornecido **não é** maior ou igual a 0. Se o valor não for maior ou igual a 0 (no caso, valores negativos), apresentar a mensagem "valor inválido".
3. Se o valor fornecido for positivo, apresentar a mensagem "valor válido, você informou" e apresentar também o valor válido fornecido.

Programa

```
/* Operador NOT (NAO) - ! */
#include <stdio.h>
int main(void)
{
  int VALOR;
  printf("\nEntre um valor inteiro positivo: ");
  scanf("%d", &VALOR);
  if (!(VALOR >= 0))
    printf("valor invalido\n");
  else
    printf("valor valido, voce informou %d\n", VALOR);
  return 0;
}
```

Observe atentamente o operador lógico de negação **!** e perceba que para utilizá-lo torna-se necessário abrir um conjunto de parênteses. Para negar a condição (VALOR >= 0), foi necessário utilizar a sintaxe **(!** <(condição)>**)**. Grave o programa com o nome **testnot.c**.

4.5.4 Operador Lógico de Disjunção Exclusiva

O operador lógico de disjunção exclusiva (normalmente mencionado em inglês como **xor**) existente na linguagem C é representado pelo símbolo **"^"** (circunflexo), sendo utilizado para efetuar operações em nível binário. Não sendo esta a intenção deste trabalho, o tema será exposto a partir de uma solução lógica/matemática muito peculiar. O operador lógico de disjunção exclusiva é utilizado quando um dos relacionamentos lógicos (quando houver mais de um relacionamento) de uma condição necessita ser verdadeiro, enquanto o outro relacionamento lógico deve ser falso. Em seguida é apresentada a tabela-verdade para esse tipo de operador:

Tabela 4.5 - Tabela-verdade para operador ^

Condição 1	Condição 2	Resultado Lógico
0 (Falsa)	0 (Falsa)	0 (Falso)
1 (Verdadeira)	0 (Falsa)	1 (Verdadeiro)
0 (Falsa)	1 (Verdadeira)	1 (Verdadeiro)
1 (Verdadeira)	1 (Verdadeira)	1 (Falso)

O operador lógico de conjunção exclusiva faz com que seja executada uma determinada operação, se apenas uma das condições mencionadas gerar um resultado lógico verdadeiro. Para conseguir esse efeito no mesmo nível operacional que ocorre com o uso dos operadores lógicos **&&** (e), **||** (ou) e **!** (não), que não são operadores de execução em nível binário, deve-se usar a expressão lógica **((cond1 && (!cond2)) || ((!cond1) && cond2))** quando se desejar avaliar a expressão **(cond1 xor cond2)**. A seguir, é apresentado um exemplo com a utilização do operador lógico de disjunção exclusiva.

Escrever um programa em linguagem C que faz a solicitação do sexo de duas pessoas que pretendem formar um par para participar de uma quadrilha em uma festa junina. Os administradores da festa determinaram que somente serão aceitos pares heterogêneos (formados por pessoas de sexos diferentes). Não serão aceitos casais formados por pessoas do mesmo sexo. Para atender este requisito, o programa deve, após a entrada do sexo dos participantes, verificar se os sexos informados formam realmente um par, e neste caso apresentar uma mensagem informando sucesso em relação a essa possibilidade; caso contrário, o programa deve indicar a impossibilidade de formação de par.

Algoritmo

1. Informar o sexo do primeiro participante.
2. Informar o sexo do segundo participante.

Verificar se os sexos informados podem formar um par de dança. Caso seja possível, apresentar mensagem sobre o sucesso da composição; caso contrário, informar que a combinação não é possível.

Programa

```
/* Operador XOR (OU Exclusivo) - (S1 == 1 xor S2 == 1) */
#include <stdio.h>
int main(void)
{
  int S1, S2;
  printf("Entre sexo do 1o. - [1] = M / [0] = F: ");
  scanf("%d", &S1);
  printf("Entre sexo do 2o. - [1] = M / [0] = F: ");
  scanf("%d", &S2);
  if ((S1 == 1 && (!S2 == 1)) || ((!S1 == 1) && S2 == 1))
    printf("O 1o. pode dancar com o 2o.\n");
  else
    printf("O 1o. nao pode dancar com o 2o.\n");
  return 0;
}
```

No exemplo apresentado, se forem fornecidos valores de sexos diferentes na entrada para os participantes (1 para masculino e 0 para feminino ou 0 para feminino e 1 para masculino), o programa confirma a possibilidade de formação de par. Se forem fornecidos sexos iguais para as entradas solicitadas, o programa indica a não possibilidade de formação de par. Grave o programa com o nome **testxor.c**.

4.5.5 Precedência dos Operadores Lógicos

Os operadores lógicos **and**, **or** e **xor** permitem mais de uma condição para a tomada de uma única decisão. Já o operador lógico **not** tem por finalidade a negação do estado lógico de uma condição.

Para usar adequadamente os operadores lógicos em expressões lógicas, é necessário levar em consideração a ordem de precedência desses operadores. A tabela de precedência de operadores lógicos exibe a ordem hierárquica de execução dos operadores lógicos que podem ser utilizados em uma expressão lógica.

Tabela 4.6 - Precedência de operadores lógicos

Operador	Operação	Precedência
!	Negação	1
&&	Conjunção	2
||	Disjunção inclusiva	3

A tabela de precedência dos operadores lógicos indica a prioridade de execução em uma expressão lógica. O operador **!** é o que possui maior nível de prioridade, portanto deve ser operacionalizado primeiro. Num segundo momento tem-se o operador lógico **&&** que possui nível médio de prioridade e por último os operadores **||** com baixo nível de prioridade.

Por exemplo, a expressão lógica **(A == B) && ! (A > 5)** deve ser avaliada a partir de **! (A > 5)** e, somente depois de saber seu resultado, pode ser realizada a avaliação do restante da expressão lógica.

4.5.6 Programa Triângulo

Escrever um programa em linguagem C que leia três valores referentes às medidas dos lados de um triângulo, considerando lados como A, B e C. Verificar se os lados fornecidos formam realmente um triângulo. Se for esta condição verdadeira, deve ser indicado qual tipo de triângulo foi formado: isósceles, escaleno ou equilátero.

Algoritmo

Para estabelecer o algoritmo, é necessário em primeiro lugar saber o que realmente é um triângulo. Se você não souber o que é um triângulo, consequentemente não conseguirá resolver o problema. Triângulo é uma forma geométrica (polígono) composta por três lados, sendo cada lado menor que a soma dos outros dois lados. Perceba que é uma regra (uma condição) e deve ser considerada. É um triângulo quando $A<B+C$, quando $B<A+C$ e quando $C<A+B$.

Tendo certeza de que os valores informados para os três lados formam um triângulo, são analisados os valores para estabelecer qual tipo de triângulo será formado: isósceles, escaleno ou equilátero.

Um triângulo é isósceles quando possui dois lados iguais e um diferente, sendo $A==B$ ou $A==C$ ou $B==C$; é escaleno quando possui todos os lados diferentes, sendo $A!=B$ e $B!=C$; é equilátero quando possui todos os lados iguais, sendo $A==B$ e $B==C$.

1. Ler três valores para os lados de um triângulo: *A*, *B* e *C*.
2. Verificar se cada lado é menor que a soma dos outros dois lados:
    a. Se sim, saber se $A==B$ e se $B==C$, sendo verdade o triângulo é equilátero;
    b. Se não, verificar se $A==B$ ou se $A==C$ ou se $B==C$, sendo verdade o triângulo é isósceles; caso contrário, o triângulo será escaleno.
3. Caso os lados fornecidos não caracterizem um triângulo, avisar a ocorrência.

Programa

```
/* Checa Triangulo */
#include <stdio.h>
int main(void)
{
  float A, B, C;
  printf("\nInforme o lado A: "); scanf("%f", &A);
  printf("Informe o lado B: "); scanf("%f", &B);
  printf("Informe o lado C: "); scanf("%f", &C);
  printf("\n");
  if (A<B+C && B<A+C && C<A+B)
    if (A==B && B==C)
      printf("Triangulo Equilatero\n");
    else
      if (A==B || A==C || C==B)
        printf("Triangulo Isosceles\n");
      else
        printf("Triangulo Escaleno\n");
  else
    printf("Os valores fornecidos nao formam um triangulo\n");
  return 0;
}
```

Com a utilização dos operadores lógicos o exemplo demonstra a capacidade de um programa definir se determinados valores fornecidos formam realmente um triângulo, e se a condição for verdadeira, o programa indica o tipo de triângulo formado. Se forem fornecidos, respectivamente, os valores 8, 2 e 3 para os lados A, B e C, será apresentada a mensagem **"Os valores fornecidos não formam um triângulo"**, pois a primeira condição da instrução **if** (*A*<*B*+*C* **&&** *B*<*A*+*C* **&&** *C*<*A*+*B*) resulta o valor lógico falso, uma vez que 8 é maior que a soma de 2 com 3.

Caso sejam fornecidos os valores 8, 8 e 2, a primeira instrução **if** será verdadeira, indicando que os lados fornecidos formam um triângulo. Apesar de o lado A ser igual ao lado B, mas este não ser igual ao lado C, o triângulo formado não é do tipo equilátero. A segunda instrução **if** tem seu resultado lógico como falso, sobrando a verificação da condição para a terceira instrução **if** que indica o triângulo como sendo do tipo isósceles, pois o lado A é igual ao lado B. Grave este programa com o nome **operador.c**.

4.6 Arquivo de Cabeçalho ISO 646

À medida que uma linguagem de programação evolui, ela recebe novos recursos, e a linguagem de programação C não é diferente. Assim sendo, a linguagem de programação C "ganhou" o arquivo de cabeçalho ISO 646 que permite usar palavras nos operadores lógicos. No lugar de **&&** usar **and**, no lugar de **||** pode-se usar **or** e no lugar de **!** agora é possível usar **not**.

Esses novos recursos somente podem ser usados se o arquivo de cabeçalho **iso646.h** for mencionado no início do código de programa. No sentido de demonstrar esse recurso, considere o programa seguinte.

```
/* Arquivo de Cabecalho – iso646.h */
#include <stdio.h>
#include <iso646.h>
int main(void)
{
  int NUMERO;
  printf("\nInforme um numero entre 0 e 9: ");
  scanf("%d", &NUMERO);
  if (NUMERO >= 0 and NUMERO <= 9)
    printf("Valor na faixa de 0 a 9");
  else
    printf("Valor fora da faixa de 0 a 9");
  return 0;
}
```

Qualquer valor fornecido dentro da faixa estabelecida pela linha de comando **if** (NUMERO >= 0 **and** NUMERO <= 9) apresenta a mensagem "valor na faixa de 0 a 9". Qualquer valor acima de 9 ou abaixo de 0 resulta na apresentação da mensagem "valor fora da faixa de 0 a 9". Grave o programa com o nome **testiso646.c**.

4.7 Divisibilidade

Divisibilidade é a qualidade do que é divisível. Neste contexto duas definições devem ser conhecidas e entendidas pelos programadores de computador, sendo os conceitos de múltiplos e divisores de números naturais. Entende-se por número natural um valor numérico que seja inteiro e positivo.

Múltiplos são os resultados obtidos da multiplicação de dois números naturais, enquanto divisores são os números que dividem outros números com o objetivo

de gerar um resultado de divisão exato, ou seja, obter resto de divisão sempre zero. Quando o resto de uma divisão de números naturais é igual a zero, tem-se divisibilidade, ou seja, resultado de divisão exata. Perceba que valores do tipo **float** não entram nesta abordagem, apenas os do tipo **int**.

Pelo fato de todo número natural ser múltiplo de si mesmo, uma forma de trabalhar esses valores é fazer as operações de cálculo com o uso de divisores. Em uma divisão existem alguns termos que precisam ser conhecidos: dividendo (valor que será dividido), divisor (valor que divide o dividendo), quociente (resultado da divisão do dividendo pelo divisor) e resto (valor que sobra da operação de divisão). A figura 4.1 mostra o processo de divisão de dois números naturais.

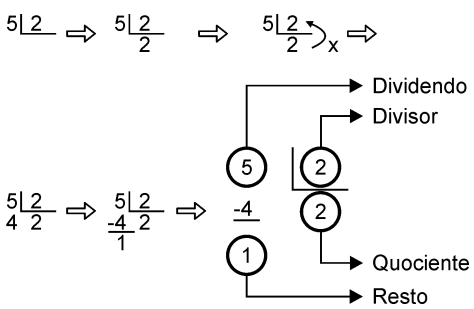

Figura 4.1 - Exemplo de divisão de dois números naturais.

De acordo com a figura 4.1, fica fácil deduzir como obter a divisibilidade de um dividendo (valor **5**) por um divisor (valor **2**). Considere a expressão matemática **Resto = Dividendo – Divisor · Quociente** para obter a divisibilidade, na qual as variáveis **Dividendo**, **Divisor** e **Quociente** serão substituídas por valores inteiros positivos. Assim sendo, considere os valores de dividendo **5** e divisor **2** na expressão matemática **Resto = 5 – 2 · 2**. Basta realizar primeiramente o cálculo da multiplicação **2 · 2** para obter o valor **4** que será subtraído do valor **5** e resultará o valor **1** para a variável **Resto**, ou seja, **Resto** é diferente de zero. Neste caso não ocorreu a divisibilidade do valor **5** pelo valor **2** porque o valor **2** não é múltiplo do valor **5**.

A expressão matemática **Resto = Dividendo – Divisor · Quociente** é obtida a partir da equação matemática: $r = a - n \left\lfloor \frac{a}{n} \right\rfloor$

Em que a variável **r** representa o resto da divisão, a variável **a** representa o dividendo e a variável **n** o divisor (KNUTH, 1972). Observe a indicação do cálculo do quociente com o uso da função *parte inteira inferior* que obtém o resultado inteiro da divisão do dividendo **a** pelo divisor **n**. A função *parte*

inteira tem o objetivo de retornar a parte inteira (o expoente) de um número real, desconsiderando as casas decimais (a mantissa) desse número.

A função *parte inteira* pode ser representada de duas formas: *parte inteira inferior*, também denominada *chão* ou *piso*, e *parte inteira superior*, também chamada *teto*. Na forma *inferior* retorna o expoente do valor, desconsiderando toda a mantissa desse valor. Já na forma *superior* retorna o arredondamento do valor para o valor inteiro mais próximo. As funções *parte inteira inferior* e *parte inteira superior* são graficamente representadas como:

$\lceil x \rceil$ Parte inteira superior

$\lfloor x \rfloor$ Parte inteira inferior

Do ponto de vista computacional, a expressão matemática para obter a divisibilidade dos valores deve ser convertida em uma expressão aritmética. Assim sendo, a expressão matemática **Resto = Dividendo – Divisor · Quociente** será escrita na linguagem C como **Resto = Dividendo – Divisor * (Dividendo / Divisor)**, levando-se em consideração que as variáveis **Resto**, **Dividendo** e **Divisor** são do tipo **int**. Neste caso o operador aritmético de divisão "**/**" gerará automaticamente um quociente do tipo **int**. Considerando um programa que verifique se o valor **5** é divisível pelo valor **2**, seria escrito em linguagem C como **Resto = 5 – 2 * (5 / 2);**.

A expressão aritmética **Resto = Dividendo – Divisor * (Dividendo / Divisor)** é uma operação algorítmica genérica, que pode ser facilmente adaptada para qualquer linguagem de programação. Caso deseje fazer uma simplificação desse tipo de operação em linguagem C, pode-se fazer uso do operador de resto de divisão (operador de módulo de divisão "**%**"). Neste caso a expressão **Resto = 5 – 2 * (5 / 2);**, pode ser escrita como **Resto = 5 % 2;**.

A título de demonstração e uso do processo de divisibilidade em linguagem C, considere como exemplo o problema computacional seguinte: *desenvolver um programa de computador que leia um valor numérico inteiro e faça a apresentação desse valor caso seja divisível por 4 e 5. Não sendo divisível por 4 e 5, o programa deve apresentar a mensagem "Valor não é divisível por 4 e 5".* Para resolver este problema, é necessário, além de usar o operador lógico **&&**, verificar se a condição do valor lido é ou não divisível por 4 e 5.

A seguir são apresentadas duas soluções para o problema proposto, sendo a primeira com o uso da forma genérica de solução do problema e a segunda com o uso do operador de módulo de divisão.

Algoritmo

1. Ler um valor numérico inteiro qualquer (variável N).
2. Calcular o resto da divisão de N por 4, usar a variável R4.
3. Calcular o resto da divisão de N por 5, usar a variável R5.
4. Verificar se as variáveis R4 e R5 possuem o valor de resto igual a zero. Se sim, apresentar o valor da variável N; se não, apresentar a mensagem "Valor não é divisível por 4 e 5".

Programa: Exemplo 1

```
/* Divisibilidade - Exemplo 1 */
#include <stdio.h>
int main(void)
{
  int N, R4, R5;
  printf("\nEntre um valor inteiro natural: ");
  scanf("%d", &N);
  R4 = N - 4 * (N / 4);
  R5 = N - 5 * (N / 5);
  if (R4 == 0 && R5 == 0)
    printf("%d\n", N);
  else
    printf("Valor nao e divisivel por 4 e 5\n");
  return 0;
}
```

Observe neste primeiro exemplo o uso das expressões aritméticas de cunho genérico **R4 = N - 4 * (N / 4)** e **R5 = N - 5 * (N / 5)** para a obtenção do valor da divisibilidade, do resto da divisão do valor da variável **N** pelos valores dos divisores **4** e **5** respectivamente. A instrução de tomada de decisão faz uso da condição **(R4 == 0 && R5 == 0)** que será verdadeira se ambas as condições forem verdadeiras. Grave o programa com o nome **div_1.c**.

Programa: Exemplo 2

```
/* Divisibilidade - Exemplo 2 */
#include <stdio.h>
int main(void)
{
  int N, R4, R5;
  printf("\nEntre um valor inteiro natural: ");
  scanf("%d", &N);
  R4 = N % 4;
  R5 = N % 5;
  if (R4 == 0 && R5 == 0)
    printf("%d\n", N);
  else
    printf("Valor nao e divisivel por 4 e 5\n");
  return 0;
}
```

Observe neste segundo exemplo o uso do operador de módulo de divisão **R4 = N % 4** e **R5 = N % 5** para a obtenção do valor da divisibilidade, do resto da divisão do valor da variável **N** pelos valores dos divisores **4** e **5** respectivamente. Grave o programa com o nome **div_2.c**.

Exercícios

Desenvolva algoritmos, diagramas de blocos e codificação em linguagem C dos seguintes problemas:

a. Ler três valores inteiros (representados pelas variáveis A, B e C). O programa deve apresentar os valores dispostos em ordem crescente. Sugestão: utilize a propriedade distributiva e a troca de valores para ordenar os valores, pois a saída somente pode ser formada pela sequência de variáveis A, B e C, nesta ordem (em hipótese alguma utilize outra combinação de saída).

b. Efetuar a leitura de três valores (variáveis A, B e C) para calcular a equação de segundo grau. Lembre-se de que somente será uma equação de segundo grau se o valor da variável A for diferente de zero. Outro detalhe é que somente existirá o cálculo das raízes se o valor de delta for diferente de zero. É necessário considerar também o fato de o valor de delta ser igual a

zero. Se isso ocorrer, existirá apenas o cálculo de uma raiz. Sugestão: utilize a biblioteca math.h (#include <math.h>) antes da função "main()".

c. Ler quatro valores referentes a quatro notas escolares de um aluno e mostrar uma mensagem informando que o aluno foi aprovado se o valor da média for maior ou igual a 5. Se o aluno não foi aprovado, apresentar uma mensagem informando esta condição. Junto com as mensagens apresentar o valor da média do aluno.

d. Ler quatro valores referentes a quatro notas escolares de um aluno e apresentar uma mensagem informando que o aluno foi aprovado se o valor da média escolar for maior ou igual a 7. Se o valor da média for menor que 7, solicitar a nota de exame, somar com o valor da média e obter nova média. Se a nova média for maior ou igual a 5, apresentar uma mensagem informando que o aluno foi aprovado em exame. Se o aluno não foi aprovado, mostrar uma mensagem informando esta condição. Mostrar junto com as mensagens o valor da média do aluno.

e. Ler dois valores numéricos inteiros ou reais e exibir a diferença do maior para o menor.

5

capítulo

Laços de Repetição

Objetivos

Este capítulo aborda os recursos relacionados à execução de laços (malhas) de repetição interativos e iterativos. São utilizados laços interativos e iterativos do tipo condicional pré-teste, condicional pós-teste e laços iterativos com variável de controle.

Utilizam-se os comandos:

- *while*
- *do...while*

E também a função:

- *for()*

5.1 Laços de Repetição

Existem situações em que é necessário repetir o trecho de um programa um determinado número de vezes, o que pode ser conseguido de duas formas. Na primeira será escrito o mesmo trecho tantas vezes quanto necessário, bastante trabalhoso; na segunda forma podem ser utilizados laços de repetição, conhecidos também como loopings ou malhas de repetição.

Supondo que há necessidade de construir um programa que execute um determinado trecho (bloco) de instruções cinco vezes, certamente ele será feito com a repetição manual dos trechos desejados, uma vez que o conhecimento até então adquirido induz a esta solução.

A vantagem em utilizar laços de repetição (loopings) é que o programa passa a ser menor e seu processamento aumentado sem alterar o tamanho do código de programação. Desta forma é possível determinar repetições com números variados de vezes.

5.2 Laço Condicional Pré-Teste

Essa estrutura realiza um teste lógico no início do laço de repetição, verificando se é permitido executar o trecho de instruções subordinado a ele. Existem para esse tipo de laço duas possibilidades de ação, uma sendo o laço executado quando a condição é verdadeira, e outra quando a condição é falsa. A linguagem C faz uso do laço que efetua a ação enquanto a condição é verdadeira. Para realizar uma ação de laço pré-teste falso, basta usar o operador lógico de negação antes da condição.

A estrutura **while** tem o seu funcionamento controlado por condição, portanto pode executar um determinado conjunto de instruções enquanto a condição verificada permanecer *Verdadeira*. No momento em que a condição se torna *Falsa,* o processamento da rotina é desviado para fora do laço de repetição.

Caso seja a condição *Falsa* logo no início do laço de repetição, as instruções dele são ignoradas. Caso seja necessário executar mais de uma instrução para uma condição verdadeira dentro de um laço, elas devem estar em um bloco com símbolos de chaves. A instrução **while** deve ser escrita:

```
while [(not] <(condição)> [)]
  <instrução>;
```

Caso venha a existir mais de uma instrução para ser executada dentro do laço, deve estar inserida em um bloco delimitado pelos símbolos **{** e **}**:

```
while [(not] <(condição)> [)]
  {
    <instrução 1 enquanto condição ser válida>;
    <instrução 2 enquanto condição ser válida>;
    <instrução 3 enquanto condição ser válida>;
    <instrução N enquanto condição ser válida>;
  }
```

O uso do operador lógico **not** é opcional e deverá ser usado quando se desejar simular um laço do tipo pré-teste falso.

A seguir são apresentados dois exemplos de aprendizagem de programas que usam laço com condição no início.

5.2.1 Programa Cálculo

Elaborar um programa em linguagem C que solicite cinco vezes dois números para o cálculo e apresentação de uma adição.

Algoritmo

Para atender a essa necessidade, é necessário definir um contador de vezes para controlar a execução do programa, e cada vez que for executado o trecho desejado do programa, esse contador deve ser incrementado de mais 1. Observe os detalhes para a solução do problema:

1. Criar uma variável para servir como contador com valor inicial 1.
2. Enquanto o valor do contador for menor ou igual a 5, processar os passos 3, 4 e 5.
3. Ler os dois valores.
4. Efetuar o cálculo, implicando o resultado em *R*.
5. Apresentar o valor calculado na variável *R*.
6. Acrescentar ao contador 1.
7. Quando o contador for maior que 5, encerrar o processamento.

Programa

```
/* Looping do tipo While Versao 1A */
#include <stdio.h>
int main(void)
{
  int A, B, R, I;
  I = 1;
  while (I <= 5)
    {
      printf("\n\nEntre um valor para A: "); scanf("%d", &A);
      printf("Entre um valor para B: "); scanf("%d", &B);
      R = A + B;
      printf("\nO resultado corresponde a: %d", R);
      I = I + 1;
    }
  return 0;
}
```

Além de utilizar as variáveis de ação *A*, *B* e *R*, foi necessário criar uma quarta variável de controle, no caso *I* para controlar a contagem do número de vezes que o trecho de programa deve ser executado.

Assim que o programa é inicializado, a variável contador é atribuída com o valor 1 (*I* = 1). Em seguida a instrução **while** (*I* <= 5) verifica se a condição estabelecida é verdadeira, pois o valor da variável *I* que neste momento é 1, é realmente menor que 5 e enquanto for, deve processar o laço. Desta forma, é iniciada a execução da rotina de instruções do laço de repetição delimitado pela instrução **while**.

Depois de solicitar os valores, processar a operação de adição e exibir o resultado, o programa encontra a linha com a instrução *I* = *I* + 1, indicando o acréscimo de 1 à variável contador. Observe que a variável *I* possui neste momento o valor 1 e somado a mais 1 ela passa a ter o valor 2, ou seja, *I* = *I* + 1, sendo assim *I* = 1 + 1 que resulta *I* = 2.

Após a variável *I* estar carregada com o valor 2, o processamento do programa volta para a instrução **while** (*I* <= 5), que verifica a condição da variável. Sendo a condição *Verdadeira*, executa novamente a mesma rotina de instruções. Quando o processamento do programa chegar na instrução *I* = *I* + 1, fará com que a variável *I* passe a possuir o valor 3. O programa processa novamente a rotina de instruções, passando o valor de *I* para 4, que será verificado, e sendo menor que 5, será executada mais uma vez a mesma rotina de instruções.

A variável *I* passa a possuir o valor 5. A instrução **while** (*I* <= 5) verifica o valor da variável *I* que é 5 com a condição *I* <= 5. Veja que 5 não é menor que 5, mas é igual, e sendo esta condição verdadeira, a rotina deve ser executada mais uma vez.

Em seguida, o valor da variável *I* passa a ser 6, que resulta para a instrução **while** uma condição falsa e por conseguinte desvia o processamento para a primeira instrução encontrada após o fechamento do bloco com o símbolo de chave direita "}", no caso para o fim do programa. Grave o programa com o nome **loop_w1a.c**.

No exemplo apresentado, a rotina de cálculo é executada cinco vezes. Caso queira executar essa rotina um número maior ou menor de vezes, basta trocar o valor 5 da instrução **while** (*I* <= 5) pelo valor desejado. Por exemplo, executar a mesma rotina por 15 vezes ficaria **while** (*I* <= 15).

O uso de contadores na linguagem C pode ser representado de outra forma. No programa anterior, é encontrada a linha *I* = *I* + 1 que poderia ter sido escrita como *I* ++. Veja a seguir um exemplo com esta forma de escrita, faça a alteração proposta e grave o programa com o nome **loop_w2a.c**, rode-o em seguida e observe o seu funcionamento.

```
/* Looping do tipo While Versao 2A */
#include <stdio.h>
int main(void)
{
  int A, B, R, I;
  I = 1;
  while (I <= 5)
    {
      printf("\n\nEntre um valor para A: "); scanf("%d", &A);
      printf("Entre um valor para B: "); scanf("%d", &B);
      R = A + B;
      printf("\nO resultado corresponde a: %d", R);
      I ++;
    }
  return 0;
}
```

Imagine ainda uma outra situação na qual o usuário deseja executar a rotina do programa várias vezes, mas ele não sabe quantas vezes ao certo deve executar o trecho de programa. Neste caso, não seria conveniente manter um contador para controlar o laço. Seria melhor que o programa fizesse ao usuário uma pergunta, solicitando se ele deseja ou não continuar executando o programa. Em seguida, o exemplo da nova situação:

Algoritmo

1. Criar uma variável para ser utilizada como resposta.
2. Enquanto a resposta for sim, executar os passos 3, 4 e 5.
3. Ler os valores.
4. Efetuar o cálculo, implicando o resultado em *R*.
5. Apresentar o valor calculado na variável *R*.
6. Quando a resposta for diferente de "s" (sim), encerrar o processamento.

Programa

```
/* Looping do tipo While Versao 3A */
#include <stdio.h>
int main(void)
{
  int A, B, R, RESP;
  RESP = 1;
  while (RESP == 1)
    {
      printf("\n\nEntre um valor para A: "); scanf("%d", &A);
      printf("Entre um valor para B: "); scanf("%d", &B);
      R = A + B;
      printf("\nO resultado corresponde a: %d", R);
      printf("\n\nDeseja continuar?");
      printf("\nTecle [1] para SIM / [2] para NAO: ");
      scanf("%d", &RESP);
    }
  return 0;
}
```

No programa anterior, o contador foi substituído pela variável *RESP*, que enquanto tiver o seu valor igual a **1**, executa o bloco de instruções subordinado a **while**. Neste caso, o número de vezes que a rotina vai se repetir será controlado pelo usuário e depende da informação fornecida para a variável *RESP*. Se o usuário responder **1**, o programa continua; se responder **2**, é encerrado. Grave o programa com o nome **loop_w3a.c**.

Observação

Veja que o programa anterior usa dois códigos numéricos para controlar a resposta do usuário (**1** para sim e **2** para não). O uso de *strings* para representar as ações **SIM** e **NÃO** é um detalhe delicado a ser tratado com a linguagem C. O capítulo 6 trata de *strings* detalhadamente.

A codificação em linguagem C permite simplificar a forma escrita do valor de uma variável. No programa anterior, é iniciada a variável *RESP* como do tipo **int** e antes de iniciar o laço, ela é atribuída com o valor **1** para dar início ao processamento do laço de repetição. Veja a seguir uma sugestão de simplificação do código do programa. Faça a alteração proposta e grave a nova versão com o nome **loop_w4a.c**.

```
/* Looping do tipo While Versao 4A */
#include <stdio.h>
int main(void)
{
  int A, B, R, RESP = 1;
  while (RESP == 1)
    {
      printf("\n\nEntre um valor para A: "); scanf("%d", &A);
      printf("Entre um valor para B: "); scanf("%d", &B);
      R = A + B;
      printf("\nO resultado corresponde a: %d", R);
      printf("\n\nDeseja continuar?");
      printf("\nTecle [1] para SIM / [2] para NAO: ");
      scanf("%d", &RESP);
    }
  return 0;
}
```

Após a inicialização da variável com o seu tipo, está sendo atribuído simultaneamente o valor da variável.

5.2.2 Programa Fatorial

Elaborar em linguagem C um programa que calcule a fatorial de um número qualquer. Supondo que o número seja 5, o programa deve apresentar o resultado 5! (fatorial de 5). Desta forma, temos que 5! = 5 . 4 . 3 . 2 . 1 ou 5! = 1 . 2 . 3 . 4 . 5, equivalente a 120.

Algoritmo

Fatorial é o produto dos números naturais desde 1 até o inteiro n. Sendo assim, o cálculo de uma fatorial é conseguido pela multiplicação sucessiva do número de termos. No caso do cálculo de uma fatorial de número 5, este é o número de termos a ser utilizado. O programa deve executar as multiplicações sucessivamente e acumulá-las a fim de possuir o valor 120 após cinco passos. O número de passos deve ser controlado por um contador. Veja em seguida:

1. Inicializar as variáveis *FATORIAL* e *CONTADOR* com 1.
2. Solicitar o valor de um número para calcular a sua fatorial.
3. Multiplicar sucessivamente a variável *FATORIAL* pela variável *CONTADOR*.
4. Incrementar 1 à variável *CONTADOR,* efetuando o controle até o limite definido no passo 2.
5. Apresentar ao final o valor obtido.

Pelo fato de ser necessário calcular a fatorial de um número qualquer (N!), implica que o contador deve variar de 1 a N, por este motivo deve ser a variável *CONTADOR* inicializada com valor 1. Como a variável *FATORIAL* possui o resultado do cálculo da fatorial pretendida (por meio de uma multiplicação sucessiva), ela deve ser inicializada com valor 1. Se for inicializada com zero, não há resultado final, pois qualquer valor multiplicado por zero resulta zero.

Observe dentro do laço a indicação de dois contadores. O primeiro funcionando como um acumulador, pois terá no final o resultado da fatorial, e o segundo sendo utilizado para controlar a execução do laço e ser a base para o cálculo do acumulador.

Logo no início do programa, as variáveis *CONTADOR* e *FATORIAL* são igualadas ao valor 1. Na primeira passagem dentro do laço, a variável *FATORIAL* é implicada pelo seu valor atual, no caso 1, multiplicado pelo valor da variável *CONTADOR* também 1 que resulta 1. Em seguida a variável *CONTADOR* é incrementada de 1, tendo agora o valor 2. Como 2 é menor ou igual a *N*, ocorre um novo cálculo. Desta vez a variável *FATORIAL* que possui o valor 1 é multiplicada pela variável *CONTADOR* que possui o valor 2, resultando 2 para *FATORIAL*. Então, a variável *CONTADOR* é incrementada de 1, tendo agora o valor 3.

São executados os outros cálculos até que a condição se torne falsa e seja apresentado o valor da fatorial do número definido. Considerando a variável *N* com valor 5 e a variável *CONTADOR* com valor 5, a variável *FATORIAL* estará com o valor 120. Neste ponto, o laço é executado mais uma vez, tornado o valor de CONTADOR igual a 6, e encerrando o laço, apresenta o valor da variável *FATORIAL*.

Programa

```
/* Looping do tipo While para Fatorial */
#include <stdio.h>
int main(void)
{
  int CONTADOR, N;
  long FATORIAL = 1;
  CONTADOR = 1;
  printf("\nPrograma Fatorial\n");
  printf("\nFatorial de que numero: "); scanf("%d", &N);
  while (CONTADOR <= N)
    {
      FATORIAL *= CONTADOR;
      CONTADOR ++;
    }
  printf("\nFatorial de %d equivale a %d\n\n", N, FATORIAL);
  return 0;
}
```

Digite o programa anterior, gravando-o com o nome **fator_a.c**.

5.3 Laço Condicional Pós-Teste

Essa estrutura realiza um teste lógico no final do laço de repetição, verificando se é permitido executar o trecho de instruções subordinado a ele até certa condição ser satisfeita. Existem para esse tipo de laço duas possibilidades de ação, uma sendo o laço executado quando a condição é verdadeira e outra quando a condição é falsa. A linguagem C faz uso do laço que efetua a ação até que a condição seja verdadeira. Para realizar uma ação de laço pós-teste falso, basta usar o operador lógico de negação antes da condição.

A estrutura **do...while** tem seu funcionamento controlado por condição, portanto executará determinado conjunto de instruções até que a condição enquanto seja *Verdadeira*. No momento em que a condição se torna *Verdadeira,* o processamento da rotina é desviado para fora do laço de repetição. Uma característica desse laço é que essa estrutura faz um teste lógico no final do laço de repetição, permitindo que a ação subordinada ao laço seja executada no mínimo uma vez. A instrução **do...while** deve ser escrita:

```
do
  {
    <repita instrução 1 enquanto condição válida>;
    <repita instrução 2 enquanto condição válida>;
    <repita instrução 3 enquanto condição válida>;
    <repita instrução N enquanto condição válida>;
  }
while [(not) <(condição)> ()];
```

O uso do operador lógico **not** é opcional e deverá ser usado quando se desejar simular um laço do tipo pós-teste falso.

Uma característica de uso do laço **do...while** é o fato de ele ser uma instrução de laço pós-teste verdadeiro. Em muitas outras linguagens de programação o laço pós-teste é normalmente executado com a condição falsa, o que difere a linguagem C de outras linguagens de programação, e exige de programadores habituados às outras linguagens um grau de atenção no uso dessa estrutura de laço. Muitos programadores, para manterem uma estrutura de laço pós-teste compatibilizada entre a linguagem C e outras linguagens, costumam usar o operador **!** (not) na condição.

5.3.1 Programa Cálculo 2

Para exemplificar a utilização dessa estrutura de laço, será considerado como exemplo o mesmo programa anterior, que pede a leitura de dois valores, efetua a adição e apresenta o resultado cinco vezes.

Algoritmo

1. Criar uma variável para servir como contador com valor inicial 1.
2. Ler os valores.
3. Efetuar o cálculo, implicando o resultado em *R*.
4. Apresentar o valor calculado na variável *R*.
5. Acrescentar ao contador 1.
6. Repetir os passos 2, 3, 4 e 5 enquanto o contador for menor ou igual a 5.

Programa

```
/* Looping do tipo Do...While Versao 1B */
#include <stdio.h>
int main(void)
{
  int A, B, R, I = 1;
  do
    {
      printf("\n\nEntre um valor para A: "); scanf("%d", &A);
      printf("Entre um valor para B: "); scanf("%d", &B);
      R = A + B;
      printf("\nO resultado corresponde a: %d", R);
      I ++;
    }
  while (I <= 5);
  return 0;
}
```

Digite e grave o programa anterior com o nome **loop_d1b.c**. Assim que o programa for executado, a variável contador é inicializada com o valor 1 (*I* = 1). Em seguida a instrução **do** indica que todo trecho de instruções situado até a instrução **while** (I <= 5) será executado enquanto a variável I for menor ou igual a 5.

A seguir está um exemplo em que não se utiliza o contador como forma de controle do número de vezes de uma determinada rotina em uma estrutura de repetição. Considere que o usuário encerra o processamento segundo a sua vontade.

Algoritmo

1. Criar uma variável para ser utilizada como resposta.
2. Ler os valores.
3. Efetuar o cálculo, implicando o resultado em *R*.
4. Apresentar o valor calculado na variável *R*.
5. Perguntar ao usuário se deseja continuar executando o programa.
6. Repetir os passos 2, 3, 4 e 5 enquanto a resposta do usuário for igual a sim.

Programa

```
/* Looping do tipo Do...While Versao 2B */
#include <stdio.h>
int main(void)
{
  int A, B, R, RESP = 1;
  do
    {
      printf("\n\nEntre um valor para A: "); scanf("%d", &A);
      printf("Entre um valor para B: "); scanf("%d", &B);
      R = A + B;
      printf("\nO resultado corresponde a: %d", R);
      printf("\n\nDeseja continuar?");
      printf("\nTecle [1] para SIM / [2] para NAO: ");
      scanf("%d", &RESP);
    }
  while (RESP == 1);
  return 0;
}
```

Digite o programa em questão, gravando-o com o nome **loop_d2b.c**.

5.3.2 Programa Fatorial 2

Utilizando a instrução **do...while**, é exibida uma segunda versão do programa para cálculo da fatorial de um número qualquer, o qual deve ser gravado com o nome **fator_b.c**. Considere o problema:

Elaborar em linguagem C um programa que calcule a fatorial de um número qualquer.

Algoritmo

1. Inicializar as variáveis *FATORIAL* e *CONTADOR* com 1.
2. Solicitar o valor de um número para calcular a sua fatorial.
3. Multiplicar sucessivamente a variável *FATORIAL* pela variável *CONTADOR*.
4. Incrementar 1 à variável *CONTADOR,* realizando o controle enquanto o limite for menor ou igual ao valor definido no passo 2.
5. Apresentar ao final o valor obtido.

Programa

```
/* Looping do tipo Do...While para Fatorial */
#include <stdio.h>
int main(void)
{
  int CONTADOR, N;
  long FATORIAL = 1;
  CONTADOR = 1;
  printf("\nPrograma Fatorial\n");
  printf("\nFatorial de que numero: "); scanf("%d", &N);
  do
    {
      FATORIAL *= CONTADOR;
      CONTADOR ++;
    }
  while (CONTADOR <= N);
  printf("\nFatorial de %d equivale a %d\n\n", N, FATORIAL);
  return 0;
}
```

5.4 Laço com Variável de Controle

Foram apresentadas anteriormente duas formas de elaborar laços de repetição. Com as técnicas mostradas é possível elaborar rotinas que executem um laço, um determinado número de vezes, com a utilização de um contador ou mesmo de uma variável que aguarde a resposta do usuário. Independentemente da forma de tratamento, ela é denominada variável de controle.

Existe outra forma que facilita o uso de contadores finitos sem usar as estruturas de laços anteriormente apresentadas. Os laços de repetição com while e do...while passam a ser utilizados quando não se conhece de antemão o número de vezes que uma determinada sequência de instruções deve ser executada. Os laços de repetição que possuem um número finito de execuções podem ser processados por meio do laço for.

Esse laço de repetição tem o funcionamento controlado por uma variável contador, que pode ser crescente ou decrescente, tendo como sintaxe:

```
for (<início>; <fim>; <incremento>)
  <instruções>;
```

Caso exista mais de uma instrução dentro do laço, elas devem estar inseridas em um bloco, delimitado pelos símbolos de chaves.

```
for (<início>; <fim>; <incremento>)
  {
    <instruções1>;
    <instruções2>;
    <instruçõesN>;
  }
```

em que:

<início> - é uma instrução de atribuição com o valor inicial do laço.

<fim> - é uma instrução de condição com o valor final do laço.

<incremento> - expressão com o incremento do laço.

5.4.1 Programa Contagem Crescente

Elaborar em linguagem C um programa que escreva na tela os valores 1, 2, 3, 4, 5, 6, 7, 8, 9 e 10.

Algoritmo

1. Definir para a variável I a faixa de frequência de contagem de 1 até 10.
2. À medida que ocorrer cada contagem, o valor de I deve ser apresentado na tela.

Programa

```
/* Looping crescente de 1 em 1 */
#include <stdio.h>
int main(void)
{
  int I;
  for (I = 1; I <= 10; I ++)
    printf("I = %2d\n", I);
  return 0;
}
```

O primeiro argumento da função for() inicializa a variável contador com 1, o segundo argumento verifica se a variável atingiu o valor limite (I <= 10) e o terceiro se encarrega de executar o incremento de 1 em 1 à variável em uso. Grave o programa com o nome for1.c.

5.4.2 Programa Contagem Decrescente

Elaborar em linguagem C um programa que escreva na tela os valores 10, 9, 8, 7, 6, 5, 4, 3, 2 e 1.

Algoritmo

1. Definir para a variável I a faixa de frequência de contagem de 10 até 1.
2. À medida que ocorrer cada contagem, o valor de I deve ser apresentado na tela.

Programa

```
/* Looping decrescente de 1 em 1 */
#include <stdio.h>
int main(void)
{
  int I;
  for (I = 10; I >= 1; I --)
    printf("I = %2d\n", I);
  return 0;
}
```

O primeiro argumento da função **for()** inicializa a variável contador com 10, o segundo verifica se a variável atingiu o valor (I >= 1) limite e o terceiro se encarrega de executar o decremento de 1 em 1 à variável em uso. Grave o programa com o nome **for2.c**.

5.4.3 Programa Crescente 2

Elaborar em linguagem C um programa que escreva na tela os valores 1, 3, 5, 7 e 9.

Algoritmo

1. Definir para a variável I a faixa de frequência de contagem de 1 até 10.
2. Estabelecer um salto de contagem de 2 em 2, iniciando o valor a partir de 1.
3. À medida que ocorrer cada contagem, o valor de I deve ser apresentado na tela.

Programa

```
/* Looping crescente de 2 em 2 */
#include <stdio.h>
int main(void)
{
  int I;
  for (I = 1; I <= 10; I += 2)
    printf("I = %2d\n", I);
  return 0;
}
```

O primeiro argumento da função **for()** inicializa a variável contador com 1, o segundo verifica se a variável atingiu o valor limite e o terceiro se encarrega de executar o incremento de 2 em 2 à variável em uso, ou seja, *I* += 2 equivale a *I* = *I* + 2. Grave o programa com o nome **for3.c**.

5.4.4 Programa Crescente 3

Os exemplos anteriores mostraram as formas mais simples de utilização da função **for()** para construir laços de repetição finitos. Essa estrutura possibilita uma flexibilidade bastante grande em várias operações de processamento com a linguagem C.

As expressões utilizadas em um laço **for** podem ser mais de três, desde que todas estejam separadas por vírgula, que funciona como um operador que significa "faça isto e isto" e será sempre avaliada da esquerda para a direita.

Elaborar em linguagem C um programa que apresente na tela os valores: 2, 5, 8, 11, 14.

Algoritmo

1. Definir para a variável A o valor inicial 1.
2. Definir para a variável B o valor inicial 1.
3. Executar o laço até que a soma de A com B atinja o valor 15.
4. A variável A deve variar de dois em dois e a variável B deve variar de um em um.

Programa

```
/* Dois contadores */
#include <stdio.h>
int main(void)
{
  int A, B, R;
  for (A = 1, B = 1; A + B <= 15; A += 2, B ++)
    {
      R = A + B;
      printf("Resultado = %3d\n", R);
    }
  return 0;
}
```

A variável A começa com 1 e avança de 2 em 2, a variável B começa com 1 e avança de 1 em 1. Ambas as variáveis são acrescidas de seus respectivos incrementos até que a soma delas chegue ao máximo a 15. Grave o programa com o nome **for4.c**.

5.4.5 Programa Cálculo 2

Elaborar em linguagem C um programa que leia dois valores para as variáveis A e B. O programa deve somar os dois valores e apresentar o resultado, repetindo a sequência cinco vezes.

Algoritmo

1. Definir um contador que varia de 1 a 5.
2. Ler os valores.
3. Efetuar o cálculo, implicando o resultado em R.
4. Apresentar o valor calculado na variável R.
5. Repetir os passos 2, 3 e 4 até que o contador seja encerrado.

Programa

```
/* Looping do tipo For 1C */
#include <stdio.h>
int main(void)
{
  int A, B, R, I;
  for(I = 1; I <= 5; I ++)
    {
      printf("\n\nEntre um valor para A: "); scanf("%d", &A);
      printf("Entre um valor para B: "); scanf("%d", &B);


       R = A + B;
      printf("\nO resultado corresponde a: %d", R);
    }
  return 0;
}
```

Digite o programa anterior e grave-o com o nome **loop_f1c.c**.

Quando executado o programa, o conjunto de instruções situado abaixo da instrução **for** será executado cinco vezes, pois a variável I (variável de controle) inicializada com valor 1 é incrementada com mais 1 a cada vez que o processamento passa pela linha da instrução **for**.

Essa estrutura de repetição pode ser utilizada todas as vezes que for necessário repetir trechos finitos, ou seja, quando se conhecem o valor inicial e o valor final de uma variável contador.

5.4.6 Programa Fatorial 3

Utilizando a função (instrução) ***for()****, escrever um programa que apresente o resultado da fatorial de um número qualquer lido pelo teclado.*

Algoritmo

1. Inicializar as variáveis *FATORIAL* e *CONTADOR* com 1.
2. Solicitar o valor de um número para calcular a sua fatorial.
3. Multiplicar sucessivamente a variável *FATORIAL* pela variável *CONTADOR*.
4. Incrementar 1 à variável *CONTADOR* efetuando o controle até o limite definido no passo 2.
5. Apresentar ao final o valor obtido.

Programa

```
/* Looping do tipo For para Fatorial */
#include <stdio.h>
int main(void)
{
  int CONTADOR, N;
  long FATORIAL = 1;
  printf("\nPrograma Fatorial\n");
  printf("\nFatorial de que numero: "); scanf("%d", &N);
  for(CONTADOR = 1; CONTADOR <= N; CONTADOR ++)
    FATORIAL *= CONTADOR;
  printf("\nFatorial de %d equivale a %d\n\n", N, FATORIAL);
  return 0;
}
```

Note a utilização da função **for()** para estabelecer o número de repetições do laço que controla o cálculo do valor da fatorial. Ao final grave o programa com o nome **fator_c.c**.

Exercícios

Desenvolva os respectivos programas em linguagem C para os problemas indicados em seguida. Usar na resolução as estruturas de laços de repetição **while**, **do...while** e **for**:

a. Apresentar todos os valores numéricos inteiros ímpares situados na faixa de 0 a 20. Para verificar se o número é ímpar, na malha verificar a lógica dessa condição com a instrução **if**, perguntando se o número é ímpar; sendo, mostre-o, não sendo, passe para o próximo passo. Sugestão: a variável utilizada para a contagem deve obrigatoriamente ser inicializada com o valor 0 (zero). Utilize o operador % para obter o resultado do resto de uma divisão de valores inteiros e assim verificar se o número é par ou ímpar (lembre-se de que um valor é ímpar quando, dividido por dois, o resto é diferente de zero, ou seja, o resto é um valor).

b. Apresentar o somatório obtido dos cem primeiros números inteiros (1+2+3+4+5+6+7+...+97+98+99+100).

c. Apresentar os resultados de uma tabuada para um número qualquer a ser fornecido pelo usuário. Sugestão: procure apresentar o resultado da tabuada em seu formato-padrão matemático.

d. Apresentar todos os números divisíveis por 4 que sejam menores que 200. Sugestão: a variável utilizada para a contagem deve obrigatoriamente ser inicializada com o valor 1 (um). Utilize o operador % (resto de divisão) para verificar se o número é divisível por quatro (lembre-se de que um valor é divisível por outro valor quando o resto da divisão for igual a zero).

e. Apresentar os quadrados dos números inteiros de 15 a 180. Sugestão: lembre-se de que a função "pow()" somente pode ser utilizada para valores reais, não sendo aplicada para esta situação.

f. Apresentar as potências de 3, variando de 0 a 9. Deve ser considerado que qualquer número elevado a zero é 1, e elevado a 1 é ele próprio. Sugestão: resolva este problema sem utilizar a função "pow()"; trabalhe apenas com o conceito de laço para apresentar os valores calculados.

g. Escreva um programa que apresente a série de Fibonacci até o décimo quinto termo. A série de Fibonacci é formada pela sequência 1, 1, 2, 3, 5, 8, 13, 21, 34... etc.

Anotações

6

capítulo

Tabelas em Memória

Objetivos

Este capítulo apresenta uma técnica de programação que permite trabalhar com o agrupamento de vários dados em uma mesma variável. Vale salientar que esse agrupamento ocorre obedecendo sempre ao mesmo tipo de dado, a menos que se trabalhe com a estrutura de dados registro (estudada no final deste capítulo). Ele aborda matrizes de uma e duas dimensões, diferença entre índice e elemento, classificação de elementos, pesquisa de elementos, operações com strings de dados e como determinar a operação de resto entre um dividendo e um divisor.

Utiliza o comando:

- *struct*

E também as funções:

- *fflush()*
- *fgets()*
- *puts()*
- *stdin*
- *strcmp()*
- *strcpy()*

6.1 Estrutura de Dado Matricial

Uma estrutura de dado matricial pode ser um dado homogêneo que pode receber diversos nomes de identificação, como variáveis indexadas, variáveis compostas, variáveis subscritas, arranjos, vetores, matrizes, tabelas em memória ou *arrays*. No livro essa estrutura recebe a denominação de matriz.

As matrizes (tabelas em memória) são dados que podem ser "construídos" à medida que forem necessários, pois não é sempre que os tipos básicos (real, inteiro, caractere e lógico) e/ou variáveis simples são suficientes para representar a estrutura de dados utilizada em um programa.

6.2 Matrizes de Uma Dimensão ou Vetores

Essas estruturas, em particular, também são denominadas por alguns profissionais de matrizes unidimensionais. Sua utilização mais comum está vinculada à criação de tabelas e tem uma única variável dimensionada com um determinado tamanho. A dimensão de uma matriz apresenta constantes inteiras e positivas. Os nomes dados às matrizes seguem as mesmas regras de nomes de variáveis simples.

Uma matriz de uma dimensão ou vetor é representada por seu nome, tamanho (dimensão) entre colchetes e seu tipo, tendo a seguinte sintaxe:

```
tipo MATRIZ[dimensão];
```

em que:

<tipo> - o tipo de dado a ser guardado na matriz;

<matriz> - o nome atribuído à matriz;

<dimensão> - o tamanho da matriz em número de elementos.

Uma variável somente pode conter um valor por vez. No caso das matrizes, elas podem armazenar mais de um valor por vez, pois são dimensionadas exatamente para este fim. A manipulação dos elementos de uma matriz ocorre de forma individualizada, pois não é possível usar todos os elementos do conjunto ao mesmo tempo.

Para ter uma ideia de como utilizar matrizes em uma determinada situação, são apresentados em seguida alguns exemplos desse critério de programação.

6.2.1 Programa Média Geral Simples

Elaborar em linguagem C um programa que calcule e apresente o valor da média geral de uma turma de oito alunos. Deve ser a média geral das médias de cada aluno obtida durante o ano letivo.

Algoritmo

É necessário somar todas as médias e dividi-las por 8. A tabela seguinte apresenta o número de alunos, suas notas bimestrais e respectivas médias anuais. É da média de cada aluno que se calcula a média da turma.

Aluno	Nota1	Nota2	Nota3	Nota4	Média
1	4.0	6.0	5.0	3.0	4.5
2	6.0	7.0	5.0	8.0	6.5
3	9.0	8.0	9.0	6.0	8.0
4	3.0	5.0	4.0	2.0	3.5
5	4.0	6.0	6.0	8.0	6.0
6	7.0	7.0	7.0	7.0	7.0
7	8.0	7.0	6.0	5.0	6.5
8	6.0	7.0	2.0	9.0	6.0

Basta escrever um programa para calcular as oito médias de cada aluno. Para representar a média do primeiro aluno, utiliza-se a variável MD1, para o segundo MD2 e assim por diante. Sendo desse modo:

MD1 = 4.5
MD2 = 6.5
MD3 = 8.0
MD4 = 3.5
MD5 = 6.0
MD6 = 7.0
MD7 = 6.5
MD8 = 6.0

Programa

Com o conhecimento adquirido até o momento, elabora-se um programa que leia cada nota, a soma delas e a divisão do valor da soma por 8, obtendo-se a média, conforme exemplo apresentado em seguida:

```
/* Calculo de Media Escolar */
#include <stdio.h>
int main(void)
{
  float MD1, MD2, MD3, MD4, MD5, MD6, MD7, MD8;
  float SOMA, MEDIA;
  SOMA = 0;
  scanf("%f %f %f %f %f %f %f %f",
    &MD1, &MD2, &MD3, &MD4, &MD5, &MD6, &MD7, &MD8);
  SOMA = MD1 + MD2 + MD3 + MD4 + MD5 + MD6 + MD7 + MD8;
  MEDIA = SOMA / 8;
  printf("%6.2f", MEDIA);
  return 0;
}
```

Para receber a média foram utilizadas oito variáveis. Com a técnica de matrizes pode ser utilizada apenas uma variável com a capacidade de armazenar oito valores. Digite o programa e grave-o com o nome **media1.c**.

No caso deste exemplo, uma única variável indexada (a matriz) contém todos os valores das médias dos oito alunos, representados da seguinte forma:

MD[0] = 4.5
MD[1] = 6.5
MD[2] = 8.0
MD[3] = 3.5
MD[4] = 6.0
MD[5] = 7.0
MD[6] = 6.5
MD[7] = 6.0

O nome é um só; o que muda é o valor de *índice* indicado dentro dos colchetes, o qual representa o endereço em que o *elemento* está armazenado, ou seja, a nota do aluno. Para uma tabela de oito elementos os índices são numerados em linguagem C de 0 até 7.

Tanto a entrada como a saída de dados manipuladas com uma matriz são processadas passo a passo, um elemento por vez. Esses processos são executados com o auxílio de um laço de repetição, normalmente do tipo **for**.

6.2.2 Programa Média Geral

Com base na proposta anterior em relação ao programa de notas de alunos, segue o programa que fará em primeiro lugar a leitura da média de cada aluno, o cálculo da média da sala e em seguida a apresentação da média geral.

Algoritmo

1. Determinar um laço finito com oito repetições.
2. Entrar uma nota a cada contagem do laço.
3. Ao final calcular a média.
4. Mostrar o resultado obtido.

Programa

```
/* Calculo de Media : 8 alunos */
#include <stdio.h>
int main(void)
{
  float MD[8];
  float MEDIA, SOMA = 0;
  int I;
  printf("\nCalculo de media escolar\n\n");
  for (I = 0; I <= 7; I ++)
    {
      printf("Informe a %da. nota: ", I + 1); scanf("%f", &MD[I]);
      SOMA += MD[I];
    }
  MEDIA = SOMA / 8;
  printf("\nA media do grupo equivale a: %6.2f\n\n", MEDIA);
  return 0;
}
```

Digite o programa gravando-o com o nome **media2.c**. Ele faz referência à área de declaração de variáveis, a matriz *MD*[8], em que é definido o número de elementos, no caso oito, iniciando com índice 0 até o 7. Por este motivo é que a instrução **printf(**"Informe a %da. nota: ", **I+1);** indica a apresentação da variável *I* acrescida de 1. Quando o valor do índice for 0, será entrada a primeira nota; quando for 1, será a segunda nota e assim por diante.

6.2.3 Programa Índice

Desenvolver um programa que leia dez elementos de uma matriz A do tipo vetor. Construir uma matriz B de mesmo tipo, observando a seguinte lei de formação: se o valor do índice for par, o valor deve ser multiplicado por 5; sendo ímpar, deve ser somado com 5. Ao final, mostrar os conteúdos das duas matrizes.

Este exemplo demonstra como fazer o tratamento da condição do índice.

Algoritmo

1. Iniciar o contador de índice, variável *I* como 0 (zero) em um contador até 10.
2. Ler os dez valores um a um.
3. Verificar se o índice é par; se sim, multiplica por 5; se não, soma 5. Criar a matriz B.
4. Apresentar os conteúdos das duas matrizes.

Programa

```
/* Checa indice: par ou impar */
#include <stdio.h>
int main(void)
{
  int A[10], B[10];
  int I;
  printf("\n\nCalculo com checagem do indice da matriz\n\n");

  /* Entrada de dados */
```

```
  for (I = 0; I <= 9; I ++)
    {
      printf("Informe um valor para o elemento nr. %2d: ", I);
      scanf("%d", &A[I]);
    }

  /* Processamento par ou impar */

  for (I = 0; I <= 9; I ++)
    if (I % 2 == 0)
      B[I] = A[I] * 5;
    else
      B[I] = A[I] + 5;

  /* Apresentacao das matrizes */

  for (I = 0; I <= 9; I ++)
    printf("\nA[%2d] = %2d | B[%2d] = %2d", I + 1, A[I], I + 1, B[I]);
  return 0;
}
```

Digite o programa, gravando-o com o nome **chkind.c**. Observe que são utilizados no programa três laços de repetição **for**. O primeiro laço controla a entrada dos dados, o segundo verifica se cada índice da matriz A é par ou ímpar e faz as operações implicando os elementos calculados na matriz B, o terceiro é utilizado para apresentar as duas matrizes.

No laço de repetição destinado ao processamento, é utilizada a instrução **if** (*I* **%** 2 = 0) **then**, sendo **%** uma função da linguagem C que possibilita extrair o resto de uma divisão de números inteiros. Essa operação recebe o nome de módulo. Qualquer valor de dividendo dividido por 2 que resultar zero tem o dividendo par. Se o resto da divisão for 1, o dividendo é um valor ímpar.

6.2.4 Programa Elemento

Desenvolver um programa que leia cinco elementos de uma matriz A do tipo vetor. No final, apresente o total da soma de todos os elementos que sejam ímpares.

Algoritmo

Em relação ao primeiro exemplo, este apresenta uma diferença. O primeiro pedia para verificar se o índice era par ou ímpar.

Neste exemplo, é solicitado que se analise a condição do *elemento* e não do índice. Já foi alertado anteriormente para tomar cuidado e não confundir elemento com índice. Veja a solução.

1. Iniciar o contador de índice, variável *I* como 0 (zero) em um contador até 5.

2. Ler os cinco valores, um a um.

3. Verificar se o elemento é ímpar; se sim, efetuar a soma dos elementos.

4. Apresentar o total de todos os elementos ímpares da matriz.

Programa

```
/* Soma os elementos impares de um vetor */
#include <stdio.h>
int main(void)
{
  int A[5];
  int I, SOMA = 0;
  printf("\nSomatorio de elementos impares\n\n");

  /* Entrada de dados */

  for (I = 0; I <= 4; I ++)
    {
      printf("Informe um valor para o elemento nr. %2d: ", I);
      scanf("%d", &A[I]);
    }

  /* Processamento elemento impar */

  for (I = 0; I <= 4; I ++)
    if (A[I] % 2 != 0)
      SOMA += A[I];

  /* Apresentacao das matrizes */

  printf("\nA soma dos elementos equivale a: %4d\n\n", SOMA);
  return 0;
}
```

Digite o programa, gravando-o com o nome **chkele.c**. No laço de repetição destinado ao processamento foi utilizada a instrução **if** (*A*[*I*] **%** 2 **==** 1), para verificar se o elemento informado pelo teclado é um valor ímpar; sendo, ele é

acumulado na variável *SOMA* que ao final apresenta o somatório de todos os elementos ímpares digitados durante a execução do programa.

Observação

Quando se faz menção ao índice, indica-se a variável que controla o contador de índice, no caso do exemplo anterior, a variável *I*. Quando se faz menção ao elemento, indica-se: *A*[*I*], pois pega-se o valor armazenado e não a sua posição de endereço.

Com a finalidade de fixar os conceitos expostos, nesta etapa é aconselhável resolver os exercícios de fixação 1 de a até h.

6.3 Matrizes com Mais de Uma Dimensão

Com o conhecimento adquirido, você tem condições suficientes para elaborar um programa que leia as notas dos alunos, o cálculo da média de cada aluno e no final apresente a média do grupo, utilizando apenas matrizes unidimensionais. Porém, é preciso considerar que o trabalho seria grande, uma vez que seria necessário manter um controle de cada índice em cada matriz para um mesmo aluno.

Para facilitar o trabalho com estruturas deste porte são utilizadas matrizes com mais dimensões. A mais comum é a matriz de duas dimensões por se relacionar diretamente com tabelas. Matrizes com mais de duas dimensões são utilizadas com menos frequência, mas às vezes é preciso trabalhar com um número maior de dimensões. Elas são fáceis de utilizar se você dominar bem a matriz com duas dimensões.

Em matrizes de mais de uma dimensão, os elementos são manipulados de forma individualizada, sendo a referência feita sempre por meio de dois índices, sendo o primeiro para indicar a linha e o segundo para indicar a coluna. Desta forma, TABELA[2][3] indica que está sendo feita uma referência ao elemento armazenado na linha 2 coluna 3.

A linguagem C não trabalha diretamente com matrizes que tenham mais de uma dimensão como outras linguagens fazem, mas permite a simulação perfeita desse efeito. Isso ocorre porque essa linguagem trabalha com matrizes de matrizes, ou seja, uma matriz considerada de duas dimensões é na verdade uma matriz em que cada um dos elementos é outra matriz.

Uma matriz de duas dimensões faz menção a linhas e colunas e é representada pelo nome e tamanho (dimensão) entre colchetes. Desta forma, é uma matriz de duas dimensões TABELA[8][5], cujo nome é TABELA, com tamanho de 8 linhas (de 0 a 7) e 5 colunas (de 0 a 4), ou seja, é uma matriz de 8 por 5 (8 x 5). Isso significa que podem ser armazenados em TABELA até 40 elementos.

A Figura 6.1 apresenta a matriz TABELA com a indicação dos endereços (posições) que podem ser utilizados para armazenar os elementos.

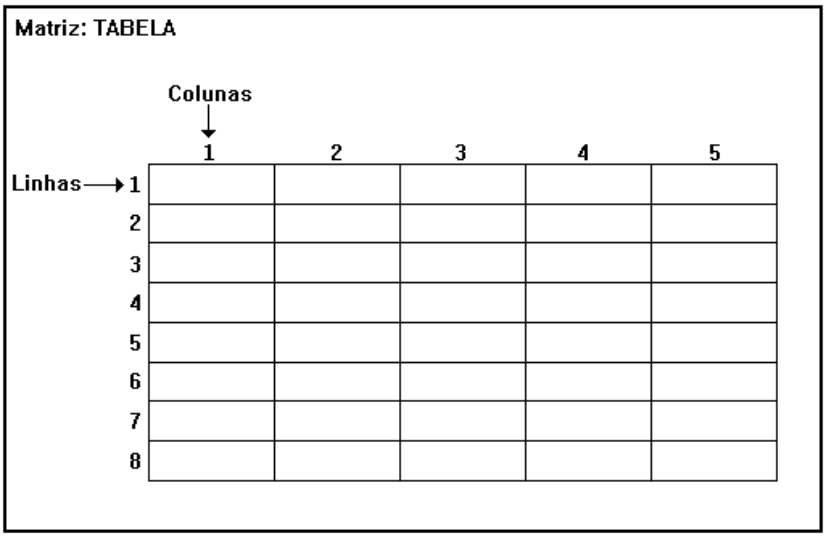

Figura 6.1 - Exemplo da matriz TABELA com as posições.

Uma matriz de duas dimensões é atribuída de forma semelhante a uma matriz de uma dimensão, sendo representada pelo nome, tamanho (dimensão de linhas e colunas) entre colchetes e tipo, tendo a seguinte sintaxe:

```
tipo MATRIZ[dimensão linha][dimensão coluna];
```

em que:

<tipo> - o tipo de dado a ser guardado na matriz;

<matriz> - o nome atribuído à matriz;

<dimensão linha> - o tamanho da matriz em número de linhas;

<dimensão coluna> - o tamanho da matriz em número de colunas.

Para mostrar o uso de matrizes de duas dimensões, será apresentado em seguida mais um exemplo de aprendizagem.

6.3.1 Programa Notas Escolares

Desenvolver um programa que efetue a entrada e a saída das notas escolares de oito alunos.

Algoritmo

A leitura e a escrita de uma matriz de duas dimensões, assim como as matrizes de uma dimensão, são processadas passo a passo.

Considerando a manipulação de quatro notas de oito alunos, a tabela em questão armazena 32 elementos. Um detalhe a ser levado em conta é a utilização de duas variáveis para controlar os dois índices de posicionamento de dados na tabela, um para cada dimensão.

Programa

```
/* Calculo de Media de 8 alunos com 4 notas */
#include <stdio.h>
int main(void)
{
  float MD[8][4], NUMERO;
  int I, J;
  printf("\nLeitura e apresentacao de notas\n");

  /* Entrada das notas */

  for (I = 0; I <= 7; I ++)
    {
      printf("\nInforme as notas do %do. aluno: \n\n", I + 1);
      for (J = 0; J <= 3; J ++)
        {
          printf("Nota %d: ", J + 1);
          scanf("%f", &NUMERO);
          MD[I][J] = NUMERO;
        }
    }

  /* Saida das notas */
```

```
  for (I = 0; I <= 7; I ++)
    {
      printf("\nAs notas do aluno %d sao : \n\n",  I+ 1);
      for (J = 0; J <= 3; J ++)
        printf("Nota %d: %5.2f\n",  J+ 1, MD[I][J]);
    }
  return 0;
}
```

Em seguida digite o programa, gravando-o com o nome **notas.c**. Em um exemplo anterior foi utilizada a variável *I* para controlar as posições dos elementos dentro da matriz, ou seja, a posição em nível de linha. Neste exemplo, a variável *I* continua com a mesma função e a segunda variável, a *J*, controla a posição da coluna.

Analisando o programa, tem-se a inicialização das variáveis *I* e *J* como 0 por meio dos laços **for**, ou seja, a leitura é feita na primeira linha da primeira coluna (0,0). Em seguida é iniciado em primeiro lugar o laço da variável *I* para controlar a posição em relação às linhas e depois é iniciado o laço da variável *J* para controlar a posição em relação às colunas.

Ao serem iniciados os valores para o preenchimento da tabela, eles são colocados na posição MD[0,0] (primeira linha e primeira coluna), lembrando que o primeiro valor dentro dos colchetes representa a linha e o segundo representa a coluna. Assim sendo, será então digitada para o primeiro aluno a sua primeira nota. Depois é incrementado 1 em relação à coluna, sendo colocada para a entrada a posição MD[0,1] (primeira linha e segunda coluna) da tabela. Desta forma, para o primeiro aluno digita-se a segunda nota.

Observação

A entrada de valores reais na matriz MD ocorre com a utilização de uma variável de apoio (tipo *float*) representada pelo rótulo NUMERO. A entrada não é feita diretamente na matriz MD e sim por intermédio da variável de apoio NUMERO. Isso é necessário, pois a entrada direta de valores reais em matrizes não é aceita corretamente pelos compiladores de linguagem C. É ideal sempre utilizar uma variável de apoio para a entrada de valores reais em matrizes de uma ou mais dimensões.

Quando o contador de coluna, o laço da variável *J*, atingir o valor 3, ele é encerrado. Em seguida o contador da variável *I* é incrementado com mais 1,

tornando-se 1. Será então inicializado novamente o contador *J* em 0, permitindo que seja digitado um novo dado na posição MD[1,0].

O mecanismo de preenchimento se estende até que o valor do contador de linhas atinja o seu último valor, no caso, 7. Esse laço de repetição é o principal, tendo a função de controlar o posicionamento na tabela por aluno. O segundo laço de repetição, mais interno, controla o posicionamento das notas. Em seguida é apresentada a relação dos alunos e suas respectivas notas.

Com a finalidade de fixar os conceitos estudados, aconselha-se resolver os exercícios de fixação 2 de "a" até "h".

6.4 String

É um tipo de dado importante encontrado nas linguagens de programação, pois é usado para a manipulação e armazenamento de textos (sequências de caracteres). Na linguagem C, esse tipo de dado recebe um tratamento diferente de outras linguagens, pois é tratado como um vetor (matriz de uma dimensão) do tipo **char**, o qual termina com um caractere "**\n**". Por esta razão este assunto é tratado separadamente neste tópico.

Já é sabido que uma matriz é o conjunto de dados de mesmo tipo, armazenado em uma mesma variável e controlado por um índice. Em linguagem C, *string* é um conjunto de elementos **char**. Desta forma, é possível acessar qualquer um dos elementos da referida matriz.

Para exemplificar o uso de *strings,* veja em seguida dois exemplos de aprendizagem.

6.4.1 Programa Saudação

Vamos estudar como trabalhar com *strings variáveis*, as quais são fornecidas via teclado, como um nome, um endereço, um número de telefone etc. Esses *strings* entram para processamento por meio da função **scanf()** com a utilização do formato **%s**.

A seguir, acompanhe um exemplo com um programa que pede o nome e o sobrenome, faz uma saudação e apresenta o nome completo.

```
/* Pede dados pessoais */
#include <stdio.h>
int main(void)
{
  char NOME[10], SB_NOME[15];
  printf("Informe seu nome ........: "); scanf("%s", NOME);
  printf("Informe seu sobre nome ..: "); scanf("%s", SB_NOME);
  printf("Seja bem vindo, %s %s\n\n", NOME, SB_NOME);
  return 0;
}
```

O exemplo anterior usa duas matrizes do tipo *string*, sendo uma *NOME* com a capacidade de armazenar até nove caracteres e a outra *SB_NOME* com a capacidade de armazenar até 14 caracteres. Em um *string* o último caractere automaticamente é o valor "\0", o qual é associado assim que a tecla **<Enter>** é acionada. Os tamanhos máximos de caracteres permitidos para as variáveis *NOME* e *SB_NOME* são, respectivamente, 9 e 14.

Caso deseje trabalhar com os tamanhos 10 e 15, é necessário definir as variáveis *NOME* e *SB_NOME* com os tamanhos 11 e 16 (**char** *NOME*[11], *SB_NOME*[16];). Grave o programa com o nome **saudação.c**.

As instruções **scanf(**"%s", *NOME***);** e **scanf(**"%s", *SB_NOME***);** fazem a leitura de cada caractere digitado e armazenam sequencialmente a partir do endereço de *NOME* e *SB_NOME*. O processo é encerrado quando um caractere branco é encontrado na *string*. Neste momento, o caractere "\0" é colocado automaticamente nessa posição. É necessário tomar cuidado com o tamanho da matriz *string*. Se a matriz possui declarado um valor de tamanho como 10, você só pode usar 9, deixando pelo menos uma posição livre para o caractere "\0".

Outro detalhe a ser observado é a ausência do operador **&** precedendo o segundo argumento das instruções de entrada com a função **scanf()**. Quando se utilizam matrizes, esse operador não pode ser usado, pois o nome de uma matriz é o seu endereço inicial. A expressão **scanf(**"%s", *NOME***);** é equivalente a **scanf(**"%s", &*NOME*[0]**);.**

Por exemplo, se for informado para *NOME* o *string Augusto* e em seguida para *SB_NOME* o *string Manzano*, é apresentada a seguinte mensagem: *Seja bem-vindo, Augusto Manzano*. Porém, se para NOME for digitado *José Augusto*, o programa não solicita o sobrenome, pois no *string José Augusto* existe um espaço em branco entre *José* e *Augusto*. O *string José* fica armazenado em *NOME* e o *string Augusto* em *SB_NOME*.

Esse efeito indesejável ocorre porque a função **scanf()** é bastante limitada para a leitura de *strings* quando em uso **%s**, pois quando é colocado um espaço em branco na sequência, a função **scanf()** considera o espaço em branco como um caractere nulo. Daí para frente tudo o que for escrito será automaticamente desconsiderado.

6.4.2 Programa Saudação 2

Para resolver o problema relacionado à entrada de *strings* da função **scanf()** com **%s**, pode-se utilizar a função de entrada de dados **fgets()** que pertence à biblioteca **stdio.h**.

A função **fgets()** possui o propósito de ler dados do tipo *string* fornecidos via teclado. Ela aceita caracteres em branco no meio do *string*, colocando o caractere de controle "**\n**" apenas quando for pressionada a tecla **<Enter>**.

Para utilizar essa função, é preciso informar três parâmetros, sendo o nome da variável *string*, o tamanho da matriz *string* e o nome do *buffer* em uso, portanto a função **fgets()** é escrita como **fgets(**VARIAVEL, TAMANHO, stdin**);**.

O programa seguinte usa a função fgets() para solicitar a entrada do nome e sobrenome. Após digitar o programa, grave-o com o nome **saudaca2.c**.

```
/* Pede dados pessoais - fgets() */
#include <stdio.h>
int main(void)
{
  char NOME[20], SB_NOME[20];
  printf("Informe seu nome ........: "); fgets(NOME, 20, stdin);
  printf("Informe seu sobre nome ..: "); fgets(SB_NOME, 20, stdin);
  printf("Seja bem vindo, %s %s\n\n", NOME, SB_NOME);
  return 0;
}
```

Além do uso da função **fgets()**, foi aumentado o tamanho das matrizes para evitar algum corte no *string*.

Assim como a função **scanf()** foi substituída pela função **fgets()**, também pode ser substituída a função **printf()** pela **puts()**. A função **puts()** é complemento da **fgets()** e sua principal característica é permitir a impressão de um único *string* por vez, além de pular sozinha uma linha após a impressão de um *string*.

Deve-se considerar que a função **puts()** (pertencente à biblioteca **stdio.h**) não será usada para todas as aplicações, pois ela somente aceita um argumento, a variável que será impressa, podendo causar alguns efeitos visuais distorcidos e incômodos. Veja em seguida o exemplo:

```
/* Dados pessoais com fgets() e puts() */
#include <stdio.h>
int main(void)
{
  char NOME[20], SB_NOME[20];
  puts("Informe seu nome ........: "); fgets(NOME, 20, stdin);
  puts("Informe seu sobre nome ..: "); fgets(SB_NOME, 20, stdin);
  printf("Seja bem vindo, %s %s\n\n", NOME, SB_NOME);
  return 0;
}
```

Digite e grave o programa com o nome **saudaca3.c**; execute-o e observe seu funcionamento.

Ao encontrar a instrução **puts(**"Digite o seu nome .....: "**);**, o cursor é posicionado na linha seguinte, em que será fornecido o respectivo dado. A apresentação do nome completo é realizada com a função **printf()**.

6.4.3 Programa Saudação 3

O uso da função **fgets()** possibilita a entrada de uma sequência de caracteres com espaços em branco. No entanto, essa função acrescenta o caractere "**\n**" que registra ao final do *string* a mudança de linha. Há situações em que esse procedimento é inconveniente.

O que muitos programadores iniciantes desconhecem é a capacidade de omitir o registro do caractere "**\n**". Isso é conseguido com a função **scanf()** que não aceita espaços em branco no meio de uma sequência de caracteres quando utiliza **%s**. Mas há uma forma de fazer com que a função **scanf()** aceite espaços em branco, o que é conseguido com **%[** e **]**, que possui como estrutura a seguinte sintaxe:

```
%tamanho[caracteres]
```

Sendo **tamanho** a definição opcional (pode ser omitida) da quantidade de caracteres a serem lidos pela função **scanf()** e **caracteres** é a indicação dos caracteres que podem ou não ser aceitos no *string* informado.

Dentro dos colchetes podem ser utilizados como delimitadores de caracteres os símbolos (**-**) subtração e (**^**) circunflexo. O símbolo de subtração é utilizado para uma faixa de caracteres a ser aceita e o circunflexo para determinar os caracteres que devem ser desconsiderados do *string* definido.

No sentido de exemplificar o recurso exposto considere um programa que solicita o nome do usuário e apresenta uma mensagem de saudação com o nome. Grave o programa com o nome **saudação3.c**.

```
/* Dados pessoais com scanf() formatado com %[] */
#include <stdio.h>
int main(void)
{
  char NOME[40];
  printf("Entre o nome completo: "); scanf("%[^\n]", NOME);
  printf("Ola, %s\n", NOME);
  return 0;
}
```

O código do programa **saudacao3.c** efetua a entrada de um nome. Todos os caracteres alfanuméricos são aceitos, exceto o caractere "**\n**" que está sendo excluído da sequência pelo símbolo circunflexo. A forma utilizada dispensa inclusive o uso da função **fflush(stdin)**.

Dentre as possibilidades de operacionalização do código de formato **%[ ]** é possível também usar:

- **%10[1234567890]** - define a entrada de até dez caracteres representados por uma sequência numérica.
- **%[^\f\n\r\t\v]** - define a entrada de qualquer caractere menos os indicados.
- **%5[a-z]** - indica a entrada de até cinco caracteres que sejam apenas minúsculos.
- **%35[A-Z]** - determina a entrada de até 35 caracteres que sejam apenas maiúsculos.
- **%25[A-Z a-z]** - determina a entrada de até 25 caracteres que sejam apenas maiúsculos ou minúsculos.

6.5 Aplicações Práticas com Matrizes

A utilização de matrizes em programação é bastante ampla. Podem ser utilizadas em diversas situações, tornando bastante útil e versátil a técnica de programação.

6.5.1 Entrada e Saída de Elementos

Para ter uma outra ideia, considere um programa que necessite ler e apresentar os nomes de dez pessoas. Ele lê e escreve os dez nomes. Considerar que cada nome tenha até 40 caracteres de comprimento, portanto será uma matriz bidimensional.

Algoritmo

1. Definir a variável *I* do tipo inteira para controlar a malha de repetição.
2. Definir a matriz *NOME* do tipo caractere para dez elementos de 40 posições.
3. Iniciar o programa com a leitura dos dez nomes.
4. Apresentar após a leitura os dez nomes.

Programa

```
/* Leitura e escrita de 10 nomes */
#include <stdio.h>
int main(void)
{
  int I;
  char NOME[10][40];
  printf("\n\nListagem de nomes\n\n");

  /* Entrada dos dados */

  for (I = 0; I <= 9; I ++)
    {
      printf("Digite o %2do. nome: ", I + 1);
      fflush(stdin); fgets(NOME[I], 40, stdin);
      scanf("%40[^\n]", NOME[I]);
    }

  /* Apresentacao dos nomes */

  for (I = 0; I <= 9; I ++)
    printf("Nome: %2d --> %s", I + 1, NOME[I]);
  return 0;
}
```

Digite o programa, gravando-o com o nome **listnome.c**. Após a execução os nomes são apresentados na mesma ordem em que foram informados.

Antes da entrada de um nome com a função **fgets(**NOME[I], 40, stdin**)**, utiliza-se a função **fflush()** com o parâmetro **stdin**.

A função **fflush()** é um comando de limpeza de *buffer* (*stream*) de teclado, em que **stdin** (standard input) é o *stream* a ser limpo, ou seja, o *stream* de entrada-padrão , no caso representado pelo teclado.

Quando se faz a entrada de dados numéricos, o *buffer* de teclado não necessita ser limpo, porém quando se trata de dados *string*, é preciso fazer a limpeza, senão o programa não funciona corretamente. Este efeito ocorre devido ao registro na entrada do caractere especial de mudança de linha que é colocado (**\n**) quando do uso da tecla **<Enter>**. Por esta razão a função **fflush()** precede a função **fgets()**, pois o caractere **\n** fica alojado no *buffer* e interfere na próxima entrada.

Nada impede de utilizar a função **fflush()** antes de uma função **scanf()**. Deste ponto em diante todos os programas desta obra vão utilizar **fflush()** tanto para dados do tipo *string* como dados dos tipos numéricos.

6.5.2 Classificação de Elementos

Construído o programa de entrada e saída dos dez nomes na matriz, é bastante útil que, antes de apresentá-los, o programa faça a classificação alfabética, apresentando os nomes em ordem, independentemente daquela em que foram informados, facilitando a localização de algum nome, quando for realizada uma pesquisa visual.

A seguir, veja dois exemplos de ordenação, sendo um para a ordenação de uma matriz inteira e outro de uma matriz *string*.

6.5.2.1 Elementos do Tipo Numérico

Imagine a seguinte situação: colocar em ordem crescente cinco valores numéricos armazenados numa matriz A e apresentá-los. Considere para esta explicação os valores apresentados na tabela seguinte:

Tabela 6.1 - Matriz de dados

Matriz: A	
Índice	**Elemento**
1	9
2	8
3	7
4	5
5	3

Os valores (elementos) estão armazenados na or-dem 9, 8, 7, 5 e 3 e devem ser apresentados na ordem 3, 5, 7, 8 e 9. Convertendo a tabela no formato matricial, teremos então:

```
A[1] = 9
A[2] = 8
A[3] = 7
A[4] = 5
A[5] = 3
```

Para realizar a troca, é necessário aplicar o método de propriedade distributiva. O elemento que estiver em A[1] deve ser comparado com os elementos que estiverem em A[2], A[3], A[4] e A[5]. Depois, o elemento que estiver em A[2] não necessita ser comparado com o que estiver em A[1], pois já foram anteriormente comparados, passando a ser comparados somente com os elementos que estiverem em A[3], A[4] e A[5]. Na sequência, o elemento que estiver em A[3] é comparado com os elementos que estiverem em A[4] e A[5] e por fim o elemento que estiver em A[4] é comparado com o que estiver em A[5].

Seguindo este raciocínio, basta comparar o valor do elemento armazenado em A[1] com o valor do elemento armazenado em A[2]. Se o primeiro for maior que o segundo, trocam-se os valores. Como a condição de troca é verdadeira, o elemento 9 de A[1] é maior que o elemento 8 de A[2]. Passa-se para A[1] o elemento 8 e para A[2] passa-se o elemento 9, desta forma os valores dentro da matriz ficam com a seguinte formação:

```
A[1] = 8
A[2] = 9
A[3] = 7
A[4] = 5
A[5] = 3
```

Seguindo a regra de ordenação, o atual valor de A[1] deve ser comparado com o próximo valor após a sua última comparação. Sendo assim, ele deve

ser comparado com o valor existente em A[3]. O atual valor do elemento de A[1] é maior que o valor do elemento de A[3], ou seja, 8 é maior que 7. Realiza-se a troca, ficando A[1] com o elemento de valor 7 e A[3] com o elemento de valor 8. Os valores da matriz passam a ter a seguinte formação:

```
A[1] = 7
A[2] = 9
A[3] = 8
A[4] = 5
A[5] = 3
```

Agora devem ser comparados os valores dos elementos armazenados nas posições A[1] e A[4]. O valor do elemento 7 de A[1] é maior que o valor do elemento 5 de A[4]. São estes trocados, passando A[1] a possuir o elemento 5 e A[4] a possuir o elemento 7. A matriz passa a ter a seguinte formação:

```
A[1] = 5
A[2] = 9
A[3] = 8
A[4] = 7
A[5] = 3
```

Até agora os elementos comparados foram sendo trocados de posição, estando agora em A[1] o elemento de valor 5 e que será mudado mais uma vez por ser maior que o valor do elemento 3 armazenado em A[5]. Desta forma a matriz passa a ter a seguinte formação:

```
A[1] = 3
A[2] = 9
A[3] = 8
A[4] = 7
A[5] = 5
```

A partir deste ponto, o elemento de valor 3 armazenado em A[1] não necessita mais ser comparado. Assim sendo, pega-se o atual valor do elemento da posição A[2] e compara-se sucessivamente com todos os outros elementos restantes. É preciso destacar que o valor do elemento armazenado em A[2] deve ser comparado com os elementos armazenados em A[3], A[4] e A[5], segundo a regra da aplicação de propriedade distributiva.

Comparando o valor do elemento 9 da posição A[2] com o elemento 8 da posição A[3] e efetuando a troca de forma que 8 esteja em A[2] e 9 esteja em A[3], a matriz passa a ter a seguinte formação:

```
A[1] = 3
A[2] = 8
A[3] = 9
A[4] = 7
A[5] = 5
```

O atual valor do elemento de A[2] deve ser comparado com o valor do elemento de A[4]. Sendo 8 maior que 7, são trocados, ficando A[2] com 7 e A[4] com 8. A matriz passa a ter a seguinte formação:

```
A[1] = 3
A[2] = 7
A[3] = 9
A[4] = 8
A[5] = 5
```

Prossegue o processo de comparação e troca. O atual valor do elemento na posição A[2] é 7 e será comparado com o valor do elemento A[5] que é 5. São estes trocados, passando A[2] ficar com o elemento 5 e A[5] com o elemento 7, conforme indicado no esquema:

```
A[1] = 3
A[2] = 5
A[3] = 9
A[4] = 8
A[5] = 7
```

Até este ponto a posição A[2] foi comparada com todas as posições subsequentes, não tendo mais nenhuma comparação para ela. Agora é feita a comparação da próxima posição com o restante, no caso, de A[3] com A[4] e A[5]. Sendo assim, o valor do elemento da posição A[3] será comparado com o valor da posição A[4]. Serão estes trocados, ficando A[3] com 8 e A[4] com 9, conforme em seguida:

```
A[1] = 3
A[2] = 5
A[3] = 8
A[4] = 9
A[5] = 7
```

Compara-se o valor do elemento da posição A[3] com o valor do elemento da posição A[5]. Sendo o primeiro maior que o segundo, ocorre a troca. Desta forma A[3] passa a possuir o elemento 7 e A[5] passa a possuir o elemento 8, como indicado em seguida:

```
A[1] = 3
A[2] = 5
A[3] = 7
A[4] = 9
A[5] = 8
```

Efetuadas todas as comparações de A[3] com A[4] e A[5], fica somente a última comparação que é A[4] com A[5], cujos valores são trocados, passando A[4] possuir o elemento de valor 8 e A[5] o elemento de valor 9, como mostrado em seguida:

```
A[1] = 3
A[2] = 5
A[3] = 7
A[4] = 8
A[5] = 9
```

A referida ordenação foi executada. Apresentando os elementos da matriz em ordem crescente, tem-se a seguinte situação de saída:

```
A[1] = 3
A[2] = 5
A[3] = 7
A[4] = 8
A[5] = 9
```

Para dados do tipo caractere o processo é idêntico, uma vez que cada letra possui um valor diferente da outra. A letra "A", por exemplo, tem valor menor que a letra "B" e assim por diante. Se a letra "A" maiúscula for comparada com a letra "a" minúscula, elas terão valores diferentes.

Cada caractere é guardado na memória de um computador segundo o valor de um código que recebe o nome de ASCII (*American Standard Code for Information Interchange* - Código Americano Padrão para Troca de Informações - consultar apêndice da obra). E é com base nessa tabela que o processo de ordenação trabalha, pois cada caractere tem um peso, um valor previamente determinado, segundo este padrão.

Algoritmo

1. Definir a variável *I* inteira para controlar a malha de repetição.
2. Definir a matriz *A* inteira para cinco elementos.
3. Colocar em ordem crescente os elementos da matriz.
4. Iniciar o programa fazendo a leitura dos cinco números.
5. Apresentar após a leitura os cinco números ordenados.

Programa

```
/* Escreve numeros ordenados */
#include <stdio.h>
int main(void)
{
  int NUMERO[5], I, J, X;
  printf("Classificacao de elementos numericos\n\n");

  /* Entrada de dados */

  for (I = 0; I <= 4; I ++)
    {
      printf("Informe o %do. valor: ", I + 1);
      fflush(stdin); scanf("%d", &NUMERO[I]);
    }

  /* Classificacao */

  for (I = 0; I <= 3; I ++)
    for (J = I + 1; J <= 4; J ++)
      if (NUMERO[I] > NUMERO[J])
        {
          X = NUMERO[I];
          NUMERO[I] = NUMERO[J];
          NUMERO[J] = X;
        }
  /* Apresentacao dos dados */

  printf("\n\n");
  for (I = 0; I <= 4; I ++)
    printf("Agora o %do. valor e': %3d\n", I + 1, NUMERO[I]);
  return 0;
}
```

Escreva o programa, grave-o com o nome **sorttabi.c** e execute-o para comprovar o seu funcionamento. Em seguida atente para dois detalhes.

O primeiro é a utilização de uma segunda variável no processo de ordenação, no caso, a variável *J*. Somente quando a variável *J* atinge o valor 4 é que esse laço de repetição se encerra, retornando o processamento ao laço da variável *I,* acrescentando um a *I* até que *I* atinja o seu limite e ambos os laços sejam encerrados. O quadro seguinte mostra o comportamento dos valores das variáveis *I* e *J* durante o processo de ordenação.

Atente para o detalhe das instruções **for** encadeadas. Vale salientar que no caso de encadeamento, será executada primeiramente a rotina mais interna, no caso a de contagem da variável *J*, passando o processamento para a rotina mais externa, quando a mais interna fechar o seu ciclo.

Tabela 6.2 - Valores das variáveis em processamento

Quando I for	J será
0	1, 2, 3, 4
1	2, 3, 4
2	3, 4
3	4

Quando a variável *I* for 0, a variável *J* será 1 e contará até 4. Ao final desse ciclo, a variável *I* é acrescida de 1, tornando-se 1, assim sendo a variável *J* passa a ser 2. Quando a variável *J* voltar a ser 4 novamente, a variável *I* passa a ser 2 e a variável *J* passa a ser 3. Esse ciclo é executado até que por fim a variável *I* seja 3 e a variável *J* seja 4, e o penúltimo elemento comparado com o seu elemento subsequente, no caso, o último.

O segundo ponto a ser observado é a utilização do algoritmo de troca, utilizado com a instrução **if** (*NUMERO*[*I*] > *NUMERO*[*J*]). Após a verificação dessa condição, sendo o primeiro número maior que o segundo, realiza-se a troca com a sequência:

```
X = NUMERO[I]
NUMERO[I] = NUMERO[J]
NUMERO[J] = X
```

Considere o vetor *NUMERO*[*I*] com o valor 8 e o vetor *NUMERO*[*J*] com o valor 9. Ao final, *NUMERO*[*I*] deve estar com 9 e *NUMERO*[*J*] deve estar com 8. Para conseguir esse efeito, é necessário usar uma variável de apoio, a qual será chamada *X*.

Para que o vetor *NUMERO*[*I*] fique livre para receber o valor do vetor *NUMERO*[*J*], *X* deve ser implicado pelo valor do vetor *NUMERO*[*I*], então *X* passa a ser 8. Neste momento pode-se implicar o valor de *NUMERO*[*J*] em *NUMERO*[*I*]. Desta forma o vetor *NUMERO*[*I*] passa a possuir o valor 9. Em seguida o vetor *NUMERO*[*J*] é implicado pelo valor que está em *X*. Ao final desse processo ter-se-á *NUMERO*[*I*] com 9 e *NUMERO*[*J*] com 8.

6.5.2.2 Elementos do Tipo Caractere

Agora é exemplificado como fazer a ordenação de uma matriz de *strings* para dez nomes. O processo é semelhante à ordenação de valores, uma vez que cada letra possui um valor diferente da outra. A letra A, por exemplo, tem valor menor que a letra B e assim por diante. Se a letra A maiúscula for comparada com a letra a minúscula, elas terão valores diferentes.

Cada caractere é guardado na memória de um computador segundo o valor de um código que recebe o nome de ASCII (*American Standard Code for Information Interchange* - Código Americano Padrão para Troca de Informações). E é com base nessa tabela que o processo de ordenação trabalha, pois cada caractere tem um peso, um valor previamente determinado, segundo este padrão.

Como é feita a ordenação de valores do tipo *string*, são utilizadas as funções **strcmp()** e **strcpy()** da biblioteca (arquivo de cabeçalho) **string.h**, que são específicas para o tratamento do dado *string*.

A função **strcmp()** processa a comparação de dois *strings*, retornando um de três valores possíveis:

- maior que zero (>0), caso o primeiro *string* seja maior que o segundo *string*;
- igual a zero (=0), caso o primeiro *string* seja igual ao segundo *string*;
- menor que zero (<0), caso o primeiro *string* seja menor que o segundo *string*.

Depois de executada cada comparação, a função **strcpy()** faz a cópia da cadeia de caracteres do segundo *string* para o primeiro *string*, ou seja, uma operação semelhante à troca de valores entre duas variáveis.

Algoritmo

1. Definir a variável I inteira para controlar a malha de repetição.
2. Definir a matriz NOME caractere para dez elementos.
3. Colocar em ordem crescente os elementos da matriz.
4. Iniciar o programa com a leitura dos dez nomes.
5. Apresentar após a leitura os dez nomes ordenados.

Programa

```
/* Leitura, ordenacao e escrita */
#include <stdio.h>
#include <string.h>
int main(void)
{
  int I, J;
  char NOME[10][40], X[40];
  printf("Listagem de nomes\n\n");

  /* Entrada dos dados */

  for (I = 0; I <= 9; I ++)
    {
      printf("Digite o %2do. nome: ", I + 1);
      fflush(stdin); fgets(NOME[I], 40, stdin);
    }

  /* Ordenacao de strings */

  for (I = 0; I <= 8; I ++)
    for (J = I + 1; J <= 9; J ++)
      if (strcmp(NOME[I], NOME[J]) > 0)
        {
          strcpy(X, NOME[I]);
          strcpy(NOME[I], NOME[J]);
          strcpy(NOME[J], X);
        }

  /* Apresentacao dos nomes */

  printf("\n");
  for (I = 0; I <= 9; I ++)
    printf("Nome: %2d --> %s", I + 1, NOME[I]);
  return 0;
}
```

Em seguida digite o programa, gravando-o com o nome **sortnome.c**. Observe o detalhe de definição da variável *X* como uma matriz do tipo *string* com a capacidade de armazenar até 40 caracteres. Isso é necessário, uma vez que a variável que será usada como suporte para a troca deve ser do mesmo tipo que a matriz de elementos.

6.5.3 Pesquisa de Elementos em uma Tabela

A utilização de matrizes pode gerar grandes tabelas, dificultando a localização de um determinado elemento de forma rápida. Imagine uma matriz com 4.000 elementos (4.000 nomes de pessoas). Será que você conseguiria encontrar

rapidamente um elemento desejado de forma manual, mesmo estando a lista de nomes em ordem alfabética? Certamente que não.

Para solucionar este problema, você pode fazer pesquisas em matrizes com programação.

6.5.3.1 Pesquisa Sequencial com Caracteres

Considerando a necessidade de trabalhar com uma matriz que possua dez nomes, veja em seguida um programa para fazer uma pesquisa na referida matriz. O método apresentado é o de pesquisa sequencial.

Algoritmo

O algoritmo seguinte estabelece a entrada de dez nomes e a apresentação de nomes que sejam solicitados na fase de pesquisa.

1. Iniciar um contador e pedir a leitura de dez nomes.
2. Criar um looping que faça a pesquisa enquanto o usuário assim desejar:
   a. Na fase de pesquisa, deve ser solicitada a informação a ser pesquisada.
   b. A informação deve ser comparada com o primeiro elemento.
   c. Sendo igual, mostra; caso contrário, avança para o próximo. Se não achar em toda lista.
   d. Informar que não existe o elemento pesquisado; se existir deve mostrá-lo.
3. Encerrar a pesquisa quando desejado.

Programa

```
/* Pesquisa */
#include <stdio.h>
#include <string.h>
int main(void)
{
  int I, ACHA;
  char NOME[10][40], PESQ[40], RESP;
  printf("\nPesquisa sequencial de nomes\n\n");

  /* Entrada dos dados */
```

```
  for (I = 0; I <= 9; I ++)
    {
      printf("Digite o %2do. nome: ", I + 1);
      fflush(stdin); fgets(NOME[I], 40, stdin);
    }

  /* Processamento da pesquisa enquanto RESP = S */


  RESP = 'S';
  while (RESP == 'S' || RESP == 's')
    {
      printf("\nEntre o nome a ser pesquisado: ");
      fflush(stdin); fgets(PESQ, 40, stdin);
      I = 0;
      ACHA = 0;
      while (I <= 9 && ACHA == 0)
        if (strcmp(PESQ, NOME[I]) == 0)
          ACHA = 1;
        else
          I ++;
      if (ACHA == 1)
        printf("%s foi localizado na posicao %d",PESQ, I+1);
      else
        printf("%s nao foi localizado", PESQ);
      printf("\n\nContinua? [S]IM/[N]AO + <Enter>: ");
      fflush(stdin); RESP = getchar();
    }
  return 0;
}
```

Em seguida digite o programa, gravando-o com o nome **pesquisa.c**.

Observe atentamente em seguida as considerações ao trecho responsável pela execução da pesquisa sequencial definida no programa anterior.

```
 1 - printf("\nEntre o nome a ser pesquisado: ");
     fflush(stdin); fgets(PESQ, 40, stdin);
 2 - I = 0;
 3 - ACHA = 0;
 4 - while (I <= 9 && ACHA == 0)
 5 -   if (strcmp(PESQ, NOME[I]) == 0)
 6 -     ACHA = 1;
 7 -  else
 8 -     I ++;
 9 - if (ACHA == 1)
10 -   printf("%s foi localizado na posicao %d", PESQ, I+1);
11 - else
12 -   printf("%s nao foi localizado", PESQ);
13 - printf("\n\nContinua? [S]IM/[N]AO + <Enter>: ");
     fflush(stdin); RESP = getchar();
```

Na linha 1 é solicitado o nome a ser pesquisado na variável *PESQ*, na linha 2 é estabelecido o valor do contador de índice como 0 (primeiro elemento) e na linha 3, a variável *ACHA* é atribuída com o valor 0 (como se estivesse com um valor lógico falso). A linha 4 apresenta a instrução **while,** indicando que enquanto o valor da variável *I* for menor ou igual a 9 e simultaneamente o valor da variável *ACHA* seja 0, deve ser processado o conjunto de instruções situadas nas linhas 5, 6, 7 e 8.

Neste ponto a instrução **if** da linha 5 verifica se o valor da variável *PESQ* é igual ao valor da variável indexada *NOME*[0], e se for igual, é sinal de que o nome foi encontrado. Neste caso, a variável *ACHA* da linha 6 passa a ter o valor 1 (verdadeira), forçando a execução da linha 10. Se o valor da linha 9 é verdadeiro, é apresentada a mensagem da linha 10.

Caso na linha 5 seja verificado que o valor de *PESQ* não é igual a *NOME*[0], será incrementado 1 à variável *I*. Será executada a próxima verificação de *PESQ* com *NOME*[1] e assim por diante. Caso o processamento chegue até o final e não seja encontrado nada, a variável *ACHA* permanece com valor 0. Quando analisada pela linha 9, é considerada falsa e apresenta a mensagem da linha 12.

A variável *ACHA* exerce um papel importante na rotina de pesquisa, pois ela serve como um pivô, estabelecendo um valor verdadeiro quando uma determinada informação é localizada. Esse tipo de tratamento de variável é conhecido pelo nome de FLAG (bandeira). A variável *ACHA* é o flag, podendo-se dizer que, ao começar a rotina, a bandeira estava "abaixada" - falsa, e quando a informação é encontrada, a bandeira é "levantada" - verdadeira, indicando a localização da informação desejada.

6.5.3.2 Pesquisa Sequencial com Números

Com relação à pesquisa, é apresentado um programa de pesquisa sequencial para uma matriz numérica. Digite o programa e grave-o com o nome **pesqnun.c**.

Programa

```
/* Pesquisa 2 */
#include <stdio.h>
int main(void)
{
  int NUMERO[5], I, ACHA, PESQ;
```

```
  char RESP;
  printf("\nPesquisa sequencial de numeros\n\n");

  /* Entrada de dados */

  for (I = 0; I <= 4; I ++)
    {
      printf("Informe o %do. numero: ", I + 1);
      fflush(stdin); scanf("%d", &NUMERO[I]);
    }

  /* Processamento da pesquisa enquanto RESP = S */

  RESP = 'S';
  while (RESP == 'S' || RESP == 's')
    {
      printf("\nEntre o numero a ser pesquisado: ");
      fflush(stdin); scanf("%d", &PESQ);
      I = 0;
      ACHA = 0;
      while (I <= 4 && ACHA == 0)
        if (PESQ == NUMERO[I])
          ACHA = 1;
        else
          I ++;
      if (ACHA == 1)
        printf("%d foi localizado na posicao %d",PESQ, I+1);
      else
        printf("%d nao foi localizado", PESQ);
      printf("\n\nContinua? [S]IM/[N]AO + <Enter>: ");
      fflush(stdin); RESP = getchar();
    }
  return 0;
}
```

Com a finalidade de fixar os temas estudados, é importante resolver os exercícios de fixação 3 de "a" até "h".

6.6 Estruturas, as Matrizes Heterogêneas

Anteriormente você teve contato com matrizes e notou que somente foi possível trabalhar com um tipo de dado por matriz. Para solucionar essa deficiência, será utilizada uma estrutura de dados, denominada na linguagem C de *estruct* (estrutura), a qual consiste em trabalhar com vários dados de tipos diferentes (os membros da estrutura) em uma mesma tabela. Por esta razão, esse dado é considerado heterogêneo.

O que se denomina *estruct* em linguagem C chama-se registro em outras linguagens, enquanto *membros da estrutura* em outras linguagens chamam-se

campos. Na verdade, o conceito é o mesmo, o que muda é a nomenclatura, pois uma estrutura é o conjunto de uma ou mais variáveis, geralmente de tipos diferentes, que usam um mesmo nome.

Considere que sejam informados o nome de um aluno e suas quatro notas bimestrais que devem ser agrupados em uma mesma estrutura. A Figura 6.2 mostra um exemplo do layout de uma estrutura, que é o conjunto de campos. O registro é formado pelos campos Nome, Primeira Nota, Segunda Nota, Terceira Nota e Quarta Nota, o qual pode ser denominado de uma estrutura ou um registro de aluno.

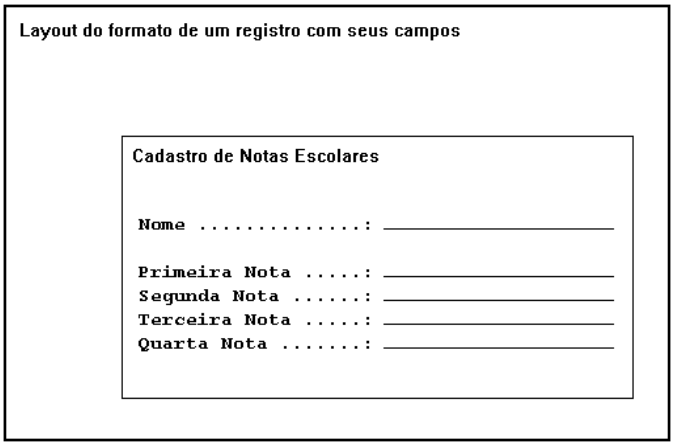

Figura 6.2 - Exemplo da estrutura de um registro com seus campos.

Na linguagem C, as estruturas devem ser declaradas ou atribuídas antes das definições das variáveis, pois é muito comum a necessidade de declarar uma variável com o tipo de estrutura atribuído. Um tipo de estrutura é declarado com a instrução **struct**, conforme a seguinte sintaxe:

```
struct <nome identificador>
  {
    <lista dos tipos e seus membros (campos)>;
  };

struct <nome identificador> <variável>;
```

Em que *nome identificador* é o nome da estrutura (registro), que no livro está com caracteres maiúsculos, em itálico, seguindo as mesmas regras das

variáveis, e *lista dos tipos e seus membros* é a relação de variáveis usadas como campos, bem como o seu tipo, que pode ser float, int, char, entre outros.

6.6.1 Programa Registro

Tomando como exemplo a proposta de criar uma estrutura denominada ALUNO, cujos membros são NOME, NOTA1, NOTA2, NOTA3 e NOTA4, ela seria assim declarada:

```
struct CAD_ALUNO
  {
    char  NOME[41];
    float NOTA1;
    float NOTA2;
    float NOTA3;
    float NOTA4;
  };

struct CAD_ALUNO ALUNO;
```

A estrutura denomina-se *CAD_ALUNO*, a qual é formada por um conjunto de dados heterogêneos (um membro *string* e quatro numéricos com ponto flutuante de precisão simples). Desta forma é possível guardar em uma mesma estrutura vários tipos diferentes de dados. O campo **NOME** é definido com 41 posições, sendo permitido a entrada de no máximo 40 caracteres para o nome de um aluno, mais um caractere para o acionamento da tecla **<Enter>**.

Qualquer menção feita a um membro de uma estrutura, seja uma leitura, uma escrita ou mesmo uma atribuição a uma variável, deve ser feita com o nome da estrutura seguido do nome da variável separado por um caractere "." (ponto) que recebe o nome de operador de associação, como exemplificado em seguida.

```
/* Leitura e escrita de uma estrutura */
#include <stdio.h>
int main(void)
{

  struct CAD_ALUNO
    {
      char  NOME[40];
      float NOTA1;
```

```
      float NOTA2;
      float NOTA3;
      float NOTA4;
    };

  struct CAD_ALUNO ALUNO;

  printf("\nCadastro de aluno\n\n");
  printf("Informe o nome ......: ");
  fflush(stdin); fgets(ALUNO.NOME, 40, stdin);
  printf("Informe a 1a. nota ..: "); scanf("%f", &ALUNO.NOTA1);
  printf("Informe a 2a. nota ..: "); scanf("%f", &ALUNO.NOTA2);
  printf("Informe a 3a. nota ..: "); scanf("%f", &ALUNO.NOTA3);
  printf("Informe a 4a. nota ..: "); scanf("%f", &ALUNO.NOTA4);
  printf("\n");
  printf("Nome .....: %s\n", ALUNO.NOME);
  printf("Nota 1 ...: %6.2f\n" , ALUNO.NOTA1);
  printf("Nota 2 ...: %6.2f\n" , ALUNO.NOTA2);
  printf("Nota 3 ...: %6.2f\n" , ALUNO.NOTA3);
  printf("Nota 4 ...: %6.2f\n" , ALUNO.NOTA4);
  printf("\n\n\n");
  return 0;
}
```

Digite o programa anterior, gravando-o com o nome **aluno1.c**. Na estrutura definida existem quatro variáveis do mesmo tipo para armazenar quatro notas (*NOTA1*, *NOTA2*, *NOTA3* e *NOTA4*). Essa estrutura pode usar uma matriz vetor para armazenar as notas. A Figura 6.3 apresenta uma estrutura para esta proposta.

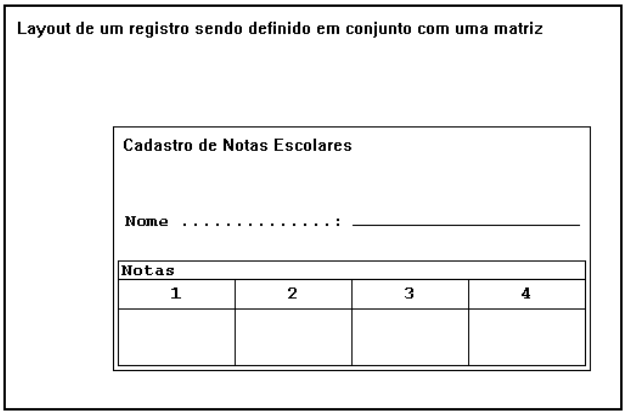

Figura 6.3 - Exemplo da estrutura de um registro usando em conjunto uma matriz.

6.6.2 Programa Registro 2

Tomando por base a proposta de criar um registro denominado ALUNO, cujas notas serão informadas em uma matriz do tipo vetor, ele seria assim declarado:

```
struct CAD_ALUNO
  {
    char  NOME[41];
    float NOTA[4];
  };

struct CAD_ALUNO ALUNO;
```

Ao ser especificada a estrutura *CAD_ALUNO*, existe nela um membro chamado *NOTA*, sendo uma matriz de uma dimensão para quatro elementos. A seguir é apresentado o código-fonte para a leitura e escrita dos dados.

```
/* Leitura e escrita de uma estrutura */
#include <stdio.h>
int main(void)
{

  struct CAD_ALUNO
    {
      char  NOME[41];
      float NOTA[4];
    };
  struct CAD_ALUNO ALUNO;

  int I;
  printf("\nCadastro de aluno\n\n");
  printf("Informe o nome ......: ");
  fflush(stdin); fgets(ALUNO.NOME, 41, stdin);
  for (I = 0; I <= 3; I ++)
    {
      printf("Informe a %da. nota ..: ", I + 1);
      fflush(stdin);
      scanf("%f", &ALUNO.NOTA[I]);
    }
  printf("\n");
  printf("Nome ......: %s\n", ALUNO.NOME);
  for (I = 0; I <= 3; I ++)
    printf("Nota %da ...: %6.2f\n", I + 1, ALUNO.NOTA[I]);
  printf("\n\n\n");
  return 0;
}
```

Digite o programa anterior, gravando-o com o nome **aluno2.c**.

Com as técnicas de programação estudadas, passou-se a ter uma mobilidade bastante grande, com a possibilidade de trabalhar de uma forma mais adequada com diversos problemas, principalmente os que envolvem dados heterogêneos, facilitando a construção de programas mais eficientes.

6.6.3 Programa Registro 3

Os programas apresentados com a utilização de estruturas (registros) só fizeram menção à leitura e escrita de um único dado. Talvez seja pertinente construir um programa que permita trabalhar com vários registros de alunos.

Para exemplificar a ocorrência, considere que você deve elaborar um programa que faça a entrada e a saída de nomes e notas de oito alunos. Isso já é familiar. Veja em seguida uma estrutura para os oito alunos:

```
struct CAD_ALUNO
  {
    char  NOME[40];
    float NOTA[4];
  };

struct CAD_ALUNO ALUNO[8];
```

A linha **struct CAD_ALUNO** *ALUNO*[8]; indica a matriz de oito registros do tipo *CAD_ALUNO* que é formado de dois tipos de dados, sendo o nome como caractere e a nota como ponto flutuante. No programa seguinte é colocada na estrutura a indicação do valor quatro para o vetor *NOTA*, e não três como vinha sendo feito.

Quando se tratar de uma estrutura de dados, você deve deixar um valor a mais no tamanho da dimensão; caso contrário, terá resultados imprevisíveis. A seguir é apresentado um programa que faz a entrada e a saída dos dados de oito alunos.

```
/* Leitura e escrita para 8 alunos */

#include <stdio.h>

int main(void)
{

  struct CAD_ALUNO
    {
      char  NOME[41];
      float NOTA[4];
    };
```

```
  struct CAD_ALUNO ALUNO[8];

  int I, J;
  float NOTA_ALU;
  printf("\n\nCadastro de aluno\n\n");
  for (J = 0; J <= 7; J ++)
    {
      printf("Informe o nome do %do. Aluno ..: ", J + 1);
      fflush(stdin); fgets(ALUNO[J].NOME, 41, stdin);
        for (I = 0; I <= 3; I ++)
          {
            printf("Informe a %da. nota ..: ", I + 1);
             fflush(stdin); scanf("%f", &NOTA_ALU);
             ALUNO[J].NOTA[I] = NOTA_ALU;
           }
    }
    for (J = 0; J <= 7; J ++)
    {
      printf("\nNome .......: "); puts(ALUNO[J].NOME);
      for (I = 0; I <= 3; I ++)
        printf("Nota %da ...: %6.2f\n", I + 1, ALUNO[J].NOTA[I]);
    }
  printf("\n\n\n");
  return 0;
}
```

O laço de repetição da variável *J* controla o número de alunos da turma, no caso 8, e o laço de repetição da variável *I* controla o número de notas, até 4 por aluno. Para cada movimentação de mais um na variável *J,* existem quatro movimentações na variável *I*. Digite o programa anterior, gravando-o com o nome **classe.c**.

Observe também a utilização da função **fflush(stdin)**; logo após a leitura dos valores das notas dos alunos. Isso é necessário, pois o *buffer* de teclado fica carregado com o valor anterior, causando resultados imprevisíveis. A função **fflush()** tem por finalidade descarregar o *buffer* pendente. O argumento **stdin** (standard input) se refere à entrada-padrão, que no caso é o teclado. A função **fflush()** pertence à biblioteca **stdio.h**.

6.6.4 Programa Registro 4

Para demonstrar a utilização de programas com tabelas de dados heterogêneos, considere um programa que leia as quatro notas bimestrais de oito alunos, apresentando no final os dados dos alunos classificados por nome. A ordenação é feita com base no nome de cada aluno e quando estiver fora da ordem, os dados devem ser trocados de posição.

```
/* Leitura, ordenacao e escrita */

#include <stdio.h>
#include <string.h>

int main(void)
{

  struct CAD_ALUNO
    {
      char  NOME[40];
      float NOTA[4];
    };
  struct CAD_ALUNO ALUNO[8], X;

  int I, J;
  float NOTA_ALU;
  printf("\n\nCadastro de aluno\n\n");
  /* Entrada de dados */

  for (J = 0; J <= 7; J ++)
    {
      printf("Informe o nome do %do. Aluno ..: ", J + 1);
      fflush(stdin); fgets(ALUNO[J].NOME, 40, stdin);
      for (I = 0; I <= 3; I ++)
        {
          printf("Informe a %da. nota ..: ", I + 1);
          fflush(stdin); scanf("%f", &NOTA_ALU);
          ALUNO[J].NOTA[I] = NOTA_ALU;
        }
    }

  /* Classificacao dos dados */

  for (I = 0; I <= 6; I ++)
    for (J = I+1; J <= 7; J ++)
      if (strcmp(ALUNO[I].NOME, ALUNO[J].NOME) > 0)
        {
          X = ALUNO[I];
          ALUNO[I] = ALUNO[J];
          ALUNO[J] = X;
        }

  /* Saida dos dados */
    for (J = 0; J <= 7; J ++)
    {
      printf("\nNome ......: "); puts(ALUNO[J].NOME);
      for (I = 0; I <= 3; I ++)
        printf("Nota %da ...: %6.2f\n", I + 1, ALUNO[J].NOTA[I]);
    }
  printf("\n\n\n");
  return 0;
}
```

O programa anterior apresenta a comparação entre os nomes dos alunos. Sendo o nome do aluno atual maior que o nome do próximo aluno, é realizada a troca não só do nome, mas de todos os elementos que estão armazenados na tabela. Isso é possível uma vez que a variável *X* é do mesmo tipo da tabela *ALUNO*, no caso *CAD_ALUNO*. Digite o programa anterior, gravando-o com o nome **sortclas.c**.

Com a finalidade de fixar o aprendizado, é importante resolver o exercício de fixação 4.

Exercícios

1. Desenvolva em linguagem C os programas dos seguintes problemas utilizando matrizes de uma dimensão:
   a. Ler e escrever dez elementos numéricos inteiros de uma matriz A do tipo vetor.
   b. Ler oito elementos numéricos inteiros em uma matriz A do tipo vetor. Construir uma matriz B de mesma dimensão com os elementos da matriz A multiplicados por 3. Apresentar os elementos da matriz B.
   c. Ler uma matriz A do tipo vetor com 15 elementos numéricos inteiros. Construir uma matriz B de mesmo tipo, sendo cada elemento da matriz B a fatorial do elemento correspondente da matriz A. Apresentar os elementos da matriz B.
   d. Ler duas matrizes A e B do tipo vetor com 20 elementos numéricos inteiros. Construir uma matriz C, sendo cada elemento de C a subtração do elemento correspondente de A com B. Apresentar os elementos da matriz C.
   e. Ler duas matrizes A e B do tipo vetor com dez elementos numéricos inteiros cada uma. Construir uma matriz C, sendo esta a junção das matrizes A e B. Desta forma, C deve ter o dobro de elementos das matrizes A e B. Apresentar os elementos da matriz C.
   f. Ler duas matrizes do tipo vetor A com cinco elementos e B com dez elementos (valores numéricos inteiros). Construir uma matriz C, sendo esta a junção das duas outras matrizes. Desta forma, C deve ter a capacidade de armazenar 15 elementos. Apresentar os elementos da matriz C.

g. Ler 15 elementos numéricos inteiros de uma matriz A do tipo vetor. Construir uma matriz B de mesmo tipo, observando a seguinte lei de formação: todo elemento da matriz B deve ser o quadrado do elemento correspondente da matriz A. Apresentar os elementos das matrizes A e B dispostos lado a lado.

h. Ler 20 elementos numéricos inteiros para uma matriz A do tipo vetor e construir uma matriz B de mesma dimensão com os mesmos elementos de A. Eles devem estar invertidos, ou seja, o primeiro elemento de A passa a ser o último de B, o segundo elemento de A passa a ser o penúltimo de B e assim por diante. Apresentar os elementos das duas matrizes.

2. Desenvolva em linguagem C os programas dos seguintes problemas utilizando matrizes de duas dimensões:

   a. Ler duas matrizes A e B com valores inteiros cada uma de duas dimensões com cinco linhas e três colunas. Construir uma matriz C de mesma dimensão, a qual é formada pela soma dos elementos da matriz A com os elementos da matriz B. Apresentar os elementos da matriz C.

   b. Ler duas matrizes A e B, cada uma com uma dimensão para sete elementos numéricos inteiros. Construir uma matriz C de sete linhas com duas dimensões. A primeira coluna deve ser formada pelos elementos da matriz A e a segunda coluna pelos elementos da matriz B. Apresentar os elementos da matriz C.

   c. Ler 20 elementos numéricos reais para uma matriz A, considerando que essa matriz tenha o tamanho de quatro linhas por cinco colunas. Em seguida apresentar os valores lidos.

   d. Ler uma matriz A de uma dimensão com cinco elementos numéricos inteiros. Construir uma matriz B de duas dimensões com três colunas. A primeira coluna da matriz B será formada pelos elementos da matriz A somados com 5, a segunda coluna pelo valor do cálculo da fatorial de cada elemento correspondente da matriz A e a terceira e última coluna pelos quadrados dos elementos correspondentes da matriz A. Apresentar os elementos da matriz B.

   e. Ler duas matrizes A e B, cada uma de uma dimensão para seis elementos numéricos do tipo real. Construir uma matriz C de duas dimensões. A primeira coluna da matriz C deve ser formada pelos elementos da matriz A multiplicados por 2 e a segunda coluna pelos

elementos da matriz B subtraídos de 5. Apresentar os elementos da matriz C.

f. Ler duas matrizes A e B com valores reais cada uma de duas dimensões com quatro linhas e quatro colunas. Construir uma matriz C de mesma dimensão, a qual é formada pela subtração dos elementos da matriz A dos elementos da matriz B. Apresentar os valores da matriz C.

g. Ler 16 elementos numéricos reais para uma matriz A, considerando uma matriz com quatro linhas por quatro colunas. Em seguida apresentar os valores existentes na diagonal principal da matriz A.

h. Ler nove elementos numéricos reais para uma matriz A, considerando uma matriz com três linhas por três colunas. Em seguida apresentar os valores existentes na diagonal principal da matriz A multiplicados por 2 e os demais elementos multiplicados por 3.

3. Desenvolva em linguagem C os programas dos seguintes problemas utilizando pesquisa sequencial e ordenação de matrizes. Utilize em todos os programas a função "fflush()" antes da entrada de dados:

   a. Ler 12 elementos do tipo *string* para uma matriz A do tipo vetor que representem nomes pessoais. Colocar os nomes em ordem decrescente e apresentá-los de forma ordenada.

   b. Ler oito elementos numéricos inteiros para uma matriz A do tipo vetor. Construir uma matriz B de mesma dimensão e tipo com os elementos da matriz A multiplicados por 5. Apresentar a matriz B em ordem crescente. Após a apresentação de todos os elementos da matriz B, o programa deve disponibilizar um recurso de pesquisa para que o usuário possa pesquisar os elementos da matriz B.

   c. Ler uma matriz A do tipo vetor com 15 elementos numéricos inteiros. Construir uma matriz B de mesmo tipo, sendo cada elemento da matriz B o fatorial do elemento correspondente da matriz A. Apresentar os elementos da matriz B ordenados de forma crescente.

   d. Ler uma matriz A com 12 elementos numéricos reais e colocar os elementos em ordem decrescente. Depois ler uma matriz B com 12 elementos numéricos inteiros e colocar os elementos em ordem decrescente, construir uma matriz C, sendo cada elemento de C a soma dos elementos correspondentes das matrizes A e B. Colocar em ordem crescente a matriz C e apresentar os seus elementos.

e. Ler duas matrizes do tipo vetor para elementos do tipo literal (*string*). A matriz A deve possuir 12 nomes femininos e a matriz B 11 nomes masculinos. Construir uma matriz C, sendo esta a junção das matrizes A e B. Desta forma, C deve ter a capacidade de armazenar 23 elementos do tipo literal. Apresentar os nomes da matriz C em ordem crescente.

f. Ler 20 elementos numéricos reais de uma matriz A do tipo vetor. Construir uma matriz B de mesmo tipo, observando a seguinte lei de formação: todo elemento de B deve ser o cubo do elemento de A correspondente. Apresentar os elementos da matriz B por meio de saída controlada por pesquisa.

g. Ler 18 elementos numéricos reais de uma matriz A do tipo vetor e construir uma matriz B de mesma dimensão e tipo com os mesmos elementos correspondentes da matriz A acrescentados de mais 2. Montar o trecho de programa com pesquisa, para pesquisar e apresentar os elementos que estejam armazenados na matriz B.

h. Ler dez elementos do tipo literal que representem nomes de pessoas para uma matriz A do tipo vetor e construir uma matriz B de mesma dimensão com os mesmos elementos de A, os quais devem estar invertidos, ou seja, o primeiro elemento de A passa a ser o último de B, o segundo elemento de A passa a ser o penúltimo de B e assim por diante. Apresentar os elementos da matriz B.

4. Efetuar o cadastro sequencial de dez registros para uma agenda de endereços, nomes e telefones. Defina uma estrutura apropriada e construa um programa que por meio de um menu de seleção esteja capacitado a realizar:

   - o cadastro de todos os dados;
   - a classificação dos dados;
   - a pesquisa por nomes com a apresentação de todos os dados;
   - a listagem total apenas dos nomes.

   Cada uma destas operações deve ser controlada por um comando **if**. Não se esqueça de deixar uma opção reservada para a saída do usuário do sistema. Ao fazer o cadastro dos dez registros, peça a listagem total antes de executar a opção de classificação. Observe a ordem em que os dados foram entrados. Somente após esta observação solicite a ordenação dos dados e peça nova listagem geral.

7

capítulo

Funções e Suas Bibliotecas

Objetivos

Este capítulo traz informações sobre a programação estruturada por meio de funções. Aborda algumas funções (as principais) existentes nas bibliotecas do compilador GCC e aquelas criadas pelo próprio programador. Aborda, também, algumas bibliotecas de funções, refinamento sucessivo, estrutura de controle de múltipla escolha, passagens de parâmetro, ponteiros e protótipos de função.

Utiliza os comandos:

- *break*
- *default*
- *switch...case*

A função:

- *getchar*()

7.1 As Funções

A linguagem C é por natureza uma linguagem de programação estruturada. A maior parte de seus recursos é conseguida com a utilização de funções, como main(), printf(), scanf(), fgets(), puts(), strcmp(), strcpy(), estudadas anteriormente. A linguagem possui uma coleção de bibliotecas, tais como stdio.h e string.h, as quais possuem um grande conjunto de várias funções que podem ser usadas a qualquer momento. Uma função também pode ser desenvolvida pelo programador.

Função é um trecho independente de código de programa com atribuições bem definidas. Uma função (também pode ser denominada como sub-rotina) pode ser interna ou externa:

- É considerada função interna quando faz parte do compilador. O conjunto das funções internas forma a chamada biblioteca de funções, como, por exemplo, as bibliotecas-padrão **stdio.h** e **string.h** (demonstradas em exemplos de programas nos capítulos anteriores).
- É considerada função externa quando é desenvolvida e implementada por um programador com a finalidade de atender a uma necessidade em particular.

De forma geral, uma função sempre deve retornar algum tipo de valor como resposta de sua execução, segundo norma ISO da linguagem C, exceto quando uma função estiver sendo usada como um procedimento que é uma forma de sub-rotina, utilizada para executar trechos acessórios de código, que não possuem como ação devolver algum tipo de valor como resposta.

O recurso de função torna a programação com linguagem C algo bastante versátil, visto que:

- Em termos de portabilidade, trechos dos programas que não possam ser aproveitados em outras plataformas podem ser facilmente isolados e facilitar a migração do programa para outro computador ou mesmo compilador, alterando apenas as funções inativas.
- Em termos de modularidade, tem-se o programa dividido em vários módulos, e cada módulo desempenha uma função. Essa estratégia de programação facilita a manutenção dos programas construídos.

- O programador torna-se capaz de criar as suas próprias bibliotecas de funções pessoais, o que faz com que a programação torne-se mais eficiente, porque é possível aproveitar códigos de programa que já foram testados, os quais podem ser usados sem problema em novos programas.

No geral, problemas complexos exigem algoritmos complexos, mas sempre é possível dividir um problema grande em problemas menores. Cada parte menor tem um algoritmo mais simples, e é esse trecho menor que pode ser definido como sendo uma função.

Quando uma função é chamada por um programa principal, ela é executada e ao seu término, o controle de processamento retorna automaticamente para a primeira linha de instrução após a linha que fez a chamada.

A linguagem C permite que uma função assuma três tipos de comportamento: o primeiro quando a função é executada e não retorna valor (simulação da execução de um procedimento), o segundo quando uma função tem a capacidade de retornar apenas um valor e, por último, quando a função trabalha com passagem de parâmetros por valor e por referência.

7.2 Utilização de Bibliotecas

Uma biblioteca de função em linguagem C é o conjunto de rotinas prontas para serem usadas pelo programador. O compilador GCC possui um conjunto extenso de bibliotecas com diversas funções de suporte a linguagem. No entanto, além das bibliotecas existentes no compilador é possível fazer uso de bibliotecas de funções fornecidas por terceiros.

A seguir, são apresentadas algumas das bibliotecas existentes no Turbo C e uma rápida descrição.

Tabela 7.1 - Bibliotecas-padrão da Linguagem C (compilador GCC)

Biblioteca	Descrição
stdio.h	Essa biblioteca é a mais utilizada na programação em línguagem C, pois é a padrão, na qual estão embutidas as funções printf(), puts(), gets(), scanf(), entre outras.
math.h	Possui as funções matemáticas usadas pela linguagem. Encontram-se funções trigonométricas, hiperbólicas, exponenciais, logarítmicas, entre outras.
string.h	Esta possui as rotinas de tratamento de *strings* e caracteres, na qual se encontram as funções strcmp() e strcpy(), entre outras.

Biblioteca	Descrição
time.h	Essa biblioteca possui as funções de manipulação de data e hora do sistema.
stdlib.h	Possui um conjunto de funções que não se enquadra em outras categorias. As funções dessa biblioteca são conhecidas como "funções miscelâneas".

Para usar esse de uma biblioteca em Linguagem C, é necessário utilizar uma diretiva que seja colocada antes da declaração da principal função de trabalho da linguagem, denominada **main()**, como aconteceu nos programas anteriores, quando da utilização da cláusula **#include**.

Uma diretiva em linguagem C é uma instrução (cláusula) precedida do caractere tralha "**#**" (também conhecido pelos nomes antífen ou *hashtag*), que deve ser sempre escrito na primeira coluna de texto do código fonte. A instrução, após o símbolo tralha, é uma palavra reservada do dicionário do pré-processador para a chamada de uma biblioteca externa à linguagem.

O pré-processador é um programa que verifica o programa-fonte escrito, neste caso em linguagem C e realiza modificações nele. O pré-processador recebe este nome pelo fato de ser o primeiro programa a ser executado em um processo de compilação, fazendo com que o programa-fonte seja alterado, na forma de texto, antes de ser propriamente compilado, ou seja, sua tradução para o código-objeto (programa em linguagem de máquina) para depois ser definido como um programa executável para o computador.

7.3 Funções Definidas pelo Programador

A partir de uma noção de como utilizar as bibliotecas-padrão da linguagem C, será apresentado como fazer a criação de bibliotecas escritas pelo próprio programador, que serão desenvolvidas com os mesmos cuidados estudados e aplicados até o momento. A sintaxe de uma função de forma geral é definida como:

```
[<tipo>] <nome> ([<parâmetros>])
[<tipos de parâmetros>];
{
  [<variáveis (locais)>];
  [<instruções>];
}
```

em que:

<tipo>	- tipo de dado ao qual a função dará retorno;
<nome>	- o nome atribuído ao procedimento;
<parâmetros>	- uma informação opcional;
<tipos de parâmetros>	- quando existente, necessita declarar tipo;
<variáveis>	- lista de variáveis locais (opcionais);
<instruções>	- processamento do corpo da função.

7.3.1 Aplicação de Função em um Programa

Desenvolver um programa calculadora que apresente um menu de seleções no programa principal. Esse menu deve dar ao usuário a possibilidade de escolher uma entre quatro operações aritméticas. Escolhida a opção desejada, deve ser solicitada a entrada de dois números, e processada a operação, deve ser exibido o resultado.

Algoritmo

O programa deve ser um conjunto de cinco rotinas, sendo uma principal e quatro secundárias. A rotina principal controla as quatro rotinas secundárias que efetuam o pedido de leitura de dois valores, fazem a operação e apresentam o resultado obtido.

Tendo uma ideia da estrutura geral do programa, é escrito em separado cada algoritmo com os seus detalhes de operação. Primeiramente o programa principal e depois as outras rotinas.

Programa Principal

1. Apresentar um menu de seleção com cinco opções:
    a. Adição
    b. Subtração
    c. Multiplicação
    d. Divisão
    e. Fim de Programa

2. Ao ser selecionado um valor, a rotina correspondente deve ser executada.
3. Ao escolher o valor 5, o programa deve ser encerrado.

Rotina 1 - Adição

1. Ler dois valores, no caso variáveis *A* e *B*.
2. Efetuar a soma das variáveis *A* e *B,* implicando o seu resultado na variável *R*.
3. Apresentar o valor da variável *R*.
4. Voltar ao programa principal.

Rotina 2 - Subtração

1. Ler dois valores, no caso variáveis *A* e *B*.
2. Efetuar a subtração das variáveis *A* e *B,* implicando o seu resultado na variável *R*.
3. Apresentar o valor da variável *R*.
4. Voltar ao programa principal.

Rotina 3 - Multiplicação

1. Ler dois valores, no caso variáveis *A* e *B*.
2. Efetuar a multiplicação das variáveis *A* e *B,* implicando o seu resultado na variável *R*.
3. Apresentar o valor da variável *R*.
4. Voltar ao programa principal.

Rotina 4 - Divisão

1. Ler dois valores, no caso variáveis *A* e *B*.
2. Efetuar a divisão das variáveis *A* e *B,* implicando o seu resultado na variável *R*.
3. Apresentar o valor da variável *R*.
4. Voltar ao programa principal.

Em cada rotina utilizam-se as mesmas variáveis, porém elas não são executadas ao mesmo tempo para todas as operações. São utilizadas em separado e somente para a rotina escolhida.

Programa

```
/* Programa Calculadora */

#include <stdio.h>

float R, A, B;

int rotadicao(void);
int rotsubtracao(void);
int rotmultiplicacao(void);
int rotdivisao(void);

int main(void)
{
  char TECLA;
  int OPCAO = 0;
  while (OPCAO != 5)
    {
      printf("\n\nMenu Principal\n");
      printf("--------------\n\n");
      printf("1 - Adicao\n");
      printf("2 - Subtracao\n");
      printf("3 - Multiplicacao\n");
      printf("4 - Divisao\n");
      printf("5 - Fim de Programa\n\n");
      printf("Escolha uma opcao: "); fflush(stdin); scanf("%d", &OPCAO);
      if (OPCAO != 5)
        {
          if (OPCAO == 1)
            rotadicao();
          if (OPCAO == 2)
            rotsubtracao();
          if (OPCAO == 3)
           rotmultiplicacao();
          if (OPCAO == 4)
            rotdivisao();
        }
    }
  return 0;
}

int rotadicao(void)
{
  printf("\n\nRotina de Soma\n");
  printf("--------------\n\n");
  printf("Entre um valor para A: ");
  fflush(stdin); scanf("%f", &A);
  printf("Entre um valor para B: ");
  fflush(stdin); scanf("%f", &B);
  R = A + B;
  printf("\nO resultado entre A e B = %6.2f\n", R);
  printf("\nTecle <Enter> para acessar o menu: ");
  getchar() + scanf("Enter");
  return 0;
```

```
}

int rotsubtracao(void)
{
  printf("\n\nRotina de Subtracao\n");
  printf("--------------------\n\n");
  printf("Entre um valor para A: ");
  fflush(stdin); scanf("%f", &A);
  printf("Entre um valor para B: ");
  fflush(stdin); scanf("%f", &B);
  R = A - B;
  printf("\nO resultado entre A e B = %6.2f\n", R);
  printf("\nTecle <Enter> para acessar o menu: ");
  getchar() + scanf("Enter");
  return 0;

}

int rotmultiplicacao(void)
{
  printf("\n\nRotina de Multiplicacao\n");
  printf("-----------------------\n\n");
  printf("Entre um valor para A: ");
  fflush(stdin); scanf("%f", &A);
  printf("Entre um valor para B: ");
  fflush(stdin); scanf("%f", &B);
  R = A * B;
  printf("\nO resultado entre A e B = %6.2f\n", R);
  printf("\nTecle <Enter> para acessar o menu: ");
  getchar() + scanf("Enter");
  return 0;
}

int rotdivisao(void)
{
  printf("\n\nRotina de Divisao\n");
  printf("-----------------\n\n");
  printf("Entre um valor para A: ");
  fflush(stdin); scanf("%f", &A);
  printf("Entre um valor para B: ");
  fflush(stdin); scanf("%f", &B);
  if (B == 0)
    printf("\nErro de divisao\n");
  else
    {
      R = A / B;
      printf("\nO resultado entre A e B = %6.2f\n", R);
    }
  printf("\nTecle <Enter> para acessar o menu: ");
  getchar() + scanf("Enter");
  return 0;
}
```

Abra uma janela em branco, digite o programa anterior e grave-o com o nome **calcula1.c**.

Note a definição inicial dos nomes das funções criadas manualmente antes da função **main()**. Essa ação estabelece os protótipos das funções a serem utilizadas no programa.

Os protótipos de função fazem com que haja uma verificação mais detalhada dos tipos em uso, desta forma o compilador pode encontrar e apresentar quaisquer conversões de tipos que sejam ilegais. Estabelecer os protótipos de funções é uma exigência da norma ANSI para a linguagem C, mesmo quando o compilador diretamente não exige essa definição.

As quatro funções criadas **rotsoma()**, **rotsubtracao()**, **rotmultiplicacao()** e **rotdivisao()** são funções do tipo **int**, as quais não retornam para o programa principal nenhum tipo de valor, por essa razão usa-se a instrução **return 0**. Essa estratégia simula na linguagem C o uso de procedimentos encontrados em outras linguagens de programação. Há programadores de linguagem C que preferem definir funções que simulam procedimentos como funções do tipo **void**, com o uso da instrução **return**, sem um valor de retorno. Confira, a seguir, uma tabela com as duas abordagens que podem ser utilizadas. Atente para os pontos em negrito.

Tabela 7.2 - Exemplos de simulação de procedimentos em linguagem C

Forma tradicional com tipo int	Forma moderna com tipo void

`int rotexemplo(void)`
`{`
`  fflush(stdin); scanf("%f", &A);`
`  fflush(stdin); scanf("%f", &B);`
`  R = A + B;`
`  printf("%6.2f\n", R);`
`  return 0;`
`}`
	`void rotexemplo(void)`
	`{`
	`  fflush(stdin); scanf("%f", &A);`
	`  fflush(stdin); scanf("%f", &B);`
	`  R = A + B;`
	`  printf("%6.2f\n", R);`
	`  return;`
	`}`

Cada função de cálculo definida no programa calculadora de forma manual tem por finalidade ler dois valores, processar o cálculo indicado e apresentar o respectivo resultado obtido.

As variáveis *A*, *B* e *R* são globais (variáveis públicas), e para tal condição ficam acima da função **main()**. São reconhecidas pelas funções que estão subordinadas à função **main()**. Já a variável *OPCAO* fica dentro da função **main()**, portanto é local, pois somente é válida dentro dessa função.

No final de cada rotina de cálculo são utilizadas antes da instrução **return 0** as seguintes linhas de código:

```
printf("\nTecle <Enter> para acessar o menu: ");
getchar() + scanf("Enter");
```

As linhas de código em uso, respectivamente, apresentam a mensagem **"Tecle <Enter> para acessar o menu: "** e usam a função **getchar()** para interromper o processamento e aguardar o acionamento de uma das teclas do teclado, ou seja, de capturar o acionamento do teclado. Neste caso, para capturar a tecla **<Enter>** usa-se junto com a função **getchar()** a concatenação da função **scanf()** apenas com um *string* (que deve possuir no mínimo dois caracteres). Neste caso foi escolhido arbitrariamente o texto **Enter**.

Observação

Em algumas das edições anteriores deste trabalho foi utilizada para capturar o acionamento da tecla **<Enter>** a linha de comando **while ((TECLA = getchar()) != '\r')** com a variável **TECLSA** do tipo **char**. A partir da versão **3.4.x** do **GCC**, essa forma não surte o efeito esperado, sendo necessário substituí-la por uma forma alternativa. Neste caso, pela instrução **getchar() + scanf("Enter");**.

7.3.2 Estrutura de Controle com Múltipla Escolha

O programa anterior apresentou o modo de seleção das funções feito com instruções **if**. Se o programa possuir um menu com 15 opções, devem ser definidas 15 instruções **if** para verificar a escolha do operador. Apesar de a forma apresentada ser válida, ela pode ser simplificada com a utilização da instrução **switch**, que possui a sintaxe:

```
switch <variável>
  {
    case <opção1> : <operação1>;         break;
    case <opção2> : <operação2>;         break;
    case <opçãoN> : <operaçãoN>;         break;
    default       : <operação default>; break;
  }
```

em que:

<variável> - o nome da variável a ser controlada na decisão;

<opção> - o conteúdo da variável a ser verificado;

<operação> - a execução de alguma instrução específica.

Observação

A instrução **default** de **switch** e qualquer operação a ela subordinada são opcionais, não caracterizando uma obrigatoriedade o seu uso. A instrução **break** tem a finalidade de desviar o processamento para fora do comando **switch**. Isso é necessário, uma vez que após o processamento da função, ele retorna para a primeira instrução após sua chamada, que neste caso será **break**.

A rotina principal do programa calculadora pode ser escrita com a instrução **switch** quando o operador escolher uma opção. O programa principal ficou menor em número de linhas. Faça as devidas alterações e grave o programa com o nome **calcula2.c**.

```
/* Programa Calculadora */

#include <stdio.h>

float R, A, B;
int rotadicao(void);
int rotsubtracao(void);
int rotmultiplicacao(void);
int rotdivisao(void);


int main(void)
{
  int OPCAO = 0;
  while (OPCAO != 5)
    {
      printf("\n\nMenu Principal\n");
      printf("--------------\n\n");
      printf("1 - Adicao\n");
      printf("2 - Subtracao\n");
      printf("3 - Multiplicacao\n");
      printf("4 - Divisao\n");
      printf("5 - Fim de Programa\n\n");
      printf("Escolha uma opcao: "); fflush(stdin); scanf("%d",
&OPCAO);
      if (OPCAO != 5)
        {
          switch (OPCAO)
            {
              case 1  : rotadicao();        break;
              case 2  : rotsubtracao();     break;
              case 3  : rotmultiplicacao(); break;

              case 4  : rotdivisao();       break;
              default : printf("Opcao invalida - Tecle <Enter>");
                        getchar() + scanf("Enter");
                        break;
            }
```

```
        }
    }
  return 0;
}

int rotadicao(void)
{
  printf("\n\nRotina de Soma\n");
  printf("--------------\n\n");
  printf("Entre um valor para A: ");
  fflush(stdin); scanf("%f", &A);
  printf("Entre um valor para B: ");
  fflush(stdin); scanf("%f", &B);
  R = A + B;
  printf("\nO resultado entre A e B = %6.2f\n", R);
  printf("\nTecle <Enter> para acessar o menu: ");
  getchar() + scanf("Enter");
  return 0;
}

int rotsubtracao(void)
{
  printf("\n\nRotina de Subtracao\n");
  printf("-------------------\n\n");
  printf("Entre um valor para A: ");
  fflush(stdin); scanf("%f", &A);
  printf("Entre um valor para B: ");
  fflush(stdin); scanf("%f", &B);
  R = A - B;
  printf("\nO resultado entre A e B = %6.2f\n", R);
  printf("\nTecle <Enter> para acessar o menu: ");
  getchar() + scanf("Enter");
  return 0;

}
int rotmultiplicacao(void)
{
  printf("\n\nRotina de Multiplicacao\n");
  printf("-----------------------\n\n");
  printf("Entre um valor para A: ");
  fflush(stdin); scanf("%f", &A);
  printf("Entre um valor para B: ");
  fflush(stdin); scanf("%f", &B);
  R = A * B;
  printf("\nO resultado entre A e B = %6.2f\n", R);
  printf("\nTecle <Enter> para acessar o menu: ");
  getchar() + scanf("Enter");
  return 0;
}

int rotdivisao(void)
{
  printf("\n\nRotina de Divisao\n");
  printf("-----------------\n\n");
  printf("Entre um valor para A: ");
  fflush(stdin); scanf("%f", &A);
```

```
  printf("Entre um valor para B: ");
  fflush(stdin); scanf("%f", &B);
  if (B == 0)
    printf("\nErro de divisao\n");
  else
    {
      R = A / B;
      printf("\nO resultado entre A e B = %6.2f\n", R);
    }
  printf("\nTecle <Enter> para acessar o menu: ");
  getchar() + scanf("Enter");
  return 0;
}
```

Antes de utilizar a instrução **switch**, verifica-se com a instrução **if...then** se o valor da opção informado é realmente diferente de 5. Sendo, será verificado se é 1, 2, 3 ou 4; não sendo nenhum dos valores válidos, a instrução **default** de **switch** apresenta a mensagem **'Opcao invalida - Tecle algo'**, informando que a tentativa foi inválida.

7.3.3 Refinamento Sucessivo

É uma técnica de programação que possibilita dividir uma rotina em outras rotinas. Deve ser aplicado com muito critério para que o programa a ser construído não se torne desestruturado e difícil de ser compreendido.

O programa calculadora apresentado anteriormente permite que seja aplicada essa técnica, pois nas quatro sub-rotinas de cálculo existem instruções que realizam as mesmas tarefas. Por exemplo, a entrada e a saída são realizadas com as mesmas variáveis. Veja que as variáveis *A*, *B* e *R* são usa-das quatro vezes, uma em cada sub-rotina.

A solução é usar as variáveis *A*, *B* e *R* como globais e construir mais duas sub--rotinas, sendo uma para entrada e outra para saída. As quatro sub-rotinas atuais são diminuídas em número de linhas, pois tudo o que se repete nas sub-rotinas será retirado. Desta forma, as variáveis *A*, *B* e *R* serão definidas como globais.

Serão implementadas duas novas rotinas, uma para a entrada dos valores e outra para a saída do resultado da operação. A seguir é apresentada a nova versão do programa calculadora, observe no código a definição das funções **entrada** e **saída**, sendo do tipo **void**.

```
/* Programa Calculadora */

#include <stdio.h>

float R, A, B;

int rotadicao(void);
int rotsubtracao(void);
int rotmultiplicacao(void);
int rotdivisao(void);

void entrada(void);
void saida(void);

int main(void)
{
  int OPCAO = 0;
  while (OPCAO != 5)
    {
      printf("\n\nMenu Principal\n");
      printf("--------------\n\n");
      printf("1 - Adicao\n");
      printf("2 - Subtracao\n");
      printf("3 - Multiplicacao\n");
      printf("4 - Divisao\n");
      printf("5 - Fim de Programa\n\n");
      printf("Escolha uma opcao: "); fflush(stdin); scanf("%d", &OPCAO);
      if (OPCAO != 5)
        {
          switch (OPCAO)
            {
              case 1  : rotadicao();        break;
              case 2  : rotsubtracao();     break;
              case 3  : rotmultiplicacao(); break;
              case 4  : rotdivisao();       break;
              default : printf("Opcao invalida - Tecle <Enter>");
                        getchar() + scanf("Enter");
                        break;
            }
        }
    }
  return 0;
}

void entrada(void)
{
  printf("Entre um valor para A: ");
  fflush(stdin); scanf("%f", &A);
  printf("Entre um valor para B: ");
  fflush(stdin); scanf("%f", &B);
  return;
}

void saida(void)
{
  printf("\nO resultado entre A e B = %6.2f\n", R);
```

```
  printf("\nTecle <Enter> para acessar o menu: ");
  getchar() + scanf("Enter");
  return;
}
int rotadicao(void)
{
  printf("\n\nRotina de Soma\n");
  printf("--------------\n\n");
  entrada();
  R = A + B;
  saida();
  return 0;
}

int rotsubtracao(void)
{
  printf("\n\nRotina de Subtracao\n");
  printf("-------------------\n\n");
  entrada();
  R = A - B;
  saida();
  return 0;
}

int rotmultiplicacao(void)
{
  printf("\n\nRotina de Multiplicacao\n");
  printf("-----------------------\n\n");
  entrada();
  R = A * B;
  saida();
  return 0;
}

int rotdivisao(void)
{
  printf("\n\nRotina de Divisao\n");
  printf("-----------------\n\n");
  entrada();
  if (B == 0)
    {
      printf("\nErro de divisao\n");
      printf("\nTecle <Enter> para acessar o menu: ");
      getchar() + scanf("Enter");
    }
  else
    {
      R = A / B;
      saida();
    }
  return 0;
}
```

Escreva a nova versão do programa, gravando-o com o nome **calcula3.c**. Ao executar o programa, você nota que, em relação às versões anteriores, ele executa as mesmas funções, não existindo nenhuma diferença do ponto de vista operacional. A diferença está na construção lógica e na estrutura do programa.

Note que as funções **entrada()** e **saída()** estão sendo definidas como funções que retornam valor do tipo **void**, ou seja, funções que não retornam nada. Daí o uso no final das funções da instrução **return** sem nenhum valor definido para retorno. Como informado anteriormente esta é uma forma de simular na linguagem C o uso de sub-rotinas do tipo procedimento. Uma sub-rotina que é chamada para a execução de alguma tarefa e após a sua execução não tem a responsabilidade de retornar algum valor para o trecho de programa que a chamou.

7.4 Utilização de Parâmetros

Os parâmetros em uma função têm por finalidade a comunicação entre rotinas de programa. Muito útil quando se trabalha apenas com variáveis locais, que são mais vantajosas por ocuparem um espaço menor em memória do que as variáveis globais. As variáveis locais só ocupam espaço em memória quando estão sendo utilizadas.

Os parâmetros são um ponto de comunicação bidirecional entre uma sub-rotina e o programa principal, ou com outra rotina hierarquicamente de nível mais alto. É possível passar valores de uma sub-rotina (função) ou rotina chamadora a outra sub-rotina e vice-versa. Isso se chama passagem de parâmetro.

7.4.1 Passagem de Parâmetro por Valor

A passagem de parâmetro por valor funciona apenas como um mecanismo de entrada de valor para uma determinada função. O processamento é executado somente dentro da função chamada, ficando o resultado obtido "preso" na função e desconhecido pelo programa chamador. Este é o tipo de função que não retorna valor, por esta razão pode ser definida como uma função do tipo **void** sem retorno de valor ou uma função do tipo **int** com retorno de valor **0** (zero). O exemplo a seguir apresenta um programa com passagem de parâmetro por valor:

```
/* Fatorial sem retorno de valor */

#include <stdio.h>

void fatorial(int N);

int main(void)
{
  int LIMITE;
  char TECLA;
  printf("Calculo de fatorial\n\n");
  printf("Qual fatorial: ");
  fflush(stdin); scanf("%d", &LIMITE);
  fatorial(LIMITE);
  printf("\nTecle <Enter> para finalizar: ");
  getchar() + scanf("Enter");
  return 0;
}

void fatorial(int N)
{
  int I, FAT;
  FAT = 1;
  for (I = 1; I <= N; I ++)
    FAT *= I;
  printf("\nFatorial de %d = a: %d\n", N, FAT);
  return;
}
```

Escreva o programa e grave-o com o nome **fator1.c**. É indicada no programa a passagem de parâmetro por valor. No caso, a variável *N* da função **fatorial** recebe o valor fornecido pela variável *LIMITE*, o qual estabelece o número de vezes que o laço de repetição deve ser executado.

Na função **fatorial()** encontra-se a variável *FAT* utilizada para calcular o valor da fatorial dentro do laço. Ao término do laço de repetição o valor calculado é apresentado pela variável *FAT*, que somente é válida na sua função, por isso o valor dessa variável fica "preso" em rotina. A passagem de parâmetro por valor é utilizada somente para a entrada de um determinado valor em uma sub-rotina.

Anteriormente, quando se estudou função neste capítulo, foi informado que uma função sempre retorna apenas um valor. Também foram vistas funções cujos valores de retorno não foram definidos, pois sempre foram indicados como retorno zero (quando da utilização do comando **return 0**, retorno zero

ou apenas **return** quando a função é definida como do tipo de retorno **void**. O retorno de um valor é feito pelo comando **return** e devolvido para o nome da função. O comando **return** pode ser escrito das seguintes formas:

`return <expressão>;` - retorna o valor da expressão;

`return;` - não retorna valor.

Como exemplo de uso de função com passagem de parâmetro por valor com retorno considere o programa seguinte:

```
/* Fatorial com retorno de valor */

#include <stdio.h>

int fatorial(int N);

int main(void)
{
  int LIMITE;
  char TECLA;
  printf("Calculo de fatorial\n\n");
  printf("Qual fatorial: ");
  fflush(stdin); scanf("%d", &LIMITE);
  printf("\nFatorial de %d = a: %d\n", LIMITE, fatorial(LIMITE));
  printf("\nTecle <Enter> para finalizar: ");
  getchar() + scanf("Enter");
  return 0;
}

int fatorial(int N)
{
  int I, FAT;
  FAT = 1;
  for (I = 1; I <= N; I ++)
    FAT *= I;
  return FAT;
}
```

Neste exemplo é indicado o uso da instrução **return** da função **fatorial(int** *N* **)** com retorno de *FAT*, a qual faz o retorno do valor da fatorial para fora da função, devolvendo o valor para a rotina chamadora. Digite o programa anterior e grave-o com o nome **fator2.c**.

Compare o código da função **fatorial()** dos dois programas. No primeiro exemplo a saída da informação (resultado do cálculo da fatorial) está colocada na função **fatorial()**, a qual forçosamente retorna o valor zero.

No segundo exemplo a saída do resultado do cálculo da fatorial é conseguida fora da função **fatorial()**, a qual faz o retorno pelo comando **return(**FAT**)**, devolvendo o valor para a função **fatorial(**LIMITE**)**, quando do uso da

linha de comando **printf(**"\nFatorial de %d equivale a: %d\n", *LIMITE*, **fatorial**(*LIMITE*)**)**.

Após aprender como fazer o retorno de um valor, é implementada no programa calculadora uma função que deve retornar o valor do cálculo solicitado. Veja a seguir um exemplo dessa função.

```
float calculo(float X, float Y, char OPERADOR)
{
  float RESULTADO;
  switch (OPERADOR)
    {
      case '+' : RESULTADO = X + Y; break;
      case '-' : RESULTADO = X - Y; break;
      case '*' : RESULTADO = X * Y; break;
      case '/' : RESULTADO = X / Y; break;
    }
  return RESULTADO;
}
```

Em seguida é apresentado o programa calculadora com as novas modificações após ter acrescentado os novos detalhes. Grave o programa com o nome **calcula4.c**.

```
/* Programa Calculadora */

#include <stdio.h>

char TECLA;
float R, A, B;

int rotadicao(void);
int rotsubtracao(void);
int rotmultiplicacao(void);
int rotdivisao(void);
void antrada(void);
void saida(void);
float calculo(float X, float Y, char OPERADOR);

int main(void)
{
  int OPCAO = 0;
  while (OPCAO != 5)
    {
      printf("\n\nMenu Principal\n");
      printf("--------------\n\n");
      printf("1 - Adicao\n");
      printf("2 - Subtracao\n");
      printf("3 - Multiplicacao\n");
      printf("4 - Divisao\n");
      printf("5 - Fim de Programa\n\n");
      printf("Escolha uma opcao: "); fflush(stdin); scanf("%d", &OPCAO);
      if (OPCAO != 5)
        {
```

```
          switch (OPCAO)
            {
              case 1  : rotadicao();        break;
              case 2  : rotsubtracao();     break;
              case 3  : rotmultiplicacao(); break;
              case 4  : rotdivisao();       break;
              default : printf("Opcao invalida - Tecle <Enter>");
                        getchar() + scanf("Enter");
                        break;
            }
        }
    }
  return 0;
}
void entrada(void)
{
  printf("Entre um valor para A: ");
  fflush(stdin); scanf("%f", &A);
  printf("Entre um valor para B: ");
  fflush(stdin); scanf("%f", &B);
  return;
}

void saida(void)
{
  printf("\nO resultado entre A e B = %6.2f\n", R);
  printf("\nTecle <Enter> para acessar o menu: ");
  getchar() + scanf("Enter");
  return;
}

float calculo(float X, float Y, char OPERADOR)
{
  float RESULTADO;
  switch (OPERADOR)
    {
      case '+' : RESULTADO = X + Y; break;
      case '-' : RESULTADO = X - Y; break;
      case '*' : RESULTADO = X * Y; break;
      case '/' : RESULTADO = X / Y; break;
    }
  return RESULTADO;
}

int rotadicao(void)
{
  printf("\n\nRotina de Soma\n");
  printf("--------------\n\n");
  entrada();
  R = calculo(A, B, '+');
  saida();
  return 0;
}

int rotsubtracao(void)
{
```

```
  printf("\n\nRotina de Subtracao\n");
  printf("-------------------\n\n");
  entrada();
  R = calculo(A, B, '-');
  saida();
  return 0;
}

int rotmultiplicacao(void)
{
  printf("\n\nRotina de Multiplicacao\n");
  printf("-----------------------\n\n");
  entrada();
  R = calculo(A, B, '*');
  saida();
  return 0;
}

int rotdivisao(void)
{
  printf("\n\nRotina de Divisao\n");
  printf("-----------------\n\n");
  entrada();
  if (B == 0)
    {
      printf("\nErro de divisao\n");
      printf("\nTecle <Enter> para acessar o menu: ");
      getchar() + scanf("Enter");
    }
  else
    {
      R = A / B;
      saida();
    }
  return 0;
}
```

Em cada rotina de cálculo, as linhas que possuíam a fórmula *R* = *A* + *B*, por exemplo, foram substituídas pela função *R* = **formula(***A*, *B*, '+'**);**. Outro detalhe é a colocação do protótipo da função de cálculo antes da função **main()**. O protótipo de uma função é exigido por alguns compiladores para reconhecimento e validação da função pelo programa.

Outro detalhe é o fato de trabalhar com tipos diferentes de dados na função cálculo. Por esta razão cada dado dentro dos parênteses é precedido de seu tipo.

7.4.2 Passagem de Parâmetro por Referência

A passagem de parâmetro por referência funciona como um mecanismo de entrada e saída de valor para uma função junto aos seus parâmetros. A

alteração é devolvida para a rotina chamadora por meio dos parâmetros e não da função propriamente dita.

O exemplo anterior fez o retorno de um valor, mas não utilizou passagem de parâmetro por referência. Quando se utiliza o comando **return**, somente é possível retornar um valor para a função. Havendo a necessidade de retornar mais de um valor, utiliza-se a passagem de parâmetro por referência.

Para demonstrar funções com passagem de parâmetro por referência, considere a leitura de dez valores inteiros em uma matriz de uma dimensão (vetor) e coloque-os em ordem crescente. A ordenação é realizada com uma função apropriada para este fim.

1. Criar uma função que compare e faça a troca dos elementos.
2. Para a troca estabelecer três variáveis, sendo *X* como auxiliar e *A*, *B* como valores.
3. Para verificação de condição comparar com duas variáveis *A* e *B*.
4. Pelo fato de as sub-rotinas retornarem uma resposta, seus parâmetros são por referência.

O programa seguinte especifica as funções de troca e de verificação da condição para a troca de valores, com a passagem de parâmetro por referência. Para fazer a passagem de parâmetro por referência, utiliza-se como referência o endereço da variável por meio do operador **&** na função principal ou na função chamadora.

Na função de ordenação, são feitas referências a ponteiros pelo símbolo ***** (asterisco). Para cada variável utilizada existe um endereço a ela relacionado, indicando a posição de memória com o início do armazenamento dos bytes relativos ao valor dessa variável.

A linguagem C permite que em uma variável seja armazenado o endereço de outra variável. Para que isso seja possível, é necessário trabalhar com uma variável como sendo um ponteiro, um apontador. A declaração de um ponteiro é semelhante à de uma variável simples, tendo como diferencial o asterisco antes da variável.

Como o programa usa a passagem de parâmetro por referência, os valores dos endereços de &*VETOR*[*I*] e &*VETOR*[*J*] são passados para os apontadores **A* e **B,* respectivamente, que após a troca devolvem os valores trocados.

```
/* Passagem de parametro por referencia */

#include <stdio.h>

int troca(int *A, int *B);

int main(void)
{
  int VETOR[9], I, J;
  printf("\n\nOrdenacao de vetor\n\n");

  /* Entrada de dados */

  for (I = 0; I <= 9; I ++)
    {
      printf("Entre o %2d. elemento: ", I + 1);
      fflush(stdin); scanf("%d", &VETOR[I]);
    }

  /* Ordenacao do vetor */

  for (I = 0; I <= 8; I ++)
    for (J = I + 1; J <= 9; J ++)
      troca(&VETOR[I], &VETOR[J]);
  /* Apresentacao do vetor ordenado */

  printf("\n");
  for (I = 0; I <= 9; I ++)
    printf("Agora o %2do. Elemento e': %d\n", I + 1, VETOR[I]);
  printf("\nTecle <Enter> para finalizar: ");
  getchar() + scanf("Enter");
  return 0;
}

int troca(int *A, int *B)

/* Definicao dos ponteiros que receberao os valores dos vetores */
{
  int X;
  if (*A > *B)
    {
      X  = *A;
      *A = *B;
      *B =  X;
    }
  return 0;
}
```

O retorno da função **troca()** é zero, mas em contrapartida os valores dos apontadores "*A" e "*B" são alterados internamente e devolvidos pelos mesmos apontadores "*A" e "*B", utilizados para a entrada dos valores na função. Em seguida grave o programa com o nome **ordnum.c**.

7.5 Biblioteca Definida pelo Programador

Até o momento, foram utilizadas algumas bibliotecas internas da linguagem C, que são fornecidas com o programa compilador. Além desse uso, é possível ao programador desenvolver as próprias bibliotecas, com recursos customizados para atender às suas necessidades.

Para demonstrar essa possibilidade, ao longo deste tópico, será desenvolvida uma biblioteca que fornece recursos de limpeza de tela, colorização do texto escrito e do fundo de tela e controle de posicionamento do cursor para o modo console de operação.

Serão estudados, nesta parte, os recursos-padrão: modo *Terminal ANSI* e modo *Windows API*, comentados a seguir.

7.5.1 Modo Terminal ANSI

O modo *Terminal ANSI* oferece controle padronizado das ações vinculadas ao uso do terminal de vídeo e teclado. Terminal é um equipamento que oferece ao usuário uma interface de acesso ao computador, como o teclado e o monitor de vídeo.

O uso do equipamento chamado *terminal* é comum em computadores de grande porte, em que se tem um console de acesso às funções do computador que esteja distante do terminal, diferentemente de um mini ou microcomputador cujas funções de entrada e saída em vídeo (terminal) e de processamento (CPU) são conectadas próximas umas das outras, quando não no mesmo console, como é o caso dos *notebooks*.

O modelo mais famoso de *terminal* para computadores de grande e médio porte foi o VT100, da empresa DEC. Esse terminal foi o primeiro a possuir um microprocessador e um dos primeiros a incorporar o controle por sequência de *escape ANSI*. Como a sequência de *escape* foi uma proposta de padronização do órgão ANSI (*American National Standards Institute* - Instituto Nacional Americano de Padrões) para os recursos de acesso a terminais da época, o modelo VT100 acabou por prevalecer até os dias atuais em diversos sistemas operacionais, e seus recursos são disponibilizados como padrão nos sistemas operacionais UNIX e nos muitos sabores Linux. Em relação aos sistemas operacionais da empresa Microsoft, o recurso está disponível, de forma indireta, nos programas MS-DOS e em todas as versões do Windows de 32 *bits*.

Para usar os recursos do modo Terminal ANSI nas versões de 64 *bits* do Windows, são necessárias ferramentas desenvolvidas por terceiros, como é o caso do programa **ANSICON**, ou de recursos equivalentes ao modo de operação da Windows API, como apresentado mais adiante.

Existem diversas ações que podem ser executadas por um programa de computador no modo console de texto, para alterar parte do comportamento do teclado e do monitor de vídeo. Entre essas ações, é possível alterar a forma de exibição do monitor de vídeo, controlar o local onde o cursor será exibido e redefinir algumas teclas do teclado. Essas ações podem ser controladas por meio dos códigos de *escape*.

Os procedimentos descritos nesta obra são facilmente acessíveis em diversos computadores, exceto os microcomputadores da família IBM-PC, a menos que usem o Linux ou outro sistema operacional no padrão POSIX (*Portable Operating System Interface*).

O problema é maior quando um microcomputador da família IBM-PC é usado com sistemas Microsoft (qualquer um deles), pois estes não dão suporte direto ao modo de operação do Terminal ANSI. No sentido de minimizar essa ocorrência, a Microsoft sempre manteve, em seus sistemas operacionais de 16 ou 32 *bits*, o arquivo ANSI.SYS que, para ser usado, necessita ser configurado manualmente. Mas o mesmo não ocorre nos sistemas de 64 *bits* da empresa.

O compilador GCC não possui nenhum arquivo de cabeçalho específico para essa finalidade. Qualquer controle específico deve ser desenvolvido pelo próprio programador. Por isso, a proposta deste tópico é criar uma biblioteca externa com tal funcionalidade.

Para usar os recursos do modo Terminal ANSI nos sistemas operacionais da Microsoft, disponíveis no *driver* ANSI.SYS, é necessário fazer com que o *driver* seja executado quando do processo de inicialização do sistema, principalmente com o uso de MS-DOS e MS-Windows 9x/ME. Nas demais versões de 32 *bits* dos sistemas operacionais Windows, não é necessário reinicializar o computador; apenas são realizados o fechamento da janela de linha de comando (*prompt de comando*) atual e sua abertura imediata após o procedimento de configuração.

No MS-DOS até a versão 5.22, o *driver* ANSI.SYS ficava normalmente no diretório **\DOS**; nos sistemas operacionais MS-Windows 9x/ME, encontrava-se

na pasta **\WINDOWS\COMMAND**. Nos sistemas operacionais MS-Windows NT/2000/XP/Vista/7/8 de 32 *bits*, o *driver* se encontra na pasta **\WINDOWS\SYSTEM32**. As versões de 64 *bits* dos sistemas operacionais Microsoft Windows (qualquer versão) não disponibilizam esse *driver*, pois operam segundo um padrão próprio da Microsoft, baseado na Windows API, como será observado mais adiante, exceto que se instale e utilize o programa **ANSICON**, que adiciona aos sistemas operacionais de 64 *bits*, e mesmo aos de 32 *bits*, os recursos compatíveis ao modo Terminal ANSI. No entanto, o programa ANSICON pode ocasionar efeitos indesejados no momento da compilação de alguns programas. Se isso acontecer, opte por usar o modo Windows API.

Para instalar e executar o *driver* ANSI.SYS é necessário inserir no arquivo **CONFIG.SYS** (para MS-DOS e MS-Windows 9x) ou **CONFIG.NT** (para MS-Windows NT, 2000, XP, Vista, 7 ou 8 - de 32 *bits*) uma das seguintes linhas de comando em qualquer posição do arquivo, antes do comando *files*, se este existir:

Tabela 7.3 - Localização do Arquivo ANSI.SYS

Sistema operacional	Linha de configuração para arquivo CONFIG
MS-DOS	device=c:\dos\ansi.sys
MS-Windows 9x	device=c:\windows\command\ansi.sys
MS-Windows NT/2000/XP/Vista/7/8	device=%SystemRoot%\system32\ansi.sys

Caso esteja utilizando o MS-Windows ME, a linha deve ser inserida no arquivo **CONFIG.SYS** de forma diferente, em razão de seu mecanismo de segurança contra alterações nos arquivos de sistema:

- Em primeiro lugar, reinicialize o computador com o disco de inicialização.
- Em seguida, vá para o *drive* **C** e insira a linha **device=c:\windows\command\ansy.sys** no arquivo **CONFIG.SYS**.

Retire do *drive* o disquete e proceda à reinicialização do computador. A partir desse momento, o *driver* **ansi.sys** está carregado.

Os sistemas operacionais de 64 *bits* da Microsoft são desprovidos de tal recurso, como já mencionado, tendo como alternativa o programa **ANSICON** do sítio: https://github.com/adoxa/ansicon/downloads (acessado em agosto de 2013) e baixar a última versão do programa na lista apresentada. Em seguida, descompacte o arquivo e copie o conteúdo da pasta **x86** (Windows 32 *bits*) ou

x64 (Windows 64 *bits*) para a pasta **System32**. Abra a janela de *prompt de comando* e execute o comando **ansicon –i** para habilitar os recursos do Terminal ANSI, que ficará disponível, automaticamente, todas as vezes que o modo *prompt de comando* for executado. Mas somente instale esse recurso se não desejar, em hipótese nenhuma, usar o modo Windows API.

Para desabilitar o programa **ANSICON** do modo *prompt de comando*, é necessário fazer uma alteração na chave do arquivo de registro do Windows. Assim sendo, execute o programa **regedit** e localize em **HKEY_CURRENT_USER\Software\Microsoft\Command Processor** a chave **AutoRun**, dê duplo clique para que seja apresentada a caixa de diálogo **Editar Cadeia de Caracteres** e remova do campo **Dados do Valor** seu conteúdo, deixando-o em branco. Após acionar o botão **OK**, feche o programa **regedit**. Na próxima vez em que o programa de acesso ao modo *prompt de comando* for chamado, o recurso **ANSICON** estará desabilitado.

7.5.2 Modo Windows API

Windows API é um recurso que oferece aos sistemas operacionais Windows, em suas diversas versões, sejam de 32 *bits*, sejam de 64 *bits*, um extenso conjunto de funções e serviços voltados ao acesso do modo console do sistema e, também, ao acesso do modo gráfico. Para usar esse recurso, não é necessário instalar e configurar nenhuma ferramenta externa; basta utilizar a biblioteca **windows.h** existente no compilador TDM-GCC.

O modo Windows API é um componente operacional com funcionalidades que o diferem dos recursos encontrados no modo Terminal ANSI, o que os torna incompatíveis, podendo, prejudicar o desenvolvimento de programas multiplataforma que respeitem o atendimento da norma ISO da linguagem C.

Primeiro, será focado o modo Terminal ANSI; em seguida, será apresentado o acesso aos recursos da Windows API.

7.5.3 Arquivo de Cabeçalho Externo: Terminal ANSI

Para controlar o vídeo ou o teclado por meio dos recursos do modo Terminal ANSI/VT100, é necessário usar comandos específicos, precedidos do código de controle de *escape* que, na linguagem C, consiste no código numérico octal

\033 de acesso à função da tecla **<Esc>**. Todo valor octal, em linguagem de programação C, é iniciado com o algarismo **0** (zero) à esquerda.

Os comandos (sequência de escape) são divididos em dois grupos de ação (vídeo e teclado). No grupo *vídeo*, é possível manipular vídeo, controlar e movimentar o cursor e os atributos de tela. No grupo *teclado*, é possível redefinir a ação de qualquer tecla.

A seguir, é apresentada apenas uma parte da tabela com os códigos de controle de vídeo. As funções do teclado não serão usadas, por isso são omitidas. Para maiores detalhes sobre códigos de controle de *escape*, consulte o sítio **MS-DOS 6 - Device Drivers**, no endereço **http://users.cybercity.dk/~bse26236/batutil/help/ANSI~S_S.HTM**, ou o sítio **ANSI CODES**, no endereço **http://bluesock.org/~willg/dev/ansi.html** (visitados em agosto de 2013).

As tabelas seguintes apresentam os códigos de controle ANSI para limpar vídeo, movimentar cursor e determinar a posição do cursor. A indicação **ESC** deve se substituída pelo código **\033**.

Tabela 7.4 - Código ANSI para Limpar Vídeo

Limpeza do vídeo		
Sequência ESC	**Função**	**Descrição**
ESC[2J	Limpar vídeo	Efetua a limpeza do vídeo e posiciona o cursor no topo esquerdo da tela.
ESC[K	Limpar linha	Efetua a limpeza da linha a partir da posição em que o cursor estiver parado.

Tabela 7.5 - Código ANSI para Manipular Cursor

Posição do cursor		
Sequência ESC	**Função**	**Descrição**
ESC[s	Salva posição	Salva a posição atual do cursor para posterior restauração.
ESC[u	Restaura posição	Restaura a posição do cursor previamente salvo com a sequência de salvamento.

Tabela 7.6 - Código ANSI para Movimentar Cursor

Movimentação do cursor		
Sequência ESC	**Função**	**Descrição**
ESC[valor A	Cursor para cima	Move o cursor para cima o número de vezes estabelecido no parâmetro *valor*.
ESC[valor B	Cursor para baixo	Move o cursor para baixo o número de vezes

Movimentação do cursor		
Sequência ESC	**Função**	**Descrição**
		estabelecido no parâmetro *valor*.
ESC[valor C	Cursor para direita	Move o cursor para a direita o número de vezes estabelecido no parâmetro *valor*.
ESC[valor D	Cursor para esquerda	Move o cursor para a esquerda o número de vezes estabelecido no parâmetro *valor*.
ESC[lin;col H	Posiciona cursor	Move o cursor para a *lin* (linha - 1 a 25) e *col* (coluna - 1 a 80) especificadas.

É necessário considerar que o recurso de modo Terminal ANSI é utilizado diretamente pelo sistema operacional e não pelo programa em si. O programa apenas o ativa por intermédio de suas instruções.

Observação

É pertinente considerar que, se o leitor utilizar o ambiente de programação **Code::Blocks** em um dos sistemas operacionais Windows, terá problemas no uso das funções de Terminal ANSI, mesmo que tenha configurado a execução do driver ANSI.SYS ou instalado o programa ANSICON. O **Code::Blocks** para Windows, quando instalado, não executa de seu ambiente as funções de Terminal ANSI nos sistemas operacionais Microsoft, pois usa o compilador TDM-GCC (GCC para Windows com suporte direto a Windows API), o qual possui recursos próprios para tal ação. Mais à frente serão mostrados os recursos de acesso ao modo Windows API.

Para fazer um teste básico de funcionalidade dos códigos de controle ANSI, em seguida são apresentadas duas funções, sendo uma para limpar a tela (função **clear**) e outra para posicionar o cursor (função **position**) em um determinado ponto da tela. Observe atentamente o seguinte exemplo de programa:

```
/* Definicao de Funcoes ANSI */

#include <stdio.h>

void clear(void);
void position(int LINE, int ROW);

/* Teste das Funcoes Definidas */

int main(void)
{
  clear();
```

```
  position(12,23); /* Escreve na linha 12 coluna 23 */
  printf("Teste de Limpeza e Posicionamento");
  position(23, 1);
  return 0;
}

/**********************/
/* Funcao: Limpa Tela */
/**********************/

void clear(void)
{
  printf("\033[2J");
  return;
}

/****************************/
/* Funcao: Posiciona Cursor */
/****************************/

void position(int LINE, int ROW)
{
  printf("\033[%d;%dH", LINE, ROW);
  return;
}
```

Em seguida grave o programa com o nome **progansi.c** e rode-o para observar seu funcionamento. Se o driver ANSI não estiver carregado, o efeito apresentado será semelhante às seguintes linhas:

```
←[2J←[2;20HTeste de Limpeza e Posicionamento
←[24;33HTecle <Enter>
```

Mas se o driver ANSI estiver devidamente carregado, o efeito será muito bom. A tela é limpa e o texto escrito na posição desejada. Vale salientar que podem ser utilizados para a linha valores entre 1 e 24, para a coluna valores entre 1 e 80.

É necessário levar em conta que o programa compilado precisa do driver ANSI instalado no computador em que o programa será executado (por exemplo, no computador de um cliente). Os códigos de controle ANSI são usados pelo sistema operacional e não pelo programa em si. O programa apenas os ativa por intermédio de suas instruções.

Esta recomendação é necessária apenas para sistemas operacionais da Microsoft, como MS-DOS e Prompt do MS-DOS (quando executado de dentro do Windows).

Para utilizar os códigos ANSI em sistemas operacionais como UNIX e Linux, não é necessário instalar nenhum driver de controle ANSI. Esses sistemas operacionais conseguem interpretar os códigos automaticamente.

As funções definidas para um programa podem ser utilizadas em outros programas. Imagine que as funções **clear()** e **position()** (definidas pelo programador), bem como as demais funções desenvolvidas, úteis a outros programas. Para evitar o inconveniente de sempre ter de escrevê-las no código de um novo programa, é possível guardá-las em um arquivo pessoal de biblioteca externa.

Para evitar o inconveniente de precisar escrevê-las no código de programa a ser definido, há a possibilidade de guardá-las em um arquivo de cabeçalho pessoal semelhante aos arquivos de cabeçalho, como **stdio.h**, **math.h**, entre outros.

Para utilizar funções externas definidas pelo próprio programador, é preciso ter no mínimo três arquivos de programa, sendo o primeiro pelos protótipos das funções, o segundo pelo código das funções a serem usadas e o terceiro pelo programa em si.

A partir do exposto escreva o código seguinte referente às constantes e aos cabeçalhos dos protótipos das funções a serem usadas. Assim sendo, digite o código seguinte do programa com as funções de limpeza de tela e posicionamento do cursor.

```
/* Definicao do arquivo de cabecalho ANSI */

void clear(void);
void position(int LINE, int ROW);
void clearline(void);
```

Grave o programa com o nome **crt.h**.

Completada a primeira etapa, é necessário criar o arquivo que conterá os códigos das funções. Assim sendo, grave o programa seguinte com o nome **bibcrt.c**.

```
/* Codigo da bilbioteca CRT */

#include <stdio.h>

/**********************/
/* Funcao: Limpa Tela */
/**********************/

void clear(void)
{
  printf("\033[2J");
  return;
}

/****************************/
/* Funcao: Posiciona Cursor */
/****************************/

void position(int LINE, int ROW)
{
  printf("\033[%d;%dH", LINE, ROW);
  return;
}

/*****************************************************/
/* Funcao: Limpa Linha a partir da posicao do cursor */
/*****************************************************/

void clearline(void)
{
  printf("\033[K");
  return;
}
```

Além dos comandos ANSI para limpar a tela e posicionar o cursor, define-se a função **clearline()** que tem a finalidade de limpar toda linha de texto escrita no vídeo, a partir da posição em que o cursor se encontra. No próximo capítulo será escrito um programa que usa essa função.

Para fazer um teste da nova biblioteca, altere o programa **calculo5.c** de acordo com as indicações seguintes em negrito e grave-o com o nome **calcula5.c**.

```
/* Programa Calculadora - ANSI */

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

float R, A, B;

int rotadicao(void);
int rotsubtracao(void);
int rotmultiplicacao(void);
```

```
int rotdivisao(void);
void entrada(void);
void saida(void);
float calculo(float X, float Y, char OPERADOR);
int main(void)
{
  int OPCAO = 0;
  while (OPCAO != 5)
    {
      clear();
      position( 1, 1); printf("Menu Principal");
      position( 2, 1); printf("--------------");
      position( 4, 1); printf("1 - Adicao");
      position( 5, 1); printf("2 - Subtracao");
      position( 6, 1); printf("3 - Multiplicacao");
      position( 7, 1); printf("4 - Divisao");
      position( 8, 1); printf("5 - Fim de Programa");
      position(10, 1); printf("Escolha uma opcao: ");
      fflush(stdin); scanf("%d", &OPCAO);
      if (OPCAO != 5)
        {
          switch (OPCAO)
            {
              case 1  : rotadicao();        break;
              case 2  : rotsubtracao();     break;
              case 3  : rotmultiplicacao(); break;
              case 4  : rotdivisao();       break;
              default : printf("Opcao invalida - Tecle <Enter>");
                        getchar() + scanf("Enter");
                        break;
            }
        }
    }
  return 0;
}

void entrada(void)
{
  position( 5, 1); printf("Entre um valor para A: ");
  fflush(stdin); scanf("%f", &A);
  position( 6, 1); printf("Entre um valor para B: ");
  fflush(stdin); scanf("%f", &B);
  return;
}

void saida(void)
{
  position( 9, 1); printf("\nO resultado entre A e B = %6.2f\n", R);
  position(11, 1);
  printf("\nTecle <Enter> para acessar o menu: ");
  getchar() + scanf("Enter");
  return;
}

float calculo(float X, float Y, char OPERADOR)
{
  float RESULTADO;
```

```
  switch (OPERADOR)
    {
      case '+' : RESULTADO = X + Y; break;
      case '-' : RESULTADO = X - Y; break;
      case '*' : RESULTADO = X * Y; break;
      case '/' : RESULTADO = X / Y; break;
    }
  return RESULTADO;
}

int rotadicao(void)
{
  clear();
  position( 1, 1); printf("Rotina de Soma");
  position( 2, 1); printf("--------------");
  entrada();
  R = calculo(A, B, '+');
  saida();
  return 0;
}

int rotsubtracao(void)
{
  clear();
  position( 1, 1); printf("Rotina de Subtracao");
  position( 2, 1); printf("-------------------");
  entrada();
  R = calculo(A, B, '-');
  saida();
  return 0;
}

int rotmultiplicacao(void)
{
  clear();
  position( 1, 1); printf("Rotina de Multiplicacao");
  position( 2, 1); printf("-----------------------");
  entrada();
  R = calculo(A, B, '*');
  saida();
  return 0;
}

int rotdivisao(void)
{
  clear();
  position( 1, 1); printf("Rotina de Divisao");
  position( 2, 1); printf("-----------------");
                                                        entrada();
  if (B == 0)
    {
      position( 9, 1); printf("Erro de divisao");
      position(11, 1);
      printf("\nTecle <Enter> para acessar o menu: ");
      getchar() + scanf("Enter");
    }
  else
```

```
    {
      R = calculo(A, B, '/');
      saida();
    }
  return 0;
}
```

A biblioteca **crt.h** é chamada pela linha de comando **#include "crt.h"** e não pela linha **#include <crt.h>**. Existem duas formas aceitas de fazer referência à chamada de uma biblioteca externa por pré-processamento, sendo uma por meio dos símbolos (<) menor que e (>) maior que, e outra por meio de aspas.

A diferença entre elas é que para as bibliotecas-padrão (aquelas que acompanham o compilador) a referência será sempre feita entre os sinais de menor que e maior que. Referências em relação a bibliotecas externas criadas por um programador para atender a uma necessidade específica e particular devem ser feitas entre aspas. Torna-se fácil distinguir uma da outra.

Observe o uso das funções usadas pelo programador **clear()** e **position(**line, row**)** em vários pontos do programa.

Para compilar o programa usando a biblioteca baseada em arquivo de cabeçalho, execute no *prompt* de comando um dos seguintes comandos, dependendo do sistema operacional em uso:

- **gpp calcula5.c bibcrt.c –o teste (MS-Windows)**
- **g++ -o teste calcula5.c bibcrt.c (Linux)**

Ao ser executado o comando de compilação, é criado o programa executável **TESTE** que é o resultado da junção do programa **calcula5.c** com os recursos disponíveis no programa **bibcrt.c** chamados por intermédio da instrução **#include "crt.h"**.

7.5.4 Arquivo de Cabeçalho Externo: Windows API

Em seguida, é exibido um programa exemplo que faz uso dos recursos do modo *Windows API*, para os sistemas operacionais Windows pelo ambiente de programação **Code::Blocks**.

```
/* Definicao de Funcoes Windows API */

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

```
void clear(void);
void position(int LINE, int ROW);

/* Teste das Funcoes Definidas */

int main(void)
{
  clear();
  position(12,23); /* Escreve na linha 12 coluna 23 */
  printf("Teste de Limpeza e Posicionamento");
  position(23, 1);
  return 0;
}

/**********************/
/* Funcao: Limpa Tela */
/**********************/

void clear(void)
{
  HANDLE TELA;
  DWORD ESCRITA = 0;
  COORD POS;
  TELA = GetStdHandle(STD_OUTPUT_HANDLE);
  POS.X = 0;
  POS.Y = 0;
  FillConsoleOutputCharacter(TELA, 32, 80 * 25, POS, &ESCRITA);
  return;
}

/***************************/
/* Funcao: Posiciona Cursor */
/***************************/

void position(int LINE, int ROW)
{
  HANDLE TELA;
  COORD POS;
  TELA = GetStdHandle(STD_OUTPUT_HANDLE);
  POS.X = ROW - 1;
  POS.Y = LINE - 1;
  SetConsoleCursorPosition(TELA, POS);
  return;
}
```

Em seguida, grave o programa com o nome **progwapi.c**, compile-o e rode-o para observar seu funcionamento. O programa apresenta a execução de efeitos de limpeza e posicionamento de cursor na tela semelhante aos recursos reproduzidos com o modo Terminal ANSI.

Cabe, descrever com mais detalhes cada uma das ações utilizadas por meio do arquivo de cabeçalho **windows.h**, que acessa os recursos da Windows API.

Limpeza da tela

Para a limpeza da tela do monitor de vídeo no sistema operacional Microsoft Windows, usa-se a função **FillConsoleOutputCharacter()**, que preenche a tela com certo caractere fornecido. Note, no programa, as variáveis **ESCRITA** e **POS**. A variável **ESCRITA**, do tipo **DWORD**, indica o número de caracteres a ser escritos no *buffer* da tela durante o processamento da limpeza. A variável **POS** é usada para posicionar o cursor na primeira linha e coluna. O tipo **DWORD** é representado por um valor inteiro sem sinal.

Veja as linhas de código **POS.X** e **POS.Y** atribuídas com valor **0** (zero). Esses valores serão usados para posicionar o cursor no lado superior esquerdo da tela.

A função **FillConsoleOutputCharacter()** preenche a tela com o caractere fornecido, um certo número de vezes, iniciando sua ação na coordenada indicada. Essa função requer cinco parâmetros. O primeiro parâmetro, representado pela variável **TELA**, indica sua ação no monitor de vídeo; o segundo parâmetro, representado pelo valor **32**, corresponde ao código ASCII do caractere de espaço em branco (caractere usado para limpar a tela); o terceiro parâmetro, representado pelo valor **80 * 25**, fornece o comprimento da tela a ser limpa (nesse exemplo, leva-se em conta uma tela com 80 colunas e 25 linhas, tamanho-padrão da tela do console); o quarto parâmetro, representado pela variável **POS**, indica a posição inicial em que o caractere de espaço em branco será escrito; e o quinto parâmetro, representado pela variável **RESCRITA**, indica a quantidade de caracteres escritos no *buffer* do monitor de vídeo.

Posição do cursor

Para posicionar o cursor na tela do monitor no sistema operacional Microsoft Windows, usa-se a função **SetConsoleCursorPosition()**, que coloca o cursor nas coordenadas de tela indicadas. Essa função usa dois parâmetros: **TELA**, que estabelece o local onde a ação será efetivada, e **POS**, que define o local onde o cursor será colocado.

Com base no exemplo exposto, será criada uma biblioteca com as funcionalidades apresentadas. O próximo programa assemelha-se aos recursos usados no programa **calcula5.c**. Assim sendo, serão desenvolvidos três programas: o arquivo de cabeçalho **crtw.h**, o arquivo de biblioteca **bibcrtw.c** e o arquivo de programa **calcula6.c**.

A partir do exposto, escreva o código seguinte referente à definição das constantes e dos cabeçalhos dos protótipos das funções a serem usadas. Assim sendo, digite o código seguinte do programa com as funções de limpeza de tela e posicionamento do cursor.

```
 /* Definicao do arquivo de cabecalho Windows API */

void clear(void);
void position(int LINE, int ROW);
void clearline(void);
```

Grave o programa com o nome **crtw.h**.

Completada a primeira etapa é necessário criar o arquivo que conterá os códigos das funções. Assim sendo, grave o programa seguinte com o nome **bibcrtw.c**.

```
/* Codigo da bilbioteca CRTW */

#include <stdio.h>
#include <windows.h>

/**********************/
/* Funcao: Limpa Tela */
/**********************/

void clear(void)
{
  HANDLE TELA;
  DWORD ESCRITA = 0;
  COORD POS;
  TELA = GetStdHandle(STD_OUTPUT_HANDLE);
  POS.X = 0;
  POS.Y = 0;
  FillConsoleOutputCharacter(TELA, 32, 80 * 25, POS, &ESCRITA);
  return;
}

/***************************/
/* Funcao: Posiciona Cursor */
/***************************/

void position(int LINE, int ROW)
{
  HANDLE TELA;
  COORD POS;
  TELA = GetStdHandle(STD_OUTPUT_HANDLE);
  POS.X = ROW - 1;
  POS.Y = LINE - 1;
  SetConsoleCursorPosition(TELA, POS);
  return;
}
```

```
/*****************************************************/
/* Funcao: Limpa Linha a partir da posicao do cursor */
/*****************************************************/

void clearline(void)
{
  HANDLE TELA;
  COORD POS;
  CONSOLE_SCREEN_BUFFER_INFO VIDEO;
  DWORD ESCRITA = 0;
  TELA = GetStdHandle(STD_OUTPUT_HANDLE);
  GetConsoleScreenBufferInfo(TELA, &VIDEO);
  POS.Y = VIDEO.dwCursorPosition.Y;
  POS.X = VIDEO.dwCursorPosition.X;
  FillConsoleOutputCharacter(TELA, 32, 80 - POS.X , POS, &ESCRITA);
  return;
}
```

O código de programa anterior usa novos recursos do modo Windows API, comentados a seguir:

Limpeza de linha

Para a limpeza de uma linha na tela do monitor no sistema operacional Microsoft Windows, é necessário ter uma função que detecte a posição do cursor e, com base nela, limpe todos os caracteres até o final da linha. Para esse efeito, precisa-se da função **GetConsoleScreenBufferInfo()**, usada em um objeto definido pelo tipo **CONSOLE_SCREEN_BUFFER_INFO**, utilizado para obter alguma informação existente no *buffer* do monitor de vídeo (*console screen*).

Note a variável **VIDEO** do tipo **CONSOLE_SCREEN_BUFFER_INFO**, usada para capturar a informação armazenada no *buffer* do console de vídeo com a função **GetConsoleScreenBufferInfo()**.

Assim que a função **GetConsoleScreenBufferInfo()** é executada, passa a ter os valores das coordenadas da tela capturadas com **dwCursorPosition** e atribuídas à variável **POS** do tipo **COORD**, que será usada pela função **FillConsoleOutputCharacter()**.

Já é sabido que a função **FillConsoleOutputCharacter()** é usada para preencher a tela com um caractere fornecido como segundo parâmetro, certo número de vezes, no caso **80 – POS.X**, iniciando sua ação na coordenada indicada em **POS**.

O parâmetro **80 – POS.X** indica que a coordenada a ser processada pela função é a **X** (linha), atualmente capturada. O valor capturado a ser subtraído de **80** refere-se à quantidade de caracteres a serem limpos da linha desde a posição atual até o limite de 80 caracteres, levando em consideração um console de texto com 80 colunas.

O programa seguinte vai testar a biblioteca para o modo Microsoft Windows. Grave o programa com o nome **calcula6.c**.

```
 /* Programa Calculadora - Windows API */

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

float R, A, B;

int rotadicao(void);
int rotsubtracao(void);
int rotmultiplicacao(void);
int rotdivisao(void);
void entrada(void);
void saida(void);
float calculo(float X, float Y, char OPERADOR);

int main(void)
{
  int OPCAO = 0;
  while (OPCAO != 5)
    {
      clear();
      position( 1, 1); printf("Menu Principal");
      position( 2, 1); printf("--------------");
      position( 4, 1); printf("1 - Adicao");
      position( 5, 1); printf("2 - Subtracao");
      position( 6, 1); printf("3 - Multiplicacao");
      position( 7, 1); printf("4 - Divisao");
      position( 8, 1); printf("5 - Fim de Programa");
      position(10, 1); printf("Escolha uma opcao: ");
      fflush(stdin); scanf("%d", &OPCAO);
      if (OPCAO != 5)
        {
          switch (OPCAO)
            {
              case 1  : rotadicao();        break;
              case 2  : rotsubtracao();     break;
              case 3  : rotmultiplicacao(); break;
              case 4  : rotdivisao();       break;
              default : printf("Opcao invalida - Tecle <Enter>");
                        getchar() + scanf("Enter");
                        break;
            }
        }
    }
  return 0;
```

```
}

void entrada(void)
{
  position( 5, 1); printf("Entre um valor para A: ");
  fflush(stdin); scanf("%f", &A);
  position( 6, 1); printf("Entre um valor para B: ");
  fflush(stdin); scanf("%f", &B);
  return;
}

void saida(void)
{
  position( 9, 1); printf("\nO resultado entre A e B = %6.2f\n", R);
  position(11, 1);
  printf("\nTecle <Enter> para acessar o menu: ");
  getchar() + scanf("Enter");
  return;
}

float calculo(float X, float Y, char OPERADOR)
{
  float RESULTADO;
  switch (OPERADOR)
    {
      case '+' : RESULTADO = X + Y; break;
      case '-' : RESULTADO = X - Y; break;
      case '*' : RESULTADO = X * Y; break;
      case '/' : RESULTADO = X / Y; break;
    }
  return RESULTADO;
}

int rotadicao(void)
{
  clear();
  position( 1, 1); printf("Rotina de Soma");
  position( 2, 1); printf("--------------");
  entrada();
  R = calculo(A, B, '+');
  saida();
  return 0;
}

int rotsubtracao(void)
{
  clear();
  position( 1, 1); printf("Rotina de Subtracao");
  position( 2, 1); printf("-------------------");
  entrada();
  R = calculo(A, B, '-');
  saida();
  return 0;
}

int rotmultiplicacao(void)
{
  clear();
```

```
  position( 1, 1); printf("Rotina de Multiplicacao");
  position( 2, 1); printf("-----------------------");
  entrada();
  R = calculo(A, B, '*');
  saida();
  return 0;
}

int rotdivisao(void)
{
  clear();
  position( 1, 1); printf("Rotina de Divisao");
  position( 2, 1); printf("-----------------");
  entrada();
  if (B == 0)
    {
      position( 9, 1); printf("Erro de divisao");
      position(11, 1);
      printf("\nTecle <Enter> para acessar o menu: ");
      getchar() + scanf("Enter");
    }
  else
    {
      R = calculo(A, B, '/');
      saida();
    }
  return 0;
}
```

Em relação à versão 64 *bits* do sistema operacional Windows, com o compilador TDM-GCC instalado é possível usar uma das seguintes linhas de comando no *prompt*: **g++ -o teste calcula6.c bibcrtw.c**.

Se estiver em uso o ambiente **Code::Blocks**, pode-se utilizar um projeto que compile o programa por meio dos arquivos de código-fonte. Para tanto, execute, no menu do programa, a sequência de comandos:

File
New
Project...

Ao ser apresentada a caixa de diálogo **New from template**, selecione a opção **Console application** e clique no botão **Go**. Ao surgir a caixa de diálogo **Console application**, acione o botão **Next**, selecione a opção **C** e acione o botão **Next** novamente. Na etapa seguinte, insira no campo **Project title** o nome do projeto, no caso **Calcula6**, e acione o botão **Next** mais uma vez. Na etapa posterior, acione o botão **Finish**.

Na janela lateral esquerda, denominada **Management**, encontra-se a árvore de diretórios do computador. Localize na pasta **Calcula6**, abaixo de **Workspaces**,

o arquivo **main.c**, em **Sources**, e selecione-o com um duplo clique. Note que ele é apresentado com o código inicial de um programa "**Hello world!**".

O arquivo **main.c** deve ser removido. Assim sendo, com ele selecionado, acione o botão direito do *mouse* e, no menu de contexto apresentado, escolha a opção **Remove file from Project**.

Em seguida, execute o comando de menu:

Project
Add Files...

Na caixa de diálogo **Add files to project**, selecione, primeiro, o arquivo **calcula6.c**; depois, execute novamente o comando de menu **Project/Add Files...** e selecione o arquivo **crtw.h**; por último, execute mais uma vez o comando de menu **Project/Add Files...** e selecione o arquivo **bibcrt.c**.

A partir do momento em que os três arquivos estiverem carregados na pasta de projeto, compile o programa e peça sua execução.

7.6 Interação com Linha de Comando

Neste tópico, após a apresentação de diversos detalhes operacionais sobre linguagem C, funções e passagens de parâmetro é pertinente apresentar um exemplo de como fazer uso dos argumentos transferidos para um programa C a partir da linha de comando do sistema operacional.

Desenvolver um programa que efetue na linha de comando do sistema operacional a entrada de valores referentes a uma data calendário: dia, mês e ano e apresente a data informada por extenso se a data for válida. Deve ser considerada data válida quando o dia estiver entre 1 e 31, mês estiver entre 1 e 12 e ano estiver entre 1000 e 9999.

O programa a seguir não leva em conta a data referente a anos bissextos ou os meses em que há 30 ou 31 dias. Assim sendo, se ocorrer a entrada de uma data como **30 02 2012** (30 de fevereiro de 2012), o programa mostra a data como válida. Fica a critério do leitor, como exercício, implementar as verificações necessárias para evitar o aceite pelo programa de datas fora das faixas permitidas. Observe o código seguinte e grave o programa com o nome **data.c**.

```
/* Data calendario por extenso */

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

void main(int argc, char *argv[])
{

  char MES_EXT[][10] =
        {
          "janeiro",
          "fevereiro",
          "marco",
          "abril",
          "maio",
          "junho",
          "julho",
          "agosto",
          "setembro",
          "outubro",
          "novembro",
          "dezembro"
        };
  int DIA, MES, ANO;

  if (argc == 4)
    {

      DIA = atoi(argv[1]);
      MES = atoi(argv[2]);
      ANO = atoi(argv[3]);

      if (DIA < 1 || DIA > 31)
        printf("\nForneca dia entre 1 e 31.");
      else if (MES < 1 || MES > 12)
        printf("\nForneca mes entre 1 e 12.");
      else if (ANO < 1000 || ANO > 9999)
        printf("\nForneca ano entre 1000 e 9999.");
      else
        printf("\n%s de %s de %s", argv[1], MES_EXT[MES-1], argv[3]);
    }
  else
    printf("\nIncorreto: data DD MM AAAA.");

  return;
}
```

Observe no programa a função **main()** do tipo de retorno **void**, usado com a função **main()** quando do uso dos parâmetros **argc** e **argv**. Com os parâmetros **argc** e **argv** é possível passar dados na execução do programa na linha de comando do sistema operacional.

Após compilar o programa, faça na linha de comando do sistema operacional a chamada com uma das seguintes variações de execução:

data 12 05 2012

data 32 05 2012

data 12 13 2012

data 12 05 12

data 12 05

Das formas apresentadas apenas a primeira resulta na apresentação da mensagem **12 de maio de 2012**.

No sentido de entender o que ocorre com o programa, considere que **argc** (argument count, parâmetro contador) passa para o programa, no caso, **data** a quantidade de parâmetros que foi informada na linha de comando no programa. O valor do parâmetro **argc** é inteiro e positivo, sendo no mínimo **1** e representa a quantidade de parâmetros passados para o programa. No caso da chamada **data 12 05 2012** o valor de número de parâmetros passado é **4**, considerando-se o nome do programa, o dia, o mês e o ano.

O parâmetro **argv** (argument values, parâmetro de valores) é um ponteiro para uma matriz do tipo *string*, em que cada elemento desse *string* são os valores fornecidos como parâmetros da linha de comando, sendo **argv[0]** o nome do programa, **argv[1]** o valor do dia, **argv[2]** o valor do mês e **argv[3]** o valor do ano.

Quando os parâmetros são fornecidos na linha de comando automaticamente a matriz **argv** é carregada com os valores passados e o parâmetro **argc** é carregado com o número de parâmetros em uso.

Na função **main()** do programa estão as definições de trabalho. Entre elas a da matriz **MÊS_EXT** com os nomes dos meses a serem apresentados, além das variáveis inteiras **DIA**, **MÊS** e **ANO**.

Na sequência verifica-se a quantidade de parâmetros existentes em **argc** que deve ser quatro. Qualquer outro valor existente fará com que seja apresentada a mensagem: "**Incorreto: data DD MM AAAA.**", mostrando que algo deixou de ser fornecido na chamada do programa.

Se as informações fornecidas são corretas, o programa por meio da função **atoi()**, *alfa to integer* da biblioteca **stdlib.h** que tem finalidade converter em formato numérico inteiro os valores fornecidos na chamada do programa

como caracteres alfanuméricos. Note que cada uma das funções **atoi()** é atribuída a uma das variáveis a partir da informação existente na matriz **argv**.

Na sequência efetivam-se as verificações de validade para os valores fornecidos como data de calendário. A mensagem "**\n%s de %s de %s**" é responsável pela apresentação da data por extenso a partir da apresentação dos valores dos parâmetros **argv[1]** (dia), **MES_EXT[MES-1]** (nome do mês por extenso) e **argv[3]** (ano).

Exercícios

1. Desenvolva em linguagem C os programas dos problemas indicados em seguida. Procure resolver cada exercício de duas formas diferentes, sendo uma forma com funções que trabalhem com passagem de parâmetro por valor; outra forma com funções que trabalhem com retorno de valor, se assim for possível. Como sugestão, você pode gravar o exercício "a" com o nome CAP07EX1A_V para a representação da passagem de parâmetro por valor e com o nome CAP07EX1A_R para a representação do retorno de valor (apenas para os exercícios a, b, d, e).

   a. Elaborar um programa que possua uma função que permita apresentar o somatório dos N (1+2+3+4+5+6+7+...+N) primeiros números inteiros. O valor de N deve ser fornecido pelo usuário do programa.

   b. Escreva um programa que utilize uma função para calcular a série de Fibonacci de N termos. O valor de N será fornecido pelo usuário e o programa deve apresentar o valor da sequência correspondente ao termo N informado.

   c. Ler duas matrizes A e B do tipo vetor com dez elementos numéricos inteiros cada. Construir e apresentar a matriz C, sendo esta a junção das duas outras matrizes. Desta forma, C deve ter o dobro de elementos de A e B. Neste exercício, deve ser criada uma função para cada tarefa do programa, ou seja, serão quatro sub-rotinas, sendo duas para leitura das matrizes, uma para a junção e a última para apresentação dos dados. No final o programa principal deve chamar as rotinas definidas.

d. Criar um programa que calcule e apresente o valor de uma prestação em atraso, utilizando a fórmula P = V + (V * (TX/100) * T). No caso P representa prestação, V representa valor, TX representa taxa e T representa tempo.

e. Desenvolva um programa que crie uma função para calcular o valor de uma potência de um número inteiro qualquer, ou seja, ao informar para a sub-rotina o valor da base e do expoente (ambos inteiros), deve ser apresentado o resultado da potência calculada. Por exemplo, se for mencionado no programa principal a chamada da sub-rotina *POTENCIA(2,3)*, deve ser apresentado o valor 8. Não utilize nenhuma função interna da linguagem C; busque a solução deste problema utilizando apenas laço de repetição.

f. Ler uma matriz A com 12 elementos reais. Após sua leitura, colocar os seus elementos em ordem crescente. Depois ler uma matriz B também com 12 elementos reais, e colocar os elementos de B em ordem crescente. Construir uma matriz C, sendo cada elemento de C a soma do elemento correspondente de A com B. Colocar em ordem decrescente a matriz C e apresentar os seus elementos. Neste exercício, deve ser criada uma função para cada tarefa do programa.

2. Com base nas técnicas de programação apresentadas até este ponto, desenvolva um programa que, considerando o cadastro de uma agenda com dez endereços, nomes e telefones, crie um programa utilizando funções e menu que efetue:

- o cadastro dos dados (todos os 10);
- a consulta dos dados por Nome;
- a alteração de dados com erro (consulta por nome);
- remoção de dados cadastrados (consulta por nome);
- listagem geral de todos os nomes cadastrados.

Observação

- A rotina de cadastro deve fazer em sua totalidade os dez registros necessários.
- A rotina de consulta deve efetuar uma pesquisa com base no nome cadastrado na agenda e apresentar todos os dados da agenda.
- A rotina de alteração deve realizar uma pesquisa com base no nome cadastrado na agenda, apresentar os dados atuais e solicitar a entrada de novos dados para o registro apresentado.
- A rotina de remoção deve pesquisar com base no nome cadastrado na agenda, apresentar os dados atuais e em seguida apagar o registro atual com sequências de caracteres em branco ("") - entre aspas - use a função strcpy().
- A rotina de apresentação deve apenas mostrar os nomes que estejam cadastrados na agenda.

3. Desenvolver um programa que controle as notas bimestrais de até 18 alunos de uma sala de aula. O programa deve calcular a média de cada aluno, e será considerado aprovado o que obtiver média igual ou superior a 7. As notas atribuídas devem estar na escala de 1 a 10 (não forneça zero). O programa em questão deve executar as seguintes rotinas:

   - cadastrar 18 alunos (pode ser menos) e suas notas;
   - consultar alunos por nome (apresentar nome, notas e média);
   - alterar os dados dos alunos cadastrados com erro;
   - listar as médias e os nomes dos alunos aprovados;
   - listar as médias e os nomes dos alunos reprovados.

Observação

Para cadastrar até 18 alunos o programa deve controlar o acesso à matriz, não permitindo que ocorra maior número de cadastros que 18, mas permita cadastrar caso o número de registros seja inferior a 18.

8

capítulo

Arquivos em Disco

Objetivos

Este capítulo trata de arquivos para o armazenamento de dados a longo prazo. Explica operações com arquivos, formas de acesso, arquivos do tipo texto e binário, gravação e leitura de dados e ponteiros.

Utiliza o comando:

- *FILE*

E as funções:

- *fclose()*
- *fgets()*
- *fopen()*
- *fprintf()*
- *fread()*
- *fscanf()*
- *fseek()*
- *fwrite()*

8.1 Arquivo

Já foi explicado tabela em memória com a utilização de matrizes, que são a base para a utilização de arquivo. As matrizes usam um índice de controle, enquanto os arquivos um ponteiro de registro.

A principal vantagem de um arquivo é que as informações armazenadas podem ser consultadas a qualquer momento. Outra vantagem é o fato de armazenar um número maior de registros do que uma tabela em memória. Está limitado apenas ao tamanho do meio físico para gravação.

É um conjunto de registros (que pode ser apenas um registro) que, por sua vez, é um conjunto de campos (que pode ser apenas um campo), sendo cada campo o conjunto de informações.

As informações NOME, ENDEREÇO, TELEFONE, IDADE são exemplos de campos. Uma ficha de cadastro que contenha os campos para preenchimento é um registro.

8.2 Formas de Acesso

Os arquivos criados com a linguagem C podem ser acessados para leitura e escrita de duas formas: sequencial e aleatória.

8.2.1 Acesso Sequencial

O acesso sequencial ocorre quando o processo de gravação e leitura é feito de forma contínua, um após o outro a partir do primeiro registro, registro a registro, até localizar a primeira posição vazia após o último registro. O processo de leitura também ocorre de forma sequencial. Se o registro a ser lido é o último, primeiramente será necessário ler todos os registros que o antecedem. Esse processo é considerado lento.

8.2.2 Acesso Aleatório

O acesso aleatório ocorre com a transferência de dados diretamente para qualquer posição do arquivo, sem que para isso as informações anteriores precisem ser lidas (acesso sequencial). O acesso aleatório a um arquivo pode ser feito de três formas diferentes, com relação ao posicionamento do

ponteiro dentro do arquivo: início do arquivo, fim do arquivo ou o posicionamento atual do ponteiro no arquivo.

8.3 Operações com Arquivo

Um arquivo em linguagem C é do tipo **FILE**, uma estrutura formada por elementos do mesmo tipo dispostos de forma sequencial. Seu objetivo é fazer a comunicação entre a memória principal (RAM) e a secundária (meios magnéticos) por meio do programa e do sistema operacional. Esse tipo deve ser definido com a seguinte sintaxe:

```
FILE <*variável ponteiro>
```

em que:

<*variável ponteiro> - definição de um ponteiro para a estrutura do tipo FILE.

O tipo FILE deve ser escrito em caracteres maiúsculos, sendo uma estrutura cujos elementos são informações que assinalam a condição de processamento e acesso a um arquivo. Essa estrutura está presente na biblioteca **stdio.h**, por isso é necessário sempre indicar no programa a linha **#include <stdio.h>** antes da função **main()**.

Para usar um arquivo (ler ou escrever), é necessário executar duas operações básicas, sendo abertura e fechamento, conseguidas com as instruções **fopen()** e **fclose()**, desde que o arquivo exista.

Para as operações de leitura e escrita de um arquivo, ele deve ser aberto e depois de utilizado precisa ser fechado. Para a abertura de um arquivo em um programa usa-se a seguinte linha de código:

```
<variável ponteiro> = fopen("nome do arquivo","tipo de abertura");
```

em que:

<variável ponteiro> - é a variável declarada como ponteiro do tipo FILE;

nome do arquivo - é o nome do arquivo a ser manipulado;

tipo de abertura - o tipo de abertura do arquivo (ver tabela em seguida).

Para fechar um arquivo, deve ser utilizada em um programa a seguinte linha de código:

```
fclose(<variável ponteiro>);
```

em que:

<variável ponteiro> - é a mesma variável associada à função **fopen()**.

O tipo de abertura de um arquivo é especificado por três códigos do tipo *string*, a saber: letra **r** para leitura (read), letra **w** para gravação (write) e letra **a** para adicionar dados (append). A seguir acompanhe uma tabela com todas as combinações possíveis.

Tabela 8.1 - Opções de acesso a arquivos

Tipo de Abertura	Descrição
R	Este código permite apenas abrir um arquivo texto para leitura de seus dados. É necessário que o arquivo esteja presente no disco.
W	Este código permite apenas abrir um arquivo texto para escrita (gravação). Este código cria o arquivo para ser trabalhado. Caso o arquivo exista, este código recria o arquivo, ou seja, você perde o arquivo criado anteriormente. Deve ser usado com muito cuidado.
A	Este código permite apenas abrir um arquivo texto para escrita (gravação), permitindo acrescentar novos dados ao final dele. Caso o arquivo não exista, ele será então criado.

8.3.1 Arquivo Texto

O arquivo texto possibilita a criação de registros armazenados com tamanhos diferentes (o que não ocorre com os outros tipos de arquivo).

Os arquivos texto estão capacitados a armazenar caracteres, palavras, frases e também dados numéricos. Os números, entretanto, são armazenados como caracteres alfanuméricos, portanto ocupam muito mais espaço em disco do que na memória de um computador. A solução para este detalhe é utilizar funções que usem os números em formato binário, o que será visto mais adiante.

8.3.1.1 Criação, Gravação e Leitura de Palavra

Antes de iniciar qualquer operação com arquivo, é necessário criá-lo. Escreva no compilador o programa apresentado em seguida e grave-o com o nome **arqtxt01.c**.

```
/* Criacao de arquivo texto */

#include <stdio.h>
```

```
int main(void)
{
  FILE *PTRARQ;   /* definicao do ponteiro para o arquivo */
  PTRARQ = fopen("ARQTXT01.XXX", "a");
  fclose(PTRARQ);
  return 0;
}
```

Execute o programa e observe que aparentemente nada aconteceu. Se você se posicionar no diretório de trabalho de seu compilador, pode verificar o arquivo **arqtxt01.xxx** criado com um tamanho de zero byte.

O programa anterior estabelece para a variável ponteiro *PTRARQ* o tipo **FILE**, em seguida a variável ponteiro é igualada à função **fopen (**"ARQTXT01.XXX","a"**);**, que tem por finalidade definir o arquivo texto. O *string* **"a"** cria o arquivo, caso ele não exista. Por fim é utilizada a função **fclose(***PTRARQ***);**, que fecha o arquivo criado.

Após a criação do arquivo **arqtxt01.xxx**, ele pode ser utilizado para a gravação dos dados e/ou informações que deve guardar. Monte o programa seguinte e grave-o com o nome **arqtxt02.c**.

```
/* Grava palavra no arquivo texto */

#include <stdio.h>

int main(void)
{
  FILE *PTRARQ;
  char PALAVRA[20];
  PTRARQ = fopen("ARQTXT01.XXX", "w");

  printf("\n\nEscreva uma palavra: ");
  scanf("%s", PALAVRA);

  fprintf(PTRARQ, "%s", PALAVRA);
  fclose(PTRARQ);
  return 0;
}
```

Após rodar o programa vá para o diretório de trabalho do compilador e execute o comando DOS: **dir *.xxx**. É apresentado o arquivo **arqtxt01.xxx** com um número de bytes maior do que o número existente no momento de sua criação, que era zero.

O programa anterior estabelece uma variável *PALAVRA* do tipo **char** com a capacidade de armazenar até 20 caracteres. As demais instruções já são

conhecidas, com exceção da linha com a função **fprintf(),** que é semelhante à função **printf()**. A diferença é o primeiro argumento, que é a indicação da variável ponteiro e faz a saída da informação para o arquivo em disco e não para o vídeo.

A seguir é apresentado um programa que lê um arquivo texto, mostrando no vídeo a informação armazenada com o programa **arqtxt02.c**. Escreva o seguinte programa no seu compilador e grave-o com o nome **arqtxt03.c**.

```
/* Leitura de um arquivo texto */

#include <stdio.h>

int main(void)
{
  FILE *PTRARQ;
  char PALAVRA[20];
  PTRARQ = fopen("ARQTXT01.XXX", "r");
  fscanf(PTRARQ, "%s", PALAVRA);

  printf("Palavra = %s", PALAVRA);

  fclose(PTRARQ);
  return 0;
}
```

A instrução **fscanf(***PTRARQ*, "%s", *PALAVRA***);** faz a leitura do registro no arquivo, transferindo a informação lida para a variável *PALAVRA* apresentada no vídeo pela instrução **printf();**. Para realizar a leitura, a função **fopen()** usa o *string* **"r"**.

8.3.1.2 Criação, Gravação e Leitura de Frase

Após conhecer um pouco arquivos, será estudado como gravar uma palavra ou frase, caractere a caractere. Este segundo exemplo é diferente do primeiro que apenas gravou uma palavra, pois, no primeiro exemplo, se você tentar digitar uma frase, só consegue gravar a primeira palavra.

O programa seguinte cria e também grava a frase que será digitada e encerrada com a tecla **<Enter>**.

```
/* Cria e grava frase caractere a caractere */

#include <stdio.h>

int main(void)
```

```
{
  FILE *PTRARQ;
  char LETRA;
  PTRARQ = fopen("FRASE.XXX", "w");

  printf("\n\nEscreva a frase desejada\n\n");

  while((LETRA = getchar()) != '\n')
    putc(LETRA, PTRARQ);
  fclose(PTRARQ);
  return 0;
}
```

Após escrever o programa grave-o com o nome **arqtxt04.c**, execute-o e escreva pelo teclado a frase desejada. Vá até o diretório de trabalho do compilador e verifique a existência do arquivo FRASE.XXX. Se quiser ver o conteúdo do arquivo, basta executar o comando DOS **type frase.xxx**.

O segundo exemplo em relação ao primeiro apresenta uma mobilidade maior de trabalho, pois permitiu que fosse gravada uma frase.

Note no programa a instrução **putc(***LETRA*, *PTRARQ***);** que tem por finalidade apenas capturar um caractere digitado no teclado e colocá-lo na variável *LETRA* por meio da referência feita no ponteiro PTRARQ, enquanto o caractere digitado é diferente da tecla **<Enter>**, que no caso é verificado pela instrução **while(**(*LETRA* = **getche()**) != '\n'**)**.

O código **'\n'** representa o acionamento da tecla **<Enter>**, quando se utilizam dados *string* ou *struct*. O símbolo **\r** (usado anteriormente) é utilizado quando a entrada de dados é caracteristicamente numérica.

Observe um exemplo de como fazer a apresentação do conteúdo do arquivo por meio de um programa em C que lê caractere a caractere.

```
/* Le frase caractere a caractere */

#include <stdio.h>

int main(void)
{
  FILE *PTRARQ;
  char LETRA;
  PTRARQ = fopen("FRASE.XXX", "r");

  printf("\n\nFrase = ");

  while((LETRA = fgetc(PTRARQ)) != EOF)
```

```
    printf("%c", LETRA);
  fclose(PTRARQ);
  return 0;
}
```

Após escrever o programa, grave-o com o nome **arqtxt05.c** e execute-o para visualizar a apresentação da frase armazenada no arquivo.

O programa usa a função **fgetc()** para ler apenas um caractere armazenado, o qual é apresentado no vídeo pela função **printf()**, enquanto não for encontrado o fim de arquivo (EOF - End Of File).

Em C, o EOF não é um caractere de controle de fim de arquivo, mas um valor inteiro enviado ao programa pelo sistema operacional (MS-DOS), valor representado no arquivo **stdio.h** como sendo **-1**. Dependendo do sistema operacional, esse valor pode ser diferente.

No exemplo anterior, o fim de arquivo é comparado com uma variável **char**. Neste caso, EOF está sendo interpretado como um valor diferente do código ASCII 255.

8.3.1.3 Manipulação de Arquivo Texto Melhorada

Após terem sido criados alguns programas que possibilitaram usar palavras inteiras e caracteres (caractere a caractere), veja um exemplo que aceita um *string* como um todo, pois tanto a leitura quanto a escrita de um *string* inteiro em um arquivo são mais fáceis do que efetuar a operação caractere a caractere.

O programa seguinte cria e grava frases em um arquivo, semelhante ao programa de gravação de palavras, mas com um detalhe diferente, pois a gravação é feita linha a linha.

```
/* Cria e grava frases */

#include <stdio.h>

int main(void)
{
  FILE *PTRARQ;
  char RESP, FRASE[81];
  RESP = 'S';
  PTRARQ = fopen("FRASE2.XXX", "w");
  while(RESP == 'S' || RESP == 's')
    {
      printf("\n\nDigite uma frase qualquer\n\n");
```

```
      fflush(stdin);
      fgets(FRASE, 81, stdin);
      fputs(FRASE, PTRARQ);
      printf("\nDeseja continuar (S/N)? ");
      fflush(stdin);
      scanf("%c", &RESP);
    }
  fclose(PTRARQ);
  return 0;
}
```

Após escrever o programa, grave-o com o nome **arqtxt06.c**. Observe no programa a instrução **while(***RESP* == 'S' || *RESP* == 's'**)** a qual faz a gravação enquanto o usuário desejar. A função **fgets()** espera o *string* a ser digitado, a qual é encerrada com a tecla **<Enter>**, em seguida o programa tem a função **fputs()**, responsável por gravar no arquivo o *string* digitado.

É possível que o arquivo possua várias frases armazenadas, uma após a outra.

A função **fflush(**stdin**)** retorna o valor **EOF**, caso algum erro seja detectado na operação de arquivo. No capítulo anterior a função **fflush(**stdin**)** foi utilizada para limpar o *buffer* de teclado.

A seguir é apresentado um programa que lê o arquivo FRASE2.XXX criado anteriormente.

```
/* Apresenta frases */

#include <stdio.h>

int main(void)
{
  FILE *PTRARQ;
  char FRASE[81];
  PTRARQ = fopen("FRASE2.XXX", "r");
  while(fgets(FRASE, 80, PTRARQ) != NULL)
    printf("%s", FRASE);
  fclose(PTRARQ);
  return 0;
}
```

Escreva e grave o programa com o nome **arqtxt07.c**. No exemplo, veja a função **fgets()**, a qual possui três argumentos. O primeiro representa o ponteiro que indica o buffer em que está a linha lida. O segundo argumento representa o tamanho máximo de caracteres a ser lido. Esse valor deve ser um a mais que o tamanho de um *string*, devido à existência do caractere

NULL no final de cada *string*; caractere que é colocado pela função **fgets()** e representado pelo código '\0'. O terceiro argumento é o ponteiro FILE.

A função **fgets()** termina a sua leitura quando encontra o caractere de nova linha "\n" no final de cada frase, ou quando encontra o caractere de fim de arquivo EOF.

8.3.1.4 Gerenciar um Arquivo de Nomes

Para exemplificar a utilização de arquivo texto, considere um pequeno programa que deve solicitar e armazenar nomes de pessoas por meio da função "**scanf("%[^\n]", NOME);**" na rotina (função) **cadastrar**.

O programa em questão deve ter um menu com quatro opções, sendo criar arquivo, cadastrar registro, exibir registros cadastrados e finalizar o programa. Observe o algoritmo em seguida:

Programa Principal

1. Definir as variáveis de controle e abertura do arquivo.
2. Apresentar um menu de seleção com quatro opções:
   a. Criar arquivo;
   b. Cadastrar registro;
   c. Exibir registros;
   d. Fim de Programa.
3. Ao ser selecionado um valor, a rotina correspondente deve ser executada.
4. Ao escolher o valor 4, encerrar o programa.

Rotina 1 - Criar arquivo

1. Executar o comando de criação de um arquivo.
2. Fechar o arquivo.
3. Voltar ao programa principal.

Rotina 2 - Cadastrar

1. Abrir o arquivo para cadastramento.
2. Ler o nome.

3. Escrever o nome no arquivo.
4. Fechar o arquivo.
5. Voltar ao programa principal.

Rotina 3 - Exibir

1. Abrir o arquivo para leitura.
2. Apresentar os dados do arquivo enquanto não for encontrado o último registro.
3. Fechar o arquivo.
4. Voltar ao programa principal.

Apesar de simples e não ser totalmente perfeito, o programa dá uma boa visão de arquivos, pois é necessário trabalhar com alguns novos recursos. Escreva o programa seguinte para o seu compilador e grave-o com o nome **nomestxt.c**.

```
/* Programa: Nomes em um arquivo texto */

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

char NOME[41];
FILE *PTRARQ;
void criar(void);
void cadastrar(void);
void exibir(void);

int main(void)
{
  int OPCAO = 0;
  while (OPCAO != 4)
    {
      clear();
      position( 1,33); printf("Menu Principal");
      position( 2,33); printf("--------------");
      position( 5,29); printf("1 ....... Cria arquivo");
      position( 6,29); printf("2 .. Cadastra registro");
      position( 7,29); printf("3 .... Exibe registros");
      position( 8,29); printf("4 ................ Fim");
      position(11,29); printf("Escolha a sua opcao: ");
      fflush(stdin); scanf("%d", &OPCAO);
      if (OPCAO != 4)
        {
          switch (OPCAO)
            {
              case 1  : criar();     break;
```

```
                case 2  : cadastrar(); break;
                case 3  : exibir();    break;
                default : printf("Opcao invalidade - Tecle <Enter>");
                          getchar() + scanf("Enter");
                          break;
              }
          }
      }
  return 0;
}

void criar(void)
{
  clear();
  position( 1,31); printf("Criacao de Arquivo");
  PTRARQ = fopen("NOMESTXT.DAT", "w");
  position(12,31); printf("Arquivo foi criado");
  fclose(PTRARQ);
  position(24,25); printf("Tecle <Enter> para voltar ao menu");
  getchar() + scanf("Enter");
  return;
}

void cadastrar(void)
{
  clear();
  position( 1,27); printf("Cadastramento de Registro");
  PTRARQ = fopen("NOMESTXT.DAT", "a");
  position( 5,10);
  printf("Entre um Nome ou <Enter> para voltar ao menu ..: ");
  scanf("%[^\n]", NOME);
  fputs(NOME, PTRARQ);
  fputs("\n", PTRARQ);
  fclose(PTRARQ);
  getchar() + scanf("Enter");
  return;
}

void exibir(void)
{
  int LINHA = 5;
  clear();
  position( 1,27); printf("Apresentacao de Registros");
  PTRARQ = fopen("NOMESTXT.DAT", "r");
  while (fgets(NOME, 40, PTRARQ) != NULL)
    {
      position(LINHA, 5);
      printf("%s", NOME);
      LINHA += 1; /* podera' ser I ++ */
    }
  fclose(PTRARQ);
  position(24,25); printf("Tecle <Enter> para voltar ao menu");
  getchar() + scanf("Enter");
  return;
}
```

Ao usar o programa, procure cadastrar poucos registros (ideal em torno de 15), pois não está sendo previsto exibir uma quantidade muito grande de dados na tela. Primeiramente crie o arquivo, faça alguns cadastramentos, encerre o programa e saia de seu compilador.

Depois volte, chame o programa e peça para visualizar os dados cadastrados. Se for escolhida a opção para criar o arquivo, ele será recriado, destruindo os dados já cadastrados.

A função **fclose()** é utilizada antes da apresentação da mensagem para voltar ao menu. As operações com arquivos alteram o *stream*, ou seja, carregam o *buffer*. A função **fflush()** libera esse *buffer* para que a tecla **<Enter>** possa ser devidamente interpretada pelo programa.

8.3.2 Arquivo Binário

Nada impede que sejam armazenados números em arquivos texto, porém deve-se levar em consideração que um número gravado como um caractere alfanumérico ocupa um espaço maior do que se for armazenado como do tipo binário.

A seguir são apresentadas duas rotinas que exemplificam arquivos com matrizes de uma dimensão, as quais usam os arquivos, considerando-os do tipo binário. Um arquivo binário é indicado pela forma de acesso utilizada pela instrução **fopen(**"MATINT.DBC", "wb"**)**, em que **wb** é a indicação de escrita de um arquivo binário.

A primeira rotina deve solicitar a entrada de dez valores inteiros e armazenar os valores em uma matriz para transferi-los de uma só vez para um arquivo. Depois deve ser escrita uma segunda rotina que leia os dados do arquivo, transferindo-os para uma matriz, para então apresentá-los. Os programas em questão vão tratar os arquivos como binários.

Escreva o programa seguinte e grave-o com o nome **matint01.c**. Será criado um arquivo chamado MATINT.DBC que contém dez valores inteiros. Assim que os elementos são fornecidos para a matriz A, eles são gravados no arquivo de uma só vez com a instrução **fwrite()**. Nada impede de utilizar a função **fprintf()** para gravar os dados no arquivo, porém deve-se levar em conta que essa função vai gravar um registro por vez.

```
/* Matriz com Arquivo */

#include <stdio.h>

int main(void)
{
  int I, A[10];
  FILE *PTRARQ;
  PTRARQ = fopen("MATINT.DBC", "wb");
  I = 0;
  while (I <= 9)
    {
      printf("Digite o elemento - %2d: ", I + 1);
      fflush(stdin); scanf("%d", &A[I]);
      I ++;
    }
  fwrite(A, sizeof(A), 1, PTRARQ);
  fclose(PTRARQ);
  return 0;
}
```

Observação

O uso do laço **while** para coletar os dados fornecidos no teclado é um mero exemplo. A mesma ação poderia ser utilizada com os laços **do...while** e **for()**.

Note no programa a utilização da função **fwrite()** fora do laço de repetição, a qual tem a finalidade de escrever dados maiores que um byte. Essa função usa quatro argumentos, obedecendo à seguinte sintaxe:

```
fwrite(buffer, tamanho, contador, ponteiro);
```

em que:

buffer - endereço da área de memória com os dados a serem gravados;

tamanho - tamanho do item a ser gravado;

contador - especifica o número de itens a ser gravado;

variável - é a variável do tipo ponteiro de arquivo.

Outro detalhe é a função **sizeof()** que retorna o tamanho em bytes da expressão nela indicada. No caso apresentado, a função retorna o tamanho da variável A.

Escreva o programa seguinte e grave-o com o nome **matint02.c**. Ele fará a abertura e a leitura do arquivo MATINT.DBC que contém dez valores inteiros, armazenados na memória, depois apresenta os valores armazenados.

```
/* Arquivo com Matriz */

#include <stdio.h>

int main(void)
{
  int I, A[10];
  FILE *PTRARQ;
  PTRARQ = fopen("MATINT.DBC", "rb");
  fread(A, sizeof(A), 1, PTRARQ);
  I = 0;
  while (I <= 9)
    {
      printf("\nO elemento %2d equivale a: %2d", I + 1, A[I]);
      I ++;
    }
  fclose(PTRARQ);
  return 0;
}
```

A função **fread()** é idêntica à **fwrite()**, tendo como diferença a possibilidade de ler de uma só vez todos os dados de um arquivo e transferi-los para a memória. Uma das maiores aplicações das funções **fread()** e **fwritte()** é o fato de serem usadas para ler e escrever dados definidos pelo programador, como é o caso das estruturas (**struct**).

8.4 Arquivo de Acesso Direto

Todas as funções de acesso a arquivos vistas até o momento acessam os dados de um arquivo de forma sequencial a partir de um indicador de posição corrente. Para que seja possível um acesso direto a uma determinada informação de um arquivo, utiliza-se a função **fseek()**, que retorna 0 (zero) quando completar com sucesso a operação de posicionamento, ou retorna -1 (um negativo) quando ocorrer algum erro na localização da informação. A função **fseek()** possui a seguinte sintaxe:

```
fseek(<ponteiro>, <bytes>, <origem>)
```

em que:

ponteiro - é um ponteiro para o arquivo;

bytes - são os bytes de deslocamento em relação à origem;

origem - valor inteiro que representa a posição no arquivo.

O argumento <ponteiro> é para **fseek()** um ponteiro de arquivo devolvido por uma chamada à função **fopen()**.

O argumento <bytes> é representado por um valor inteiro longo positivo ou negativo, o qual depende da necessidade de posicionar no arquivo antes ou depois da origem indicada. Devem ser levados em consideração valores que sejam coerentes, ou seja, valores que não gerem erro de posicionamento, tais como tentar se posicionar depois do último registro ou antes do primeiro.

O argumento <origem> é representado por um de três valores, a saber: 0 (zero) para indicar início do arquivo, 1 (um) para indicar a posição atual no arquivo e 2 (dois) para indicar a posição de final de arquivo.

Nos casos de ocorrer abertura de um arquivo somente para leitura "r" ou escrita "w", a posição atual é inicializada com o início do arquivo, ou seja, primeiro byte. Caso o arquivo venha a ser aberto para append "a", a posição atual considerada é o final do arquivo.

A vantagem desse tipo de acesso é a possibilidade de realizar um posicionamento rápido no arquivo, sem que seja necessário o acesso de forma sequencial.

Veja em seguida um exemplo mais profissional com arquivos com linguagem C. São criadas rotinas básicas que devem fazer o controle de um arquivo que trabalhe com o tipo **struct**.

Para este exemplo é considerado um arquivo para gerenciar o cadastro de uma escola, o qual deve possuir os campos matrícula, nome (com até 30 posições efetivas, o valor 31 corresponde ao espaço a mais para o caractere de retorno de linha "**\n**") e notas, conforme em seguida:

```
struct REG_ALUNO
  {
    int   MATRICULA;
    char  NOME[31];
    float NOTAS[4];
  };

struct REG_ALUNO ALUNO;
```

O programa executa as rotinas de cadastro e consulta, que utilizam como base a rotina de **pesquisa** sequencial, pois o arquivo não estará ordenado. O processo

de pesquisa em um arquivo é semelhante ao de uma matriz, pois será necessário percorrer todo o arquivo até localizar a informação desejada, se ela existir.

A rotina de **cadastramento** antes de inserir um novo registro deve pesquisar o arquivo para verificar se as "novas" informações já existem, evitando assim duplicidade de registros. Não existindo, deve gravar o novo registro.

A rotina de **consulta** fará uma pesquisa no arquivo a fim de localizar o registro solicitado; encontrando-o, essa rotina o apresenta para visualização.

A seguir é apresentado o programa (em formato normal sem negrito e itálico) que deve gerenciar o cadastro de alunos. Não está sendo levada em conta a série ou sala do aluno, bem como as operações de remoção de registros.

A cada rotina completa é apresentado no código de programa um breve comentário de seu funcionamento e também de algum recurso da linguagem C que venha a ser utilizado. Ao final grave o programa com o nome **geres.c** (Gerenciamento Escolar).

```
/*****************************************************************
 Chamada de  programas-fontes  externos, adicionados  ao programa no
 processo de compilacao e linkedicao.
 *****************************************************************/

#include <stdio.h>
#include <string.h>
#include "crt.h"

/*****************************************************************
 Definicao da estrutura de dados e das variaveis globais.
 *****************************************************************/

struct REG_ALUNO
  {
    int   MATRICULA;
    char  NOME[31];
    float NOTAS[4];
  };

struct REG_ALUNO ALUNO;

FILE   *ARQALU;
int    I, NR_MATRIC;
char   RESP;
char   SISTEMA[29] = "PROGRAMA DE CADASTRO ESCOLAR";
/*****************************************************************
 Programa principal com o controle de abertura ou criacao  do arquivo
 de dados, e apresentacao do menu principal.
 *****************************************************************/

void line(void);
```

```
void center(int LINHA, const char *MENSAGEM);
void clearline(void);
void tela(void);
void mostradados(void);
int pesquisa(int NUMERO);
int cadastra(void);
int consultar(void);

int main(void)
{

  /* Verifica existencia do arquivo, se nao existe, cria */

  char OPCAO;
  if((ARQALU = fopen("CADALU.DAT","rb+")) == NULL)
    {
      ARQALU = fopen("CADALU.DAT","wb+");
      fwrite(&ALUNO, sizeof(struct REG_ALUNO), 1, ARQALU);
    }
  do
    {
      clear();
      line();
      center( 2, SISTEMA);
      center( 3, "Menu de Opcoes\n");
      line();
      position( 9,26); printf("Cadastrar ..............: [1]");
      position(11,26); printf("Consultar ..............: [2]");
      position(13,26); printf("Finalizar ..............: [3]");
      position(17,26); printf("Entre com a opcao: --> ");
      scanf("%c", &OPCAO); fflush(stdin);
      switch(OPCAO)
        {
          case '1' : cadastra();  break;
          case '2' : consultar(); break;
        }
    }
  while (OPCAO != '3');
  fclose(ARQALU);
  position(23, 1);
  return 0;
}

/*****************************************************************
 Funcoes definidas pelo usuario com a finalidade de simplificar o uso
 de algumas instrucoes da linguagem.
 *****************************************************************/

/* Traca uma linha */

void line(void)
{
  int POSICAO;
  for (POSICAO = 0; POSICAO < 80; POSICAO ++)
    printf("-");
  return;
}

/* Centraliza uma mensagem no video em uma determinada linha */
```

```
void center(int LINHA, const char *MENSAGEM)
{
  int COLUNA;
  COLUNA = ((80 - strlen(MENSAGEM)) / 2);
  position(LINHA, COLUNA);
  printf(MENSAGEM);
  return;
}

/* Apresenta a tela de cadastro e consulta */

void tela(void)
{
  position(10,18); clearline();
  position(11,18); clearline();
  position(12,18); clearline();
  position(13,18); clearline();
  position(14,18); clearline();
  position(15,18); clearline();
  position(10, 1); printf("Matricula ...: ");
  position(11, 1); printf("Nome .........: ");
  position(12, 1); printf("1a. Nota ....: ");
  position(13, 1); printf("2a. Nota ....: ");
  position(14, 1); printf("3a. Nota ....: ");
  position(15, 1); printf("4a. Nota ....: ");
  return;
}

/* Mostra os dados quando da consulta ou tentativa de cadastro */

void mostradados(void)
{
  position(10,18); printf("%d", NR_MATRIC);
  position(11,18); printf("%s", ALUNO.NOME);
  for (I = 0; I < 4; I ++)
    {
      position(12 + I,18);
      printf("%5.2f", ALUNO.NOTAS[I]);
    }
  return;
}


/*********************************************************************
 Rotina de pesquisa, comum as rotinas de controle do programa.
 *********************************************************************/

int pesquisa(int NUMERO)
{
  int ACHOU = 0;
  rewind(ARQALU);
  while (!feof(ARQALU) && ACHOU != 1)
    {
      fread(&ALUNO, sizeof(struct REG_ALUNO), 1, ARQALU);
      if (NUMERO == ALUNO.MATRICULA)
        ACHOU = 1;
    }
  return ACHOU;
}
```

```
/*********************************************************************
 Rotinas de controle do programa principal.
 *********************************************************************/

/* Cadastramento dos dados */

int cadastra(void)
{
  clear();
  line();
  center( 2, SISTEMA);
  center( 3, "Modulo de Cadastramento\n");
  line();
  position( 6, 1); printf("Digite os dados abaixo:");
  do
    {
      position(22, 1); clearline();
      position(23, 1); clearline();
      printf("Digite [0] para voltar ao menu");
      tela();
      position(10,18); fflush(stdin); scanf("%d%*c", &NR_MATRIC);
      position(23, 1); clearline();
      if (NR_MATRIC != 0)
        {
          if (pesquisa(NR_MATRIC) == 1)
            {

              /* Apresenta os dados caso exista no arquivo */

              mostradados();
              position(22, 1); printf("Este registro ja existe");
              position(23, 1);
              printf("Tecle <Enter> para nova tentativa");
              getchar();
            }
          else
            {

              /* Localiza posicao para gravar registro */

              fseek(ARQALU, (long) sizeof(struct REG_ALUNO), 2);

              /* Grava registro */

              position(11,18);
              fflush(stdin); fgets(ALUNO.NOME, 31, stdin);
              for(I = 0; I < 4; I ++)
                {
                  position(12 + I,18);
                  fflush(stdin); scanf("%f", &ALUNO.NOTAS[I]);
                }
              ALUNO.MATRICULA = NR_MATRIC;
              fwrite(&ALUNO, sizeof(struct REG_ALUNO), 1, ARQALU);
              position(23, 1);
              printf("Tecle <Enter> para novo cadastro");
              getchar();
            }
        }
    }
  while (NR_MATRIC != 0);
```

```
  return 0;
}

/* Consulta dos dados */

int consultar(void)
{
  clear();
  line();
  center( 2, SISTEMA);
  center( 3, "Modulo de Consulta\n");
  line();
  position( 6, 1); printf("Digite o numero de matricula:");
  do
    {
      position(22, 1); clearline();
      position(23, 1); clearline();
      printf("Digite [0] para voltar ao menu");
      tela();
      position(10,18); fflush(stdin); scanf("%d%*c", &NR_MATRIC);
      position(23, 1); clearline();
      if (NR_MATRIC != 0)
        {
          if (pesquisa(NR_MATRIC) == 1)
            {

              /* Apresenta os dados caso exista no arquivo */
              mostradados();
              position(23, 1);
              printf("Tecle <Enter> para nova tentativa");
              getchar();
            }
          else
            {
              position(22, 1);
              printf("Este registro nao esta cadastrado");
              position(23, 1);
              printf("Tecle <Enter> para nova tentativa");
              getchar() + scanf("Enter");
            }
        }
    }
  while (NR_MATRIC != 0);
  return 0;
}
```

O programa anterior utiliza, de certa forma, todos os detalhes apresentados nesta obra. Observe o uso das funções definidas internamente, tais como **line()**, **center()**, **tela()**, **clearline()**, **mostradados()**, **pesquisa()**, **cadastra()** e **consultar()**.

O arquivo de cabeçalho **string.h** é indicado devido à função **strlen()** existente na rotina **center()**.

Na rotina **pesquisa()** é utilizada a função **feof()** pertencente ao arquivo de cabeçalho **stdio.h**, a qual retorna o valor verdadeiro (valor não zero) caso o final de arquivo tenha sido alcançado. Essa função é ideal para trabalhar com arquivos binários. Na rotina é solicitado para ler o arquivo enquanto não for fim de arquivo. Há também a função **rewind()** que posiciona o ponteiro do arquivo em seu início.

Na rotina **cadastra()** é utilizada para a entrada do valor do número de matrícula do aluno a linha de código "**scanf("%d%*c", &NR_MATRIC);**", em que o formato "**%d**" determina que a entrada será um valor numérico do tipo **int** (decimal). O formato "**%*c**" é uma forma usada para capturar na entrada do valor numérico o caractere de retorno de linha "**\n**" e excluí-lo do valor fornecido quando a tecla **<Enter>** é acionada. Desta forma, torna-se o *buffer* de entrada limpo, sendo então possível a execução da linha "**fgets(ALUNO.NOME, 31, stdin);**". Caso não se use o formato "**%*c**" na função **scanf()** antes da função **fgets()**, haverá problemas de parada para a entrada da função **fgets()**.

O símbolo de asterisco (*****) usado no formato "**%*c**" estabelece que o próximo dado será descartado. Neste caso, o próximo a ser descartado é um dado caractere devido ao formato "**c**".

O formato "**%d%*c**" para a função **scanf()** faz com que apenas o valor numérico inteiro seja guardado na variável, além de permitir o uso independente da função **getchar()**, neste caso sem a concatenação com a função **scanf()**, como em "**getchar() + scanf("Enter");**".

Na entrada de dado de um nome de aluno não exceda os trinta caracteres previstos, que ocasiona funcionamento insatisfatório do programa, pois não está sendo prevista essa ocorrência por parte do programa. Fica o desafio para implementação de um recurso que limite a entrada do usuário em 30 caracteres.

Estude atentamente cada rotina do programa para perceber os detalhes utilizados, e não se esqueça do driver ANSI devidamente instalado para que as operações de tela tenham sucesso nas suas ações.

Exercícios

Utilizando arquivos binários, construa em linguagem C os seguintes programas:

a. Ler cinco valores numéricos inteiros em uma matriz A de uma dimensão. Após a leitura os dados devem ser armazenados em um arquivo binário.

b. Elaborar um programa que leia os cinco valores numéricos armazenados no arquivo criado no exercício anterior. Depois de os valores estarem armazenados na matriz, o programa deve apresentar a soma dos elementos de índice ímpar.

c. Elaborar um programa que leia dez valores numéricos inteiros para uma matriz A de uma dimensão. Construir uma matriz B de uma dimensão, sendo cada elemento de B o quadrado correspondente do elemento de A, e gravar a matriz B em um arquivo. Ao final transferir os dados gravados no arquivo para a matriz C e apresentá-los a partir da matriz C.

d. Elaborar um programa que efetue a leitura de duas matrizes A e B de duas dimensões com vinte elementos (4x5). Construa uma matriz C que será a soma elemento a elemento das matrizes A e B. A matriz C tem a mesma dimensão das matrizes A e B. Gravar a matriz C em um arquivo binário. Em seguida o programa deve ler os dados do arquivo e transferi-los para a matriz D que será então apresentada.

e. Ler cinco valores numéricos reais em uma matriz A de uma dimensão. Após a leitura os dados devem ser armazenados em um arquivo binário.

f. Elaborar um programa que leia os cinco valores numéricos armazenados no arquivo criado no exercício anterior e apresente-os em seguida.

g. Ler oito valores numéricos reais para as matrizes A e B (cada matriz deve conter oito elementos). Os dados da matriz A devem ser gravados em um arquivo denominado ARQ1.DBC e os dados da matriz B em um arquivo denominado ARQ2.DBC. Observe que é necessário trabalhar com dois arquivos simultaneamente.

h. Escrever um programa que leia os arquivos ARQ1.DBC e ARQ2.DBC e transfira, respectivamente, seus oito elementos reais para as matrizes X e Y. Construir uma matriz Z que será a soma dos elementos das matrizes X e Y. Apresentar os dados da matriz X, Y e Z.

Anotações

Bibliografia

FARRER, H. **Algoritmos Estruturados**. Rio de Janeiro: Guanabara, 1986.

FORBELLONE, A. L. V.; EBERSPÄCHER, H. F. **Lógica de Programação:** *A Construção de Algoritmos e Estruturas de Dados.* São Paulo: Makron, 1993.

HADDAD, R. I. **C#:** *Aplicações e Soluções.* São Paulo: Érica, 2001.

INSTITUTO BRASILEIRO DE PESQUISA EM INFORMÁTICA. **Dominando a Linguagem C**. IBPI Press, 1993.

KNUTH, D. E. **The Art of Computer Programming:** *Fundamental Algorithms*. Massachusetts: Addison-Wesley, vol 1, 1972.

MANZANO, J. A. N. G.; OLIVEIRA, J. A. **Algoritmos:** *Lógica para Desenvolvimento de Programação de Computadores.* São Paulo: Érica, 1996.

MATEUS, C. A. **C++ Builder 5:** *Guia Prático.* São Paulo: Érica, 2000.

MIZHARI, V. V. **Treinamento em Linguagem C:** *Módulo 1.* São Paulo: Makron, 1990.

_____. **Treinamento em Linguagem C:** *Módulo 2.* São Paulo: Makron, 1990.

MÓDULO CONSULTORIA E INFORMÁTICA. **Linguagem C:** *Programação e Aplicações.* Livros Técnicos e Científicos, 1989.

SCHILDT, H. **C:** *Completo e Total.* 3. ed. rev. atual. São Paulo: Makron, 1996.

Marcas Registradas

Borland, Tubo C, Turbo C++, Borland C++ são marcas registradas da Borland Inc.

Microsoft, MS-DOS, Windows, Quick C são marcas registradas da Microsoft Corporation.

Digital Mars é marca registrada da Digital Mars.

Bloodshed e Dev _C++ são marcas registradas da Bloodshed.

GCC e Mingw são produtos freeware submetidos à licença GNU.

Todos os demais nomes registrados, marcas registradas ou direitos de uso citados neste livro pertencem aos seus respectivos proprietários.