

Sistema Aberto de Educação



Guia de Estudo

Linguagem e Técnicas de Programação II



Instituição Credenciada pelo MEC
Centro Universitário do Sul de Minas



SABE – Sistema Aberto de Educação

**Av. Cel. José Alves, 256 - Vila Pinto
Varginha - MG - 37010-540
Tele: (35) 3219-5204 - Fax - (35) 3219-5223**

Instituição Credenciada pelo MEC – Portaria 4.385/05

**Centro Universitário do Sul de Minas - UNIS/MG
Unidade de Gestão da Educação a Distância – GEaD**

**Mantida pela
Fundação de Ensino e Pesquisa do Sul de Minas - FEPESMIG**

Varginha/MG

Todos os direitos desta edição reservados ao Sistema Aberto de Educação – SABE.
É proibida a duplicação ou reprodução deste volume, ou parte do mesmo, sob
qualquer meio, sem autorização expressa do SABE.

005.1

S232g. SANT' ANA, Tomás Dias.

Guia de Estudos – Linguagem e Técnicas de
Programação II – Tomás Dias Sant' Ana.
Varginha: GEaD-UNIS/MG, 2008.

82 p.

1. Linguagem de Programação. 2. Lógica. 3.
Linguagem C I. Título.

REITOR
Prof. Ms. Stefano Barra Gazzola

GESTOR
Prof. Ms. Tomás Dias Sant' Ana

Supervisor Técnico
Prof. Ms. Wanderson Gomes de Souza

Coord. do Núcleo de Recursos Tecnológicos
Prof^a. Simone de Paula Teodoro Moreira

Coord. do Núcleo de Desenvolvimento Pedagógico
Prof^a. Vera Lúcia Oliveira Pereira

Revisão ortográfica / gramatical
Prof^a. Maria José Dias Lopes Grandchamp






Design/diagramação
Prof. César dos Santos Pereira

Equipe de Tecnologia Educacional
Prof^a. Débora Cristina Francisco Barbosa
Jacqueline Aparecida Silva
Prof. Lázaro Eduardo da Silva

Autor
TOMÁS DIAS SANT' ANA

Tecnólogo em Processamento de Dados (1997) e Bacharel em Ciência da Computação (1998) – UNIFENAS, Alfenas - MG. Mestre em Ciência da Computação (2001) pela USP, São Carlos – SP. Pós-Graduando em Negócios para Executivos pela FGV – São Paulo, SP. Foi coordenador Geral de Pós-Graduação e atualmente ocupa o cargo de Gestor da Unidade de Educação a Distância - GEaD do Centro Universitário do Sul de Minas - UNIS, Varginha - MG. Atua também como coordenador do Curso de Pós-Graduação *Lato Sensu* em Redes de Computadores e é professor em cursos de Graduação e Pós-Graduação, presencial e a distância, neste mesmo Centro Universitário.

TABELA DE ÍCONES

	Realize. Determina a existência de atividade a ser realizada. Este ícone indica que há um exercício, uma tarefa ou uma prática para ser realizada. Fique atento a ele.
	Pesquise. Indica a exigência de pesquisa a ser realizada na busca por mais informação.
	Pense. Indica que você deve refletir sobre o assunto abordado para responder a um questionamento.
	Conclusão. Todas as conclusões, sejam de idéias, partes ou unidades do curso virão precedidas desse ícone.
	Importante. Aponta uma observação significativa. Pode ser encarado como um sinal de alerta que o orienta para prestar atenção à informação indicada.
	Hiperlink. Indica um link (ligação), seja ele para outra página do módulo impresso ou endereço de Internet.
	Exemplo. Esse ícone será usado sempre que houver necessidade de exemplificar um caso, uma situação ou conceito que está sendo descrito ou estudado.
	Sugestão de Leitura. Indica textos de referência utilizados no curso e também faz sugestões para leitura complementar.
	Aplicação Profissional. Indica uma aplicação prática de uso profissional ligada ao que está sendo estudado.
	Checklist Ou Procedimento. Indica um conjunto de ações para fins de verificação de uma rotina ou um procedimento (passo a passo) para a realização de uma tarefa.
	Saiba Mais. Apresenta informações adicionais sobre o tema abordado de forma a possibilitar a obtenção de novas informações ao que já foi referenciado.
	Revendo. Indica a necessidade de rever conceitos estudados anteriormente.

Índice de Figuras

Figura 1 – Processo de Compilação e Execução de um programa.....	11
--	----

Índice de Tabelas

Tabela 1 – Tipos de Linguagem de Programação.....	9
Tabela 2 – Palavras Chaves da Linguagem C	14
Tabela 3 – Tipos Básicos de Dados em C.....	17
Tabela 4 – Exemplo: Modificadores de Tipo.....	17
Tabela 5 – Comandos de formato do <i>printf</i>	25
Tabela 6 – Comandos de barra invertida	27
Tabela 7 – Comandos de barra invertida	30
Tabela 8 – Precedência dos Operadores Aritméticos	31
Tabela 9 – Tabela Verdade dos Operadores Lógicos	31
Tabela 10 – Precedência dos Operadores Relacionais e Lógicos	32
Tabela 11 – Operadores Bit a Bit	32
Tabela 12 – Funções de manipulação de Arquivos.....	66
Tabela 13 – Modo de abertura de arquivo.....	68

Sumário

Apresentação	7
1. FUNDAMENTOS DA LINGUAGEM C	8
1.1. Linguagem de Programação	8
1.2. Linguagem C	13
1.3. Tipos de Dados	16
1.4. Variáveis	17
1.5. Exercícios	21
2. COMANDOS BÁSICOS EM C	22
2.1. Entrada e Saída pelo Console	22
2.2. Constantes	27
2.3. Operadores	29
2.4. Expressões	33
2.5. Exercícios	34
3. Estruturas Básicas de Controle	37
3.1. Comando de Seleção	37
3.2. Comando de Iteração	40
3.3. Comandos de Desvio	43
3.4. Matrizes e Strings	44
3.5. Exercícios	48
4. Estruturas Avançadas de Controle	50
4.1. Ponteiros	50
4.2. Funções	52
4.3. Estruturas	56
4.4. Alocação Dinâmica de Memória	60
4.5. Exercícios	64
5. Mecanismos de Entrada e Saída Avançados	66
5.1. Entrada e Saída com Arquivo	66
5.2. Entrada e Saída com Modo Gráfico	74
5.3. Exercícios	81
REFERÊNCIAS	82

Apresentação

“... Não busque tarefas grandiosas e de evidência. Procure dar conta integralmente do serviço pequenino que lhe foi confiado. Da perfeição com que executar dependerá sua oportunidade para receber uma incumbência maior.”

C. Torres Pastorino, Minutos de Sabedoria, 1988

Prezado(a) aluno(a):

Este é o seu Guia de Estudos da disciplina de Linguagem e Técnicas de Programação II, ministrada para o curso de Bacharelado em Sistemas de Informação pelo Centro Universitário do Sul de Minas – UNIS-MG, através do Sistema Aberto de Educação - SABE. O Guia irá auxiliá-lo no desenvolvimento das competências necessárias para a construção lógica de programas de computador utilizando a Linguagem de Programação C.

Cada disciplina do seu curso tem características e importância peculiares. Especificamente a disciplina de Linguagem e Técnicas de Programação II irá conciliar as técnicas desenvolvidas nas disciplinas de Algoritmos e Técnicas de Programação I e, apresentará novas técnicas de programação, utilizando a Linguagem de Programação C. Com o auxílio deste Guia de Estudos, você irá aprender os passos básicos, a lógica e as técnicas básicas e avançadas para construção programas de computador em C.

Esses desafios não são simples e, para superá-los, você deverá acompanhar as atividades e interações no nosso Ambiente Virtual de Aprendizagem. Seu primeiro desafio é encarar a produção de programas de frente e superar as dificuldades naturais. Assim como acontece durante o estudo de Algoritmo e Técnicas de Programação I, muitos ex-alunos consideram essa disciplina um “bicho de sete cabeças”; como também sou ex-aluno da disciplina, posso afirmar que é apenas falta de empenho. Para ter sucesso, você deve testar todos os exemplos, realizar as atividades e interagir com os colegas e também com o professor.

O desafio está lançado!

Tomás Dias Sant’ Ana

Professor de Linguagem e Técnicas de Programação II

1. FUNDAMENTOS DA LINGUAGEM C

Em Algoritmos, você aprendeu técnicas e padrões de resolução de problemas. Para que esses problemas sejam resolvidos pelo computador, é necessário que sejam escritos em uma **Linguagem de Programação**.

As principais técnicas de programação foram aplicadas na disciplina de Linguagem e Técnicas de Programação I com a Linguagem de Programação Pascal. Nesta unidade você poderá visualizar os conceitos básicos de Linguagem de Programação C, incluindo variáveis, constantes e também os tipos de dados.

1.1. Linguagem de Programação

Como vimos na disciplina de Linguagem e Técnicas de Programação I a **linguagem de programação** é o meio pelo qual podemos indicar os “passos” que devem ser realizados pelo computador para resolver problemas. Uma linguagem de programação possui um conjunto de símbolos (comandos, identificadores, caracteres ASCII, etc.) e regras de sintaxe que permitem a construção de sentenças que descrevem de forma precisa ações compreensíveis e executáveis para o computador.





LINGUAGEM DE PROGRAMAÇÃO = SÍMBOLOS + REGRAS DE SINTAXE

Utilizando as linguagens de programação, colocamos algoritmos numa forma que o computador possa interpretá-los, ou seja, na forma de programas computacionais. Para que o computador execute o algoritmo proposto, as operações devem ser transcritas para uma linguagem que a máquina consiga compreender. Na realidade, os computadores só podem executar algoritmos expressos em **linguagem de máquina** que constitui-se de um conjunto de instruções capazes de ativar diretamente os dispositivos eletrônicos do computador (FARRER, 1999).

Consequentemente, a Linguagem de Máquina, é uma linguagem de difícil aprendizado e pouco expressiva para as pessoas. Para tornar a atividade de programação mais acessível, foram desenvolvidas outras linguagens, denominadas de “**Linguagens de Programação**”, que funcionam como uma forma alternativa de se comunicar com o computador (FARRER, 1999).

Uma linguagem de programação é uma notação formal para descrição de algoritmos que serão executados por um computador. Como todas as notações formais, uma linguagem de programação tem dois componentes: **Sintaxe e Semântica**.

	A sintaxe consiste em um conjunto de regras formais, que especificam a composição de programas a partir de letras, dígitos e outros símbolos. Por exemplo, regras de sintaxe podem especificar que cada parêntese aberto em uma expressão aritmética deve corresponder a um parêntese fechado, e que dois comandos quaisquer devem ser separados por um ponto-e-vírgula.
---	---

	As regras de semântica especificam o “significado” de qualquer programa, sintaticamente válido, escrito na linguagem.
---	--

Existem diversas linguagens de programação, cada uma com suas características específicas e com níveis de complexidade e objetivos diferentes, como pode ser visto na Tabela 1 abaixo.

Linguagem	Características
Linguagem de Máquina	É específica para cada computador.
Linguagem de Baixo Nível	Utiliza mnemônicos para representar instruções elementares. Exemplo: Assembly.
Linguagem de Alto Nível	Utiliza instruções próximas da linguagem humana buscando facilitar o desenvolvimento dos programas. Exemplos: Pascal, C, Cobol, Clipper.

Tabela 1 – Tipos de Linguagem de Programação

- **Linguagem de Programação de Baixo Nível:** Conhecida como Linguagem Assembler ou Linguagem de Montagem, ou ainda, Linguagem Simbólica. Utiliza números binários, hexadecimais, alguns símbolos e letras para compor os programas. Está muito próxima da Linguagem de Máquina, na qual cada instrução simbólica corresponde, praticamente, a uma instrução de máquina. Para transformar o programa escrito em Linguagem Assembler em código de máquina executável, é utilizado um programa-suporte denominado de MONTADOR.
- **Linguagens de Programação de Alto Nível:** São linguagens de programação que utilizam notações matemáticas e grupos de palavras para representar as instruções de máquina, tornando o processo de programação mais próximo do entendimento humano. Muitas destas linguagens foram desenvolvidas para atender os problemas de áreas de aplicação específicas, como, por exemplo, linguagens para aplicações comerciais, científicas, administrativas, de ensino, etc (FARRER, 1999).

Embora seja teoricamente possível a construção de computadores especiais, capazes de executar programas escritos em uma linguagem de programação qualquer, os computadores existentes hoje em dia são capazes de executar somente programas em Linguagem de Máquina.

Linguagens de Máquina são projetadas levando-se em conta os seguintes aspectos:

- rapidez de execução de programas;
- custo de sua implementação; e
- flexibilidade com que permitem a construção de programas de nível mais alto.

Por outro lado, linguagens de programação de alto nível são projetadas em função de :

- facilidade de construção de programas; e
- confiabilidade dos programas.



O **problema** é: como a linguagem de nível mais alto pode ser implementada em um computador cuja linguagem é bastante diferente e de nível mais baixo?

SOLUÇÃO: Por meio da tradução de programas escritos em linguagens de alto nível para a linguagem de baixo nível do computador.

Para executarmos um programa escrito numa linguagem de alto nível, é preciso primeiro traduzir o código-fonte para o código-objeto. O processo de tradução pode dar-se em tempo de execução caso a linguagem use um interpretador (traduz e executa instrução a instrução), ou todas as instruções podem ser traduzidas antes que se inicie a execução do programa, o que ocorre no caso de linguagens que usam tradutores do tipo compilador (FARRER, 1999).

EDIÇÃO	COMPLAÇÃO	LINK-EDIÇÃO
ALGORITMO \Rightarrow CÓDIGO-FONTE	\Rightarrow CÓDIGO-OBJETO	\Rightarrow PROGRAMA EXECUTÁVEL

Figura 1 – Processo de Compilação e Execução de um programa

- **Compilador:** é um programa que traduz todo o **código-fonte** de programas escritos numa linguagem de alto nível em **código-objeto** antes da execução do programa. O código-objeto é o código de máquina, ou alguma variação do código de máquina.
- **Código-fonte:** não é executável diretamente pelo processador - permite apenas que o programador consiga definir o programa em uma forma legível aos humanos.
- **Código-objeto:** é o código produzido pelo compilador; é uma forma intermediária, similar à linguagem de máquina do computador. Apesar de estar representado em binário, não é executável diretamente pelo processador, pois, normalmente, o código-objeto referencia partes de programa que não estão necessariamente definidas no mesmo arquivo que o gerou, por exemplo, arquivos de bibliotecas de sub-rotinas.
- **Editor de ligação (ou link-editor):** Um programa que reúne módulos compilados e arquivos de dados para criar um programa executável.

Possibilidades de Erros no Programa:

- Erros de Compilação: erros de digitação e de uso da sintaxe da linguagem.
- Erros de Link-Edição: erro no uso de bibliotecas de sub-programas necessárias ao programa principal.
- Erros de Execução: erro na lógica do programa (semântica).

CrITÉRIOS de Qualidade de um Programa

Vejamos alguns critérios para escrevermos um programa com qualidade:

- **Integridade:** os resultados gerados pelo processamento do programa devem estar corretos, caso contrário o programa simplesmente não tem sentido;
- **Clareza:** refere-se à facilidade de leitura do programa. Se um programa for escrito com clareza, deverá ser possível a outro programador seguir a lógica do programa sem muito esforço, assim como o próprio autor do programa entendê-lo após ter estado um longo período afastado dele;
- **Simplicidade:** a clareza e precisão de um programa são normalmente melhoradas tornando seu entendimento o mais simples possível, consistente com os objetivos do programa;
- **Eficiência:** refere-se à velocidade de processamento e a correta utilização da memória. Um programa deve ter performance SUFICIENTE para atender às necessidades do problema e do usuário, bem como deve utilizar os recursos de memória de forma moderada, dentro das limitações do problema;
- **Modularidade:** consiste no particionamento do programa em módulos menores bem identificáveis e com funções específicas, de forma que o conjunto desses módulos e a interação entre eles permita a resolução do problema de forma mais simples e clara; e
- **Generalidade:** é interessante que um programa seja tão genérico quanto possível de forma a permitir a reutilização de seus componentes em outros projetos.

1.2. Linguagem C

A linguagem C foi inventada e implementada por *Dennis Ritchie*, resultado do desenvolvimento de uma linguagem mais antiga chamada BCPL:

Linguagem BCPL → Linguagem B → Linguagem C

A linguagem C e o Sistema Operacional UNIX surgiram praticamente juntos e um ajudou o desenvolvimento do outro. A versão K&R (*Brain Kernigan e Dennis Ritchie*) já era fornecida junto com o Sistema Operacional UNIX versão 5.

Existem várias implementações de C, mas para garantir compatibilidade entre estas implementações foi criado um padrão ANSI para C. ANSI é a abreviatura de *American National Standard Institute*, trata-se de um instituto americano que se encarrega da formulação de normas em diversos setores técnicos, incluindo os relativos aos computadores.

Mais recentemente, foi lançada uma versão C orientada a objeto, o C++ (inicialmente chamado C com classes). Esta versão continua vinculada ao padrão ANSI, mas inclui ferramentas novas que facilitam o desenvolvimento de grandes sistemas usando C.

Principais Características da Linguagem C:

- C é uma linguagem portátil, ou seja, um programa desenvolvido para uma máquina pode facilmente migrar para outra sem muita alteração.
- O código gerado para um programa C é mais resumido (otimizado).
- C tem somente 32 palavras chaves (27 do padrão K&R e 5 do padrão ANSI).
- C é uma linguagem estruturada.
- Não admite declaração de função dentro de função.
- Admite malhas (laços) de repetição como for.
- Admite que você indente os comandos.
- Admite blocos de comandos (comandos compostos).

- Existem versões compiladas e interpretadas de C, as versões compiladas são mais rápidas mas as versões interpretadas são preferidas quando se quer depurar um programa.
- As palavras chaves de C:

auto	double	int	struct	typedef	static
break	else	long	switch	char	while
case	enum	register	extern	return	
union	const	float	short	unsigned	
continue	for	signed	void	default	
goto	sizeof	volatile	do	if	

Tabela 2 – Palavras Chaves da Linguagem C

- Todas as palavras chaves de C são minúsculas. A linguagem C faz distinção entre palavras maiúsculas e minúsculas:

	else <> ELSE
--	---------------------------


- Um programa em C é composto de uma ou mais funções. Todo programa em C tem a função main () que é o corpo principal do programa.

Estrutura Básica de um programa em C:

Todo programa em C segue uma estrutura básica, conforme o esquema apresentado abaixo:

```
[inclusão de bibliotecas]
[declaração de constantes e variáveis globais]
[declaração dos protótipos de funções]
void main (void)
{
    [corpo do programa principal]
}
[implementação das funções do programa]
```

Um programa exemplo:

	<pre>/* Exemplo 1 - Um programa bem simples */ #include <stdio.h> //Biblioteca - entrada e saída vídeo void main (void) { puts ("Bem vindo a linguagem C"); }</pre>
---	---


Explicação:

- A execução deste programa faz a apresentação do texto: Bem vindo a linguagem C.
- Na primeira linha temos o uso da macro **#include <stdio.h>** que faz a inclusão de algumas funções de biblioteca utilizadas para ler e escrever (in/out).
- O corpo principal do programa está organizado dentro da função **main**. Esta função deve existir em todo programa C completo e é a partir desta função que começa a ser executado o programa C.
- A palavra **void** antes de **main** e entre parênteses, significa que a função **main** não devolve valor, nem espera valores (não tem parâmetros).
- Dentro da função **main** está sendo usada a função utilizada para escrever strings: **puts** e dentro o valor string que deve ser escrito. A função **puts** tem seu protótipo definido dentro do arquivo de cabeçalho **stdio.h**, daí a necessidade de fazer a inclusão destas funções neste programa.

Comentários

Comentários são textos escritos dentro do código-fonte para explicar ou descrever alguns aspectos relativos a ele. Os comentários podem ser colocados em qualquer lugar do programa onde exista um espaço em branco.

Você pode colocar comentários de duas formas: comentário de linha “//” ou comentário de bloco “/* .. */”. Quando o compilador encontra o símbolo “//”, ele salta todos os caracteres daquela linha. Da mesma forma, todos os caracteres que seguem “/*” são pulados até ser detectado o símbolo “*/”.

	<pre>//Comentário de Linha #include <stdio.h> //Biblioteca - entrada e saída vídeo void main (void) { /* No comentário de bloco podem existir várias Linhas */ puts ("Bem vindo a linguagem C"); }</pre>
---	---

1.3. Tipos de Dados

Os dados são representados pelas informações a serem processadas por um computador. Um tipo de dado especifica as características, ou seja, os valores e operações possíveis de serem utilizados com um dado desse tipo. Toda variável e constante usada em um programa tem um tipo associado a ela.

Você deve saber quais são os tipos de dados que a linguagem de programação oferece, qual o tamanho ocupado por cada um desses tipos e como você pode atribuir um nome a uma variável. A idéia de escopo, que é o local de validade de uma variável, também é importante compreender.

O endereço de uma variável geralmente é definido pelo compilador, e, na maioria das linguagens, não pode ser definido ou facilmente modificado. Portanto, não é necessário se preocupar com isso no momento. Porém, em programas mais complexos, é necessário conhecer a localização de determinadas variáveis ou dados na memória.

Alguns desses atributos são (e devem ser) definidos no momento em que o programador escreve o programa: é o caso do nome, do tipo e do escopo de uma variável. Nesse caso, diz-se que essa definição é **estática**, pois não muda. Já outros são (ou podem ser) definidos no momento em que o programa é executado. Nesse caso, diz-se que a definição desses atributos é **dinâmica**, pois muda (ou pode mudar) a cada execução ou momento de execução, como é o caso do endereço e do valor de uma variável).

Os atributos estáticos de uma variável são definidos na programação, e a esse ato dá-se o nome de declaração. **Declarar uma variável** significa especificar como a variável se chamará no código-fonte do programa que você está desenvolvendo e definir qual o tipo dos dados que ela pode armazenar e manipular. Cada linguagem possui uma forma de fazer isso. Vamos agora estudar quais são os tipos de dados que a linguagem C oferece. Há 5 (cinco) tipos básicos de dados em C:

Tipo	Tamanho	Explicação
char	1 byte	Caracter
int	2 bytes	Inteiro
float	4 bytes	ponto flutuante (real)
double	8 bytes	ponto flutuante com precisão dupla
void	0 bytes	ausência de tipo

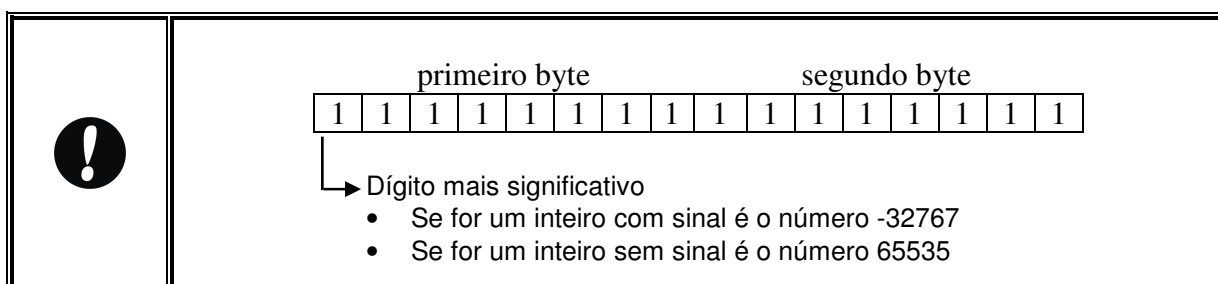
Tabela 3 – Tipos Básicos de Dados em C

A Linguagem C possui modificadores de tipo. Um modificador de tipo é usado para alterar o significado de um tipo, são eles: signed, short, long e unsigned.

Tipo	Tamanho	Faixa mínima
char	1 byte	-128 a 127
unsigned char	1 byte	0 a 255
signed char	1 byte	-128 a 127
int	2 bytes	-32768 a 32767
unsigned int	2 bytes	0 a 65535
long int	4 bytes	-2.147.483.648 a 2.147.483.647

Tabela 4 – Exemplo: Modificadores de Tipo

Observação: A diferença entre inteiros com sinal (signed) e sem sinal (unsigned) é a maneira como o bit mais significativo é interpretado. Representação esquemática:



1.4. Variáveis


Antes de estudarmos as variáveis, precisamos nos lembrar dos **identificadores**. Os identificadores são nomes a serem dados a variáveis, tipos definidos, procedimentos, funções e constantes.


Devem seguir as seguintes regras de construção:

- inicia sempre com uma letra ou um *underscore* (_);
- os caracteres subsequentes devem ser letras, número ou *underscore* (_);
- letras maiúsculas e minúsculas são tratados diferentemente em C;

- não pode ser igual a uma palavra-chave e não deve ser igual a um nome de função de um programa.

Atenção para o fato de que identificadores muito longos são mais fáceis de serem lidos pelas pessoas quando se usa uma mistura de letras maiúsculas e minúsculas; por exemplo, SalarioMinimo é mais fácil de se ler do que SALARIOMINIMO.



	Exemplo de Identificadores válidos: contador teste23 balanco_alto _variavel
---	--

	Exemplo de Identificadores INVÁLIDOS: %Quantidade - O símbolo % não é permitido 4Vendedor - Não pode começar com um número Soma Total - Não pode ter espaços entre as letras
---	--

Observação: Um identificador deverá ficar inteiramente contido em uma linha do programa, ou seja, você não pode começar a digitar o nome do identificador numa linha e acabar em outra.



Variáveis

Variável, no sentido de programação, é uma região previamente identificada, que tem por finalidade armazenar informações (dados) de um programa temporariamente. Uma variável armazena apenas um valor por vez. Sendo considerado como valor o conteúdo de uma variável; este valor está associado ao tipo de dado da variável.

	Sintaxe <code>tipo lista_de_variáveis;</code> onde: tipo é um dos tipos pré-definidos ou um tipo definido pelo usuário.
	<code>int i,j,l;</code> <code>double total, mediageral;</code>

As variáveis podem ser inicializadas, por exemplo, a variável Soma que deve receber a soma de vários números deverá ser inicializada com o valor zero (0), já a variável Multiplica, que receberá a multiplicação de alguns números deverá ser inicializada com o valor um (1).

Para inicializar uma variável deve ser utilizada a seguinte sintaxe:

	Sintaxe: tipo nome_da_variavel = constante;
	<pre>char ch = 'a'; int soma = 0; float valor = 12.56;</pre>

As variáveis ch, soma e valor são declaradas e já recebem um valor inicial.

Como vimos na disciplina de Linguagem e Técnicas de Programação I, as variáveis são de 3 tipos dependendo do local em que foram declaradas:

- Variável local
- Variável formal (parâmetro formal)
- Variável global


a) Variável Local:


É uma variável que só existe dentro do bloco no qual ela foi criada. Um bloco (comando composto) em C começa com { (abre chave) e termina com } (fecha chave).

	<pre>void main (void) { int i; }</pre>
---	--

b) Parâmetro Formal:


São os parâmetros passados para uma função que usa parâmetros (veremos com mais detalhes no item 4.2 da unidade 4). São tratados como variáveis locais a função e sua declaração é feita depois do nome da função e dentro dos parênteses.


	<pre>/* Exemplo 2 - Parametro formal */ #include <stdio.h> void EscreveCaracter (char c) { putchar (c); } main () { EscreveCaracter ('a'); }</pre>
---	--

	<ul style="list-style-type: none">• Se o tipo dos parâmetros formais forem diferentes dos tipos dos argumentos passados para a função o programa em C não vai parar, mas isto pode incorrer em ERRO.• C não verifica o número de argumentos passados para a função.• A função de biblioteca putchar é utilizada para escrever um caracter.
---	---

c) Variáveis Globais:

São as variáveis que estão disponíveis ao programa inteiro, qualquer bloco do programa pode acessá-las sem erro. Estas variáveis devem ser declaradas fora de todas as funções e no começo do programa principal.

	<pre>/* Exemplo 3 - Variavel global */ #include <stdio.h> char c; /* Declaracao da variavel global */ main () { c = 'a'; putchar (c); }</pre>
---	--

	<ul style="list-style-type: none">• O uso de variáveis globais deve ser evitado porque estas ocupam espaço na memória durante toda a execução do programa.• Deve ser explorado o uso de funções parametrizadas para melhorar a organização e a estruturação de programas em C.
---	---

1.5. Exercícios

1. Identificar o tipo de cada constante abaixo:

- a) 'a'
- b) 100.50
- c) 5
- d) 0.35
- e) '5'

2. Assinalar com um X os identificadores válidos em C:

- | | | |
|---------------------------------|--|----------------------------------|
| <input type="checkbox"/> Valor | <input type="checkbox"/> Salario-Liquido | <input type="checkbox"/> B248 |
| <input type="checkbox"/> X2 | <input type="checkbox"/> Nota Aluno | <input type="checkbox"/> A1B2C3 |
| <input type="checkbox"/> 3 x 4 | <input type="checkbox"/> Maria | <input type="checkbox"/> KM/H |
| <input type="checkbox"/> XYZ | <input type="checkbox"/> NomeDaEmpresa | <input type="checkbox"/> Sala215 |
| <input type="checkbox"/> "Nota" | <input type="checkbox"/> Ah! | <input type="checkbox"/> M{A} |

3. Descreva o tamanho (em bits) e a faixa mínima para cada tipo de dados abaixo.

```
char    ->  
int     ->  
float   ->  
double  ->  
void    ->
```

2. COMANDOS BÁSICOS EM C

Neste capítulo, você vai aprender os comandos básicos da linguagem C, entre eles: comandos de entrada e saída pelo console, constantes, operadores e expressões. Esses comandos são essenciais para construção de programas em C.

2.1. Entrada e Saída pelo Console

Como vimos em Algoritmo, à atribuição não é a única forma de atribuímos um valor a uma variável; para permitir que o usuário entre com dados durante a execução do programa, fazemos uso do comando **leia**.

Na linguagem C, temos os **comandos de entrada** e os **comandos de saída**, que são chamados comandos de I/O (*Input/Output* – Entrada/Saída). As entradas podem ser obtidas dos chamados dispositivos de entrada como o teclado, arquivos e outros, e as saídas podem ser feitas via vídeo, impressora, arquivos, etc. Os comandos nos permitirão escolher o dispositivo de **entrada (padrão: teclado)** e **saída (padrão: vídeo)** a serem utilizados.


Temos em C funções para:

- Ler e escrever caracteres: *getchar ()* e *putchar ()*
- Ler e escrever strings: *gets ()* e *puts ()*
- Entrada/Saída formatada: *scanf ()* e *printf ()* entre outras funções.

Lendo e escrevendo caracteres


- **int getchar (void)**: a função **getchar** espera até que seja digitado um caracter no teclado, então mostra o caracter e sai do processo de entrada de dados.
- **int putchar (int c)**: a função **putchar** escreve um caracter a partir da posição corrente do cursor.

Vejamos o exemplo:

	<pre>/* Exemplo 4 - Programa que lê caracteres e escreve em caixa invertida. Lê maiúscula, escreve minúscula e vice-versa */ #include <stdio.h> #include <ctype.h> main () { char ch; puts ("\nDigite '.' e <ENTER> para terminar "); do { ch = getchar (); if (islower (ch)) ch = toupper (ch); else ch = tolower (ch); putchar (ch); } while (ch != '.'); }</pre>
---	---

Alternativas para `getchar`:

- **getch** (): lê um caracter do teclado sem ecoar o caracter na tela;
- **getche** (): mesma função do `getchar` () para ambiente interativo.


	<p>Algumas implementações <i>getchar</i> foram desenvolvidas para ambiente UNIX, o qual armazena os dados digitados num <i>buffer</i> até que seja digitado um <ENTER> quando então transfere os dados do <i>buffer</i> para o programa que espera a leitura. <i>getche</i> () não usa este buffer.</p> <p>As funções <i>getch</i> () e <i>getche</i> () se encontram definidas no arquivo de cabeçalho conio.h. Esta biblioteca (conio.h) não é do padrão da linguagem C, ela está disponível no Turbo C e em outros compiladores para Windows.</p>
---	---

Lendo e escrevendo string

- **char *gets (char *str)**: lê um string de caracteres digitado pelo teclado até que seja pressionado <ENTER>. O caracter da tecla <ENTER> não fará parte do string, no seu lugar é colocado o caracter de fim de cadeia ('\0').

- **int puts (char *s):** escreve um string na tela seguido por uma nova linha ('\n'). É mais rápido que a função printf, no entanto, só opera com argumentos string's.

Vejamos o exemplo:

	<pre>/* Exemplo 5 - Programa que lê um nome e escreve uma mensagem com o nome digitado. */ #include <stdio.h> main () { char nome[20]; puts ("\n Escreva seu nome:"); // \n pulo uma linha gets (nome); printf ("Oi, %s", nome); }</pre>
---	--


Entrada e saída formatada

- **Saída formatada:**

printf ("série de controle", listadeargumentos): função para impressão formatada na tela.

onde:

- "série de controle": é uma série de caracteres e comandos de formatação de dados que devem ser impressos.
- listadeargumentos: variáveis e constantes que serão trocados pelos formatos correspondentes na série de controle.

	<pre>// Exemplo 6 - Programa de saída formatada #include <stdio.h> main () { printf ("\n----+----+"); printf ("\n%-5.2f", 123.234); printf ("\n%5.2f", 3.234); printf ("\n%10s", "hello"); printf ("\n%-10s", "hello"); printf ("\n%5.7s", "1234567890"); }</pre>
---	--

Será apresentado na tela:

```

-----+-----+
123.23
 3.23
      hello
hello
1234567

```

Observações:


- A constante de barra invertida '\n' significa salto para uma nova linha (*new line*).
- O formato "%-5.2f" indica que o número em ponto flutuante (f) deve ser apresentado com no mínimo 5 caracteres, 2 dígitos para a parte fracionária do número e deve ser justificado a esquerda (-).
- O formato "%5.2f" indica a mesma coisa só que justificado a direita.
- O formato "%10s" indica que o string (s) deve ser apresentado em 10 espaços justificado a direita.
- O formato "%-10s" indica a mesma coisa só que justificado a esquerda.
- O formato "%5.7s" indica que o string (s) deve ser apresentado com pelo menos 5 caracteres e não mais que 7 caracteres.

Alguns comandos de formato do *printf*:

Código	Formato
%c	caracter
%d	inteiro com sinal
%i	inteiro com sinal
%e	notação científica
%E	notação científica
%g	%e ou %f (+curto)
%G	%E ou %f (+curto)
%%	caracter %

Código	Formato
%o	octal
%s	string
%u	inteiro sem sinal
%x	hexadecimal minúsculo
%X	hexadecimal maiúsculo
%p	endereço
%n	ponteiro de inteiro

Tabela 5 – Comandos de formato do *printf*




```
// Exemplo 7 - Tabela de quadrado e cubo de números
#include <stdio.h>
main ()
{
    int i;
    printf ("\n\n %8s %8s %8s", "x", "x^2", "x^3");
    for (i = 1; i <= 10; i++)
        printf ("\n %8d %8d %8d", i, i * i, i * i * i);
    printf ("\n");
}
```

Será apresentado na tela:


x	x^2	x^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
... (até 10)		

- **Entrada formatada:**

scanf ("série de controle", listadeendereçodosargumentos): função para leitura formatada.



```
/* Exemplo 8 - Uso da função scanf */
#include <stdio.h>
main ()
{
    char nome[20];
    int idade;
    printf ("\nDigite um nome: ");
    scanf ("%s", nome);
    printf ("\nDigite a idade: ");
    scanf ("%d", &idade);
    printf ("%s, você tem %d anos\n", nome, idade);
}
```



- Este programa pede um nome e uma idade e monta uma frase do tipo: "João, voce tem 26 anos".
- Detalhes como a declaração char nome [20]; e da chamada da função scanf ("%d", &idade); serão esclarecidos quando discutirmos os assuntos de *strings* em C e passagem de parâmetros por referência em funções.

2.2. Constantes

Constantes são valores fixos que não podem ser alterados por um programa. Uma constante pode ser de quatro tipos:

- caracter (char): 'a', '%' , envolvidos por aspas simples (apóstrofe);
- inteiro (int): 10, -97, números sem componente fracionário;
- real (float, double): 11.12, número com componente fracionário;
- strings: "universidade", caracteres envolvidos por aspas duplas.

Além dessas a linguagem C possui:

- Constantes de barra invertida.
- Constantes hexadecimais e octais.

Constantes de barra invertida


Para descrever alguns caracteres da tabela ASCII que não podem ser definidos usando as aspas simples existem em C, alguns códigos de barra invertida que representam estes caracteres. Exemplos:

Código	Significado
\b	retrocesso (BS)
\f	alimentação de formulário (FF)
\n	nova linha
\r	retorno de carro
\t	tabulação horizontal
\"	caracter "
\'	caracter '
\0	nulo (fim de cadeia de caracteres)
\\	barra invertida
\v	tabulação vertical
\a	beep de alerta
\N	constante octal (N - número octal)
\xN	constante hexadecimal


Tabela 6 – Comandos de barra invertida

Constantes hexadecimais e octais


- **Constante Hexadecimal:** Número na base numérica 16. Os dígitos nesta base numérica são 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, ou seja, temos 16 dígitos.



	<p>1A = 26 em decimal, FF = 255 em decimal.</p>
---	---

- **Constante Octal:** Número na base numérica 8. Os dígitos nesta base numérica são 0, 1, 2, 3, 4, 5, 6, 7 ou seja, temos 8 dígitos.

	<p>11 = 9 em decimal, 24 = 20 em decimal.</p>
---	---

Para que possamos especificar ao compilador C que uma constante é Hexadecimal ou Octal usamos de dois símbolos: 0x e 0 antes do número Hexadecimal e Octal respectivamente.

	<p>0xconstante: especifica que um número é Hexadecimal 0constante: especifica que um número é Octal.</p>
---	--


	<pre>/* Exemplo 9 - Constantes Hexadecimais e Octais */ #include <stdio.h> main () { int h = 0xff; int o = 024; printf ("HEX : %X = DECIMAL : %d\n", h, h); printf ("OCTAL: %o = DECIMAL : %d\n", o, o); }</pre>
	<p>Os padrões %X e %o serão trocados pelos valores das variáveis h e o nos formatos hexadecimal e octal respectivamente. Na tela aparecerá:</p> <pre>HEX : FF = DECIMAL : 255 OCTAL: 24 = DECIMAL : 20</pre>

2.3. Operadores

Existem 5 (cinco) níveis de operadores em C: atribuição, aritméticos, relacionais, lógicos e bit a bit.

Operador de Atribuição


A linguagem C usa o sinal de igual (=) como operador de atribuição. O formato geral de uma atribuição em C é o seguinte:

	<code>nome_da_variável = expressão;</code>
---	--

Na Linguagem C é feita conversão automática de tipos na atribuição, ou seja, o valor do lado direito da atribuição é convertido no tipo do lado esquerdo. Se o tamanho em bytes do tipo do lado esquerdo for menor do que o valor atribuído alguma informação será perdida (os bits mais significativos do valor serão desprezados).

Pode-se, também, atribuir um mesmo valor para diversas variáveis em C. Assim:

X = Y = Z = 0;

	<pre>int X = 10; char Ch = 'a'; float F = 12.5; void main (void) { Ch = X; //--> Os bit + significativos de X são ignorados X = F; //--> X recebe a parte inteira de F F = Ch; //--> O valor inteiro de Ch é convertido para //ponto flutuante F = X; //--> Converte o valor inteiro de X para seu //formato em ponto flutuante }</pre>
---	---

Operadores Aritméticos

Os operadores aritméticos de C são os seguintes:

Operador	Ação
-	Subtração
+	Adição
*	Multiplificação
/	Divisão
%	Módulo (resto) da divisão inteira
++	Incremento
--	Decremento

Tabela 7 – Comandos de barra invertida

O uso destes operadores é igual ao uso nas outras linguagens, exceto os de incremento e decremento:


- Incremento (++) e decremento (--)
 $x = x + 1$ e $++x$ são instruções equivalentes em C
 $x = x - 1$ e $--x$ também o são.

No entanto os operadores de incremento e decremento podem preceder ou suceder o operando assim:

++X e X++

Existe uma diferença básica entre estes dois usos:

- $++x$: executa a operação de incremento/decremento antes de usar o valor do operando.
- $x--$: executa a operação de incremento/decremento depois de usar o valor do operando.



```
/* Exemplo 10 - Operador de Incremento */  
  
#include <stdio.h>  
  
main ()  
{  
    int X, Y;  
    X = 10;  
    Y = ++X;  
    printf ("X= %d Y= %d\n", X, Y);  
    X = 10;  
    Y = X++;  
    printf ("X= %d Y= %d\n", X, Y);  
}
```

Será apresentado na tela:

X = 11	Y = 11
X = 11	Y = 10

maior



menor

Operador	Significado
++, --	incremento, decremento
*, /, %	multiplicação, divisão e módulo
+, -	adição e subtração

Tabela 8 – Precedência dos Operadores Aritméticos



- Operadores de mesma precedência são tratados primeiro da esquerda para direita.
- Para alterar a ordem de precedência devem-se usar parênteses.

Operadores Relacionais e Lógicos

Operadores relacionais tratam das relações entre valores (=, >, < etc...) e os operadores lógicos fazem a composição de expressões relacionais.



- VERDADEIRO EM C = qualquer valor diferente de zero.
- FALSO EM C = valor zero

a	b	a && b	a b	!a
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Tabela 9 – Tabela Verdade dos Operadores Lógicos

onde:

- **&&**: é o and (e lógico) – a && b le-se "a e b".
- **||**: é o or (ou lógico) – a || b le-se "a ou b".
- **!**: é o not (negação) – ! a le-se "negação de a".

E

```

/* Exemplo 11 - Operadores Lógicos */

#include <stdio.h>

main ()
{
    int a = 1, b = 0, c;
    c = a && !b || !a && b;
    if (c)
        puts ("1");
    else
        puts ("0");
}

```

maior
↓
menor

Operador	Significado
!	negação
<, <=, >, >=, ==, !=	menor, menor igual, maior, maior igual, igual, diferente
&&	e lógico
	ou lógico

Tabela 10 – Precedência dos Operadores Relacionais e Lógicos

Todo resultado de uma expressão lógica e/ou relacional é o número 0 ou diferente de 0 (zero). Então será válido em C o seguinte programa:

E

```

/* Exemplo 12 - Resultado de uma operacao relacional */

#include <stdio.h>
main ()
{
    int X = 1;
    printf ("%d", X > 10);
}

```



Operadores Bit a Bit

São operadores que possibilitam a manipulação dos bits numa palavra ou byte da linguagem C. As operações bit a bit são AND, OR, XOR, complemento, deslocamento a esquerda, deslocamento a direita. Essas operações atuam nos bits dos operandos. São eles:

Operador	Ação
&	and binário
	or binário
^	xor binário
~	complemento
>>	deslocamento à direita
<<	deslocamento à esquerda

Tabela 11 – Operadores Bit a Bit

Os operadores bit a bit são muito usados em programas "drivers" de dispositivos (modem, scanner, impressora, porta serial) para mascarar bits de informação nos bytes enviados por estes dispositivos.


	<pre>/* Exemplo 13 - Tradução de hexadecimal para binário */ #include <stdio.h> main () { int h = 0x7A, i, m = 0x80; for (i = 1; i <= 8; i++) { if (m & h) putchar ('1'); else putchar ('0'); m = m >> 1; } }</pre>
	<ul style="list-style-type: none"> ▪ O número hexadecimal 7A foi vasculhado por este programa para ser traduzido para o seu equivalente binário. Operações bit a bit foram necessárias para encontrarmos os bits acesos (1) e apagados (0) do byte h. ▪ Será apresentado na tela: 01111010.

2.4. Expressões


Uma expressão é a composição de operadores lógicos, aritméticos e relacionais, variáveis e constantes. Em C uma expressão é qualquer combinação válida destes elementos.

Observações:


- **Conversão de tipos em expressões:** numa expressão, o compilador C converte todos os operandos no tipo do maior operando (promoção de tipo).


	<pre>char ch; result = (ch / i) + (f * d) - (f + i) int i; \ / \ / \ / float f; int double float double d; \ / / double / \ / Double</pre>
---	--

- Pode-se forçar que o resultado de uma expressão seja de um tipo específico através do uso de um molde (CASTS) de tipo. Assim:

	<pre>(float) x / 2; //o resultado de X dividido por 2 é um //um valor real</pre>
---	--

- Espaços e parênteses podem ser usados a vontade em expressões "C" sem que isto implique em diminuição de velocidade de execução da expressão. O uso de parênteses e espaço é aconselhado no sentido de aumentar a legibilidade do programa. O parênteses pode mudar a ordem de precedência dos operadores.
- Abreviações em C: A linguagem C admite que algumas atribuições sejam abreviadas.

	<pre>X = X + 10; é o mesmo que X += 10</pre>
--	--

	<p>Sintaxe:</p> <pre>variável = variável operador expressão //é o mesmo que variável operador = expressão</pre>
---	--

2.5. Exercícios

1. Descreva o que faz cada uma das funções abaixo.

```
getch() ->
getchar() ->
putchar() ->
getche() ->
gets() ->
puts() ->
```

2. Implemente um programa que leia dois valores Inteiros e apresente o resultado de sua soma, subtração, multiplicação e divisão.
3. Escreva um programa em C para apresentar um valor constante qualquer do tipo Inteiro, Caractere, Float e String. Exemplo:

4.

```
Inteiro = 10
Caractere = A
Float = 99.99
String = Linguagem C
```

5. Desenvolva um programa que escreva os valores 9, 11, 13, 19 e 25 em hexadecimal e em octal. Cada valor hexadecimal deverá ser escrito entre aspas duplas e o valor octal entre aspas simples. Os valores deverão ser escritos em linhas diferentes. Exemplo:

```
Valor 8 -> "8" '10'
Valor 11 -> "B" '13'
```

6. Escreva o conteúdo das variáveis x, y e z depois da seguinte sequência de operações:

```
int x, y, z;
x=y=10;
z=++x;
x=-x;
y++;
x=x+y-(z--);
```

7. Implemente e verifique o resultado do programa abaixo.

```
#include <stdio.h>
int main()
{
    int i, j;
    printf("\nEntre com dois números inteiros: ");
    scanf("%d%d", &i, &j);
    printf("\n%d == %d é %d\n", i, j, i==j);
    printf("\n%d != %d é %d\n", i, j, i!=j);
    printf("\n%d <= %d é %d\n", i, j, i<=j);
    printf("\n%d >= %d é %d\n", i, j, i>=j);
    printf("\n%d < %d é %d\n", i, j, i<j);
    printf("\n%d > %d é %d\n", i, j, i>j);
}
```

8. Diga se as seguintes expressões são verdadeiras ou falsas. Implemente um programa que escreva o resultado dessas expressões.

```
((10>5) || (5>10))
(! (5==6) && (5!=6) && ((2>1) || (5<=4)))
```

9. Implemente o programa e verifique o resultado. Se o resultado estiver incorreto, altere para que o programa escreva o valor corretamente.

```
#include <stdio.h>
int main ()
{
    int num = 10;
    float f;
    f=num/4;
    printf ("%f",f);
}
```

3. Estruturas Básicas de Controle


As estruturas básicas de controle foram aplicadas por você na disciplina de Linguagem e Técnicas de Programação I com a Linguagem de Programação Pascal. Nesta unidade você poderá visualizar as estruturas básicas de controle utilizando a Linguagem de Programação C, incluindo comando de seleção, comando de interação, comando de desvio, matrizes e *strings*.

O padrão ANSI divide os comandos em C nos seguintes grupos:

- **seleção:** if, switch.
- **iteração:** while, do-while e for.
- **desvio:** break, continue e return.
- **expressão:** expressões válidas em C.
- **bloco:** blocos de código entre { e }.

3.1. Comando de Seleção

- **Comando de Seleção Simples (if):** O comando if representa uma tomada de decisão do tipo "SE isto ENTÃO aquilo, SENÃO ...". A sua forma geral é:

	Sintaxe: <pre>if (expressão) comando1; else comando2;</pre>
---	---

A expressão que controla a seleção deve estar entre parênteses. O comando1, que pode ser um comando simples ou um comando em bloco, será executado se o resultado da expressão for diferente de zero (VERDADEIRO em C). Por outro lado, o comando2 será executado se o resultado da expressão for igual a zero (FALSO em C).

Diferentemente de PASCAL, em C temos um ponto e vírgula antes da palavra reservada **else**.





```
/* Exemplo 01 - Comando de selecao simples - if */
#include <stdio.h>
int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num==10)
    {
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.");
    }
    else printf ("O numero não é igual a 10");
}
```



```
/* Exemplo 02 - Comando de selecao simples - if */
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <conio.h>
void main (void)
{
    int Numero, Palpite;
    randomize ();
    Numero = rand ();
    puts ("\nAdivinhe o numero magico");
    scanf ("%d", &Palpite);
    if (Palpite == Numero)
        puts ("\n Voce ACERTOU...!");
    else
        puts ("\n Voce errou...!");
    getch ();
}
```


Na linguagem C, assim como em outras linguagens, podemos ter vários comandos **if** aninhados. As regras para aninhamento de if's é igual para outras linguagens de programação. O comando **else** em C se refere-se ao **if** mais próximo (acima deste) e que esteja no mesmo bloco.

**Sintaxe:**

```
if (i)
{
    if (j)
        comando1;
    if (k)
        comando2;
    else
        comando3;
}
else
    comando4;
```




- **Comando de Seleção Múltipla (*switch*):** O comando **switch** é próprio para se testar uma variável em relação a diversos valores pré-estabelecidos. Sua forma geral é:

	<p>Sintaxe:</p> <pre>switch (expressão) { case constante1: sequência de comandos1; break; case constante2: sequência de comandos2; break; . . . case constanteN: sequência de comandosN; break; default: sequência de comandos; }</pre>
---	---

Se o resultado da expressão for igual a constante1, será executada a sequência de comandos1. Se o resultado da expressão for igual a constante2, será executada a sequência de comandos2. A sequência de comandos após a palavra chave default será executada se o resultado da expressão não for igual a nenhuma das constantes.

O comando **break**, após cada sequência de comandos, põe fim ao comando switch. Se break não for usado, todas as instruções contidas abaixo da sequência de comandos serão executadas, verifique o exemplo abaixo:


	<pre>/* Exemplo 03 - Comando de selecao multipla */ #include <stdio.h> main () { char c; printf ("\nDigite um numero: "); c = getchar (); printf ("\n"); switch (c) { case '1': printf ("um\n"); case '2': printf ("dois\n"); case '3': printf ("tres\n"); } }</pre>
---	--



Observações:

- Se i = '1' então será apresentado na tela um dois tres.
- Se i = '2' então será apresentado na tela dois tres.
- Se i = '3' então será apresentado na tela tres.
- Como no caso do comando IF o comando SWITCH também pode ser aninhada em vários níveis.


O correto é:

	<pre>/* Exemplo 04 - Comando de selecao multipla */ #include <stdio.h> main () { char c; printf ("\nDigite um numero: "); c = getchar (); printf ("\n"); switch (c) { case '1': printf ("um\n"); break; case '2': printf ("dois\n"); break; case '3': printf ("tres\n"); break; } }</pre>
--	---

3.2. Comando de Iteração

Comandos de iteração (malha ou laço de repetição) são usados para permitir que um conjunto de instruções sejam executadas enquanto uma determinada condição seja verdadeira. Os comandos de iteração em C são: **for**, **while**, **do-while**.


- **Comando de Repetição FOR:**

	<p>Sintaxe:</p> <pre>for (inicialização; condição; incremento) comando;</pre>
---	---




onde:

- ✚ **Inicialização:** É geralmente um comando de atribuição que coloca um valor inicial para as variáveis de controle do laço for.
- ✚ **Condição:** É uma condição relacional, lógica ou aritmética que define até quando a iteração pode continuar (condição de permanência da repetição)
- ✚ **Incremento:** Define como a variável de controle do laço varia cada vez que o laço é repetido.


	<pre>/* Exemplo 05 - Contagem com o comando de repeticao for */ #include <stdio.h> main () { int X; for (X = 1; X < 100; X++) printf ("%4d", X); printf ("\n"); }</pre>
---	--

Explicação: assim que o programa entra na repetição for a variável inteira X assume o valor 1. Dentro da repetição a única instrução é escrever o valor de X, que irá variar de um em um (X++) enquanto X é menor que 100.

	<pre>/* Exemplo 06 - For com comando em bloco */ #include <stdio.h> main () { int X, Y; for (X = 100; X != 65; X -= 5) { Y = X * X; printf ("O quadrado de %3d e' %5d\n", X, Y); } }</pre>
---	--

O laço de repetição for pode trabalhar com mais do que uma única variável. Pode-se inicializar e incrementar mais do que uma variável no escopo da repetição for, veja o exemplo a seguir:




	<pre>/* Exemplo 07 - Repeticao for com 2 variaveis de controle */ #include <stdio.h> #include <string.h> main () { char s[8] = "Joaquim"; int i, j; for (i = 0, j = strlen (s) - 1; i < strlen (s); i++, j--) { printf ("%c - %c\n", s[i], s[j]); } }</pre>
---	--

- **Comando de Repetição WHILE:**

	<p>Sintaxe:</p> <pre>while (condição) { comando; }</pre>
---	---


onde:

- ✎ **condição:** é qualquer expressão onde o resultado é um valor igual ou diferente de zero.
- ✎ **comando:** é um comando simples ou um comando em bloco (comando composto).

	<pre>/* Exemplo 08 - Repeticao while */ #include <stdio.h> main () { char ch = 0; while (ch != 's') ch = getchar (); puts ("\nFim"); }</pre>
---	--

O programa ficará executando a malha while até que o usuário digite o caracter 's' minúsculo. Antes a variável ch é inicializada com nulo (zero).


- **Comando de Repetição DO - WHILE:** ao contrário dos laços while e for que testam a condição de permanência no começo, o laço do-while testa a condição de permanência no final.

	<p>Sintaxe:</p> <pre>do { comando; } while (condição);</pre>
---	---



onde:


- ↗ **condição:** é qualquer expressão onde o resultado é um valor igual ou diferente de zero.
- ↗ **comando:** é um comando simples ou um comando em bloco (comando composto).

	<pre>/* Exemplo 09 - Repeticao do-while */ #include <stdio.h> main () { int i = 1; do { printf ("%4d", i); i++; } while (i <= 100); }</pre>
---	--

3.3. Comandos de Desvio


- **Comando Return:** obriga o programa a retornar da execução de uma função. Se o programa está executando uma função e encontra o comando **return**, o programa retornará ao ponto em que foi chamada a função.

	Sintaxe: return expressão;
---	---



	<pre>/* Exemplo 10 - Comando de desvio return */ #include <stdio.h> Int menos (int a, int b) { return (a - b); } main () { int x, y; puts ("\nDigite dois inteiros separados por espaco: "); scanf ("%d%d", &x, &y); printf ("%d menos %d e' igual a %d\n\n", x, y, menos (x, y)); }</pre>
---	--



- **Comando Break:** permite terminar um case da instrução switch, ou terminar uma malha de repetição evitando o teste de condição de permanência.

	<pre>/* Exemplo 11 - Comando de desvio break */ #include <stdio.h> main () { int t = 1; for (;;) { if (t > 100) break; printf ("%4d", t++); } }</pre>
---	--

- **Comando Continue:** tem efeito contrário do comando break. Obriga a próxima iteração do laço, pulando qualquer código intermediário. Para o laço for, o comando continue faz com que a condição e o incremento sejam executados.

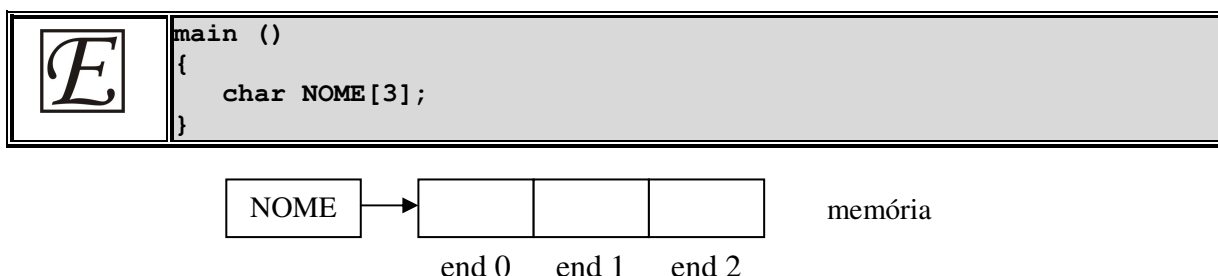
	<pre>/* Exemplo 12 - Comando de desvio continue */ #include <stdio.h> #include <string.h> main () { char s[80]; int i; printf ("\nDigite um nome completo: "); fgets (s, 79, stdin); for (i = 0; i != strlen (s); i++) { if (s[i] == ' ') continue; putchar (s[i]); } }</pre>
	Implemente o programa no Turbo C e verifique o seu funcionamento.

3.4. Matrizes e Strings

Matriz é uma coleção homogênea de dados que ocupam posições contíguas na memória e cujos elementos podem ser acessados através da especificação de



índice. O endereço mais baixo na memória corresponde ao primeiro elemento na matriz, o mais alto corresponde ao último elemento.



O nome da matriz é um ponteiro para a primeira posição na memória onde está alocada a matriz. No exemplo, NOME é um apontador do primeiro elemento da matriz na memória. Todas as matrizes em C começam no índice 0 (zero) e podem ser unidimensional ou multidimensional.

• Matriz Unidimensional (Vetor)

!

Sintaxe:

```
tipo nome_de_variavel [tamanho];
```

E

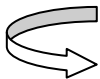
```
float VALOR [100];
```

Todas as matrizes em C começam no índice 0 (zero), então quando declaramos a matriz anterior (float VALOR [100]) estamos definindo na memória os dados VALOR [0], VALOR [1], ..., VALOR [99].

E

```
/* Exemplo 13 - Matriz Unidimensional */
#include <stdio.h>
#include <conio.h>
void main (void)
{
    int Indice, Vetor[100];
    for (Indice = 100; Indice > 0; Indice--)
        Vetor [Indice-1] = Indice;
    for (Indice = 1; Indice <= 100; Indice++)
        printf ("%4d", Vetor [Indice-1]);
}
```




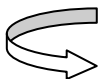
	<p>Implemente o programa no Turbo C e verifique o seu funcionamento.</p>
---	--

IMPORTANTE: A linguagem C não verifica limites na matriz, então para o exemplo anterior Vetor [2200] = 20 seria compilado sem erro, mas este programa teria um erro de lógica porque outra posição de memória será afetada de forma indesejável.


• Matriz Bidimensional


	<p>Sintaxe: tipo nome [t1][t2];</p>
---	--

	<pre>/* Exemplo 14 - Matriz Bidimensional */ #include <stdio.h> #include <conio.h> void main (void) { int IndiceL, IndiceC, Matriz[3][4]; for (IndiceL = 0; IndiceL < 3; IndiceL++) for (IndiceC = 0; IndiceC < 4; IndiceC++) Matriz [IndiceL][IndiceC] = IndiceL * IndiceC; for (IndiceL = 0; IndiceL < 3; IndiceL++) for (IndiceC = 0; IndiceC < 4; IndiceC++) printf ("%4d", Matriz [IndiceL][IndiceC]); getch (); }</pre>
--	---

	<p>Implemente o programa no Turbo C e verifique o seu funcionamento.</p>
---	--


• Inicialização de Matrizes

	<p>Sintaxe: tipo nome [t1][t2] = { lista de valores };</p>
---	---

	<pre>int Valores[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; char Teste[5] = {'a', 'b', 'c', 'd', 'e'};</pre>
---	--

- **Strings (Matriz de Caracteres)**


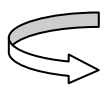
Uma string em C é uma matriz de caracteres terminada com um fim de cadeia '\0' (o caracter nulo). Por isto devemos **declarar uma string** com um caracter a mais do que necessitamos para a nossa aplicação. Desta forma se necessito de uma string de 10 posições, devo declará-lo com 11 posições.



	<code>char s [11]; /* índices de (0 - 10) */</code>
---	---

Para manipulação do tipo **string** existe na biblioteca C uma série de funções, conforme apresenta a tabela seguinte:

Nome	Função
<code>strcpy (s1, s2)</code>	copia o string s2 em s1
<code>strcat (s1, s2)</code>	concatena s2 ao final de s1
<code>strlen (s1)</code>	retorna o tamanho do string s1
<code>strcmp (s1, s2)</code>	compara o string s1 com s2
<code>strchr (s1, ch)</code>	retorna o endereço de memória da primeira ocorrência do caracter ch no string s1
<code>strstr (s1, s2)</code>	retorna o endereço de memória da primeira ocorrência do string s1 no string s2.

Tabela 12 – Funções de manipulação de Strings

	<p>Os Strings não podem ser manipulados diretamente, apenas com o uso das funções acima. Portanto, o exemplo abaixo está incorreto:</p> <pre>char s [11]; /* índices de (0 - 10) */ s = "Teste";</pre> <p>O correto é:</p> <pre>char s [11]; /* índices de (0 - 10) */ strcpy(s, "Teste");</pre>
	Implemente no Turbo C e verifique o seu funcionamento.

	<pre> /* Exemplo 15 - Funcoes para strings em C */ #include <stdio.h> #include <string.h> main () { char nome1[40], nome2[40]; printf ("\nDigite um nome : "); fgets (nome1, 39, stdin); printf ("Digite outro nome : "); fgets (nome2, 39, stdin); puts ("\n**** TESTE DA FUNCAO strlen "); printf ("Tamanho do primeiro nome : %d", strlen (nome1)); printf ("\nTamanho do segundo nome : %d", strlen (nome2)); puts ("\n\n**** TESTE DA FUNCAO strcmp "); if (!strcmp (nome1, nome2)) puts ("Os nomes sao iguais"); else puts ("Os nomes sao diferentes"); puts ("\n\n**** TESTE DA FUNCAO strcpy "); strcpy (nome1, "Universidade "); printf ("Novo nome: %s ", nome1); puts ("\n\n**** TESTE DA FUNCAO strcat "); strcat (nome1, nome2); printf ("Nomes concatenados : %s ", nome1); puts ("\n\n**** TESTE DA FUNCAO strchr "); if (strchr (nome2, 'a')) puts ("Existe 'a' no segundo nome"); else puts ("Nao existe 'a' no segundo nome"); puts ("\n**** TESTE DA FUNCAO strstr "); if (strstr (nome2, "ana")) puts ("Existe \"ana\" no segundo nome"); else puts ("Nao existe \"ana\" no segundo nome"); } </pre>
	<p>A função strcmp retorna:</p> <ul style="list-style-type: none"> • 0 (zero) se s1 e s2 são iguais • < 0 se s1 é alfabeticamente menor que s2 • > se s1 é alfabeticamente maior que s2

3.5. Exercícios

1. Escreva um programa que coloque os números de 1 a 1000 na tela na ordem inversa (começando em 1000 e terminando em 1).
2. Declare um vetor em C de 20 posições de inteiros e de nome NUMEROS. Quais serão os índices válidos para este vetor.



3. Fazer um programa em C que leia uma matriz de inteiros 3×4 e some os valores de cada coluna da matriz num vetor de 4 posições de inteiros.
4. Elaborar um programa em C para fazer a multiplicação de duas matrizes 4×4 .
5. Elaborar um programa em C para somar duas matrizes 3×3 e apresentar o resultado da soma de maneira tabular.
6. Desenvolver um programa em C para ler uma lista de números e calcular qual o maior dos números.
7. Usando matriz unidimensional, faça um programa em C que reverta uma cadeia de caracteres. **Assim: Antes: "TESTE" Depois: "ETSET"**



4. Estruturas Avançadas de Controle

Neste capítulo, você vai aprender as Estruturas Avançadas de Controle da linguagem C, entre eles: ponteiros, funções, estruturas (registros) e alocação dinâmica de memória. Esses comandos são essenciais para construção de aplicações avançadas e irá auxiliar você no desenvolvimento da disciplina de Estrutura de Dados.

4.1. Ponteiros

Ponteiros são variáveis que apontam para algum endereço de memória. São usadas para alocação dinâmica de espaço em memória, ou seja, alocação de espaço de memória que ocorre em tempo de execução dos programas.



Sintaxe:
`tipo *nome_da_variavel;`



```
char *p; /* o conteúdo do endereço apontado por p é do tipo char */
```

Operadores de ponteiros



`&` - Devolve o endereço de uma variável.



```
m = &x; /* m recebe o endereço da variável x */
```



`*` - Devolve o conteúdo de memória apontado por um ponteiro.





```
c = *p; /* c recebe o conteúdo de memória apontado por p */
```

Expressões usando ponteiro

- **Atribuição:** permite atribuir um endereço para um ponteiro.



```
void main (void)
{
    int x;                /* uma variável inteira */
    int *p1, *p2;         /* dois ponteiros de inteiros */
    p1 = &x;              /* p1 recebe o endereço de x */
    p2 = p1;              /* p2 recebe o endereço apontado por p1 */
    printf ("%p", p2);    /* escreve o endereço de x */
}
```

- **Aritmética:** são permitidas duas operações: adição (incremento) e subtração (decremento).



```
char *ch = 3000;
ch = ch + 1;
int *i = 3000;
i = i + 1;
```

Quando incrementamos um ponteiro estamos na verdade saltando o tamanho do tipo base na memória do computador. No exemplo anterior o incremento de 1 na variável **ch** provoca um salto para o próximo byte na memória (**ch aponta para char que ocupa um byte na memória**).

No caso de um incremento de 1 na variável **i**, provoca um salto de dois bytes a frente na memória (o tipo base **int** ocupa dois bytes). Para o decremento (subtração) o mecanismo é semelhante.

- **Comparação:** todos os operadores relacionais são válidos (==, <, !=, etc...)



```
int *p, *q;
if (p < q)
    printf ("o endereço apontado por p1 e < que o de q");
```



Ponteiros e Matrizes

Na declaração:

```
char str [80], *p1;
```


Tanto a variável str com p1 apontam para um caracter (str aponta para o primeiro caracter do string e p1 para um caracter qualquer).

A atribuição:

```
p1 = str
```

Faz com que p1 e str apontem para a primeira posição do string. Então para acessar o quinto elemento do string poderemos proceder de duas formas:

```
str [4] ou *(p + 4) ou p[4]
```

	<pre>/* Exemplo 16 - Aplicacao com ponteiros */ #include <stdio.h> #include <ctype.h> void main (void) { char *nome = "Universidade"; while (*nome) putchar (toupper (*nome++)); printf ("\n"); }</pre>
---	--

4.2. Funções

	<p>Sintaxe: <code>tipo NomedaFuncao(ListaParametros);</code></p>
---	---

onde:

- **tipo:** deve ser um tipo válido de C. Se não for definido nenhum tipo para função, C assume que a função devolve um valor inteiro (int).
- **NomedaFuncao:** nome válido de identificador em C pelo qual a função será chamada.



- **ListaParametros:** lista de nomes de variáveis separadas por vírgulas e seus tipos associados. Estas variáveis receberão os valores dos argumentos na chamada da função.



```
/* Exemplo 17 - Funcao esta contido */  
  
#include <stdio.h>  
#include <string.h>  
  
contido (char c, char *s); //a função retorna int  
  
void main (void)  
{  
    char nome[30], car;  
    printf ("\nDigite um nome....: ");  
    fgets (nome, 29, stdin);  
    nome[strlen (nome) - 1] = '\0';  
    printf ("\nDigite um caracter: ");  
    scanf ("%c", &car);  
    if (contido (car, nome))  
        printf ("O caracter %c esta no nome [%s]\n", car, nome);  
    else  
        printf ("O caracter %c nao esta no nome [%s]\n", car, nome);  
}  
  
contido (char c, char *s)  
{  
    while (*s)  
        if (*s == c)  
            return 1; //return é utilizado para retornar o valor  
        else  
            s++;  
    return 0;  
}
```

Chamado por Valor e chamada por Referência

Como na Linguagem Pascal, em C existem duas maneiras de se passar argumentos para uma função:

- **Passagem por valor:** uso normal dos parâmetros formais da função sem alterar os valores dos argumentos:



E

```
/* Exemplo 18 - Parametros por valor */
#include <stdio.h>
int quadrado (int x);
main ()
{
    int n = 8;
    printf ("\nO quadrado de %d e' %d\n", n, quadrado (n));
}

int quadrado (int x)
{
    return x * x;
}
```

Observação:

- ✓ Para passagem por referência devemos passar o endereço do argumento que queremos alterar dentro da função.
- ✓ Existem funções de biblioteca C que usam deste mecanismo. Como já vimos anteriormente, o Scanf é uma delas:

E

```
int t;
scanf ("%d", &t);
```

- **Passagem por referência:** o argumento é alterado pela função.

E

```
/* Exemplo 19 - Parametros por referencia */
#include <stdio.h>

void quadrado (int x, int *q);

main ()
{
    int n = 8, r;
    quadrado (n, &r);
    printf ("\nO quadrado de %d e' %d\n", n, r);
}

void quadrado (int x, int *q)
{
    *q = x * x;
}
```




Matrizes passadas para função

Pode-se passar uma matriz inteira como argumento de uma função. Na verdade quando fazemos isto estamos passando o endereço da primeira posição da matriz para a função. Assim:

	<pre>main () { int i [10]; func (i); . . . }</pre>
---	--

O parâmetro matriz da função **func** pode ser declarado de uma das 3 formas:

func (int *X) ou func (int X [10]) ou func (int X[])	
	<pre>/* Exemplo 20 - Vetores para funcao */ #include <stdio.h> void mostra (int *vetor, int tam); int total (int *vetor, int tam); main () { int i, vetA[10], stime; long ltime; ltime = time (NULL); stime = (unsigned) ltime / 2; srand (stime); for (i = 0; i < 10; i++) vetA[i] = rand () % 10; mostra (vetA, 10); printf ("%d\n", total (vetA, 10)); } void mostra (int *vetor, int tam) { int i; for (i = 0; i < tam; i++) { printf ("%d", vetor[i]); if (i < tam - 1) putchar ('+'); else putchar ('='); } } int total (int *vetor, int tam) { int i, t; for (i = 0, t = 0; i < tam; i++) t = t + vetor[i]; return t; }</pre>



Funções de Biblioteca

Como temos utilizado no dia-dia, várias funções de biblioteca C podem ser incluídas em programas. Isto é feito com a inclusão de um arquivo de cabeçalho no início do programa. Este arquivo de cabeçalho (com extensão .h) contém uma série de definições dos protótipos de funções e de constantes. Por exemplo, com a inclusão do arquivo de cabeçalho STDIO.H:

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

Com isso, as funções de biblioteca printf, scanf entre outras estarão disponíveis ao programa. Verifique abaixo alguns arquivos de cabeçalho e suas principais funções:

Arquivo de cabeçalho	Principais funções
conio.h	clrscr(); gotoxy(); clreol(); kbhit(); getch()
stdio.h	gets(); puts (); printf(); scanf(); getchar(); putchar()
stdlib.h	abort(); atof(); atol(); atoi(); exit()
string.h	strcmp(); strcat(); strcpy(); strchr(); strlen(); strstr(); strlen()
graphics.h	initgraph(); rectangle(); closegraph(); getimage(); putimage(); putpixel(); floodfill(); line()
dos.h	delay(); gettime(); setdate(); getdate(); peek(); poke(); sound()
ctype.h	toupper(); tolower()


Observação: no Turbo C, utilize o comando alt+f1 sobre o nome da função para verificar a explicação e a função do comando.


4.3. Estruturas

Estruturas são conjuntos heterogêneos de dados que são agrupados e referenciados através de um mesmo nome. Equivalente ao REGISTRO de outras linguagens de programação (por exemplo, da linguagem Pascal).



A forma geral de definição de estrutura é a seguinte:

	<pre>struct nome { tipo nome_da_variavel; tipo nome_da_variavel; tipo nome_da_variavel; ... } variaveis_estruturas;</pre>
---	---


	<pre>struct registro { char nome [30]; char endereco [30]; char cidade [20]; char telefone [15]; char estado [3]; char cep [10]; } func, auxiliar1, auxiliar2;</pre>
---	--

- Com esta declaração está sendo definida uma estrutura de nome registro e três variáveis que são do tipo desta estrutura construída: func, auxiliar1, auxiliar2.

Referência a Elementos da Estrutura

Para referenciar um elemento da estrutura usa-se o operador ponto. Por exemplo, para se atribuir o CEP 37460-000 para o campo CEP da variável estrutura **func** usa-se o seguinte código:


```
strcpy (func.cep, "37460-000");
```

	<ul style="list-style-type: none">• a atribuição de variáveis do tipo string em C só pode ser feita através da função de biblioteca strcpy.• o nome da variável estrutura seguindo por um ponto e pelo nome do elemento referencia o elemento individual da estrutura. Neste caso, func.cep referencia o campo cep da estrutura func.• o operador ponto pode aparecer em qualquer lugar. Exemplo: gets (func.nome);
---	--




Atribuição de Estruturas


Uma estrutura inteira pode ser atribuída a outra, nas versões de C que são compatíveis ao padrão ANSI.

	<pre>/* Exemplo 21 - Estruturas em C */ #include <stdio.h> void main (void) { struct { int a; int b; } x, y; x.a = 7; y = x; printf ("\n%d", y.a); }</pre>
---	--

Matrizes de Estruturas


Pode-se declarar matrizes de estruturas em C. Por exemplo, para a estrutura registro declarada anteriormente, a seguinte matriz é válida:

	<pre>struct registro matriz [100];</pre>
---	--

	<ul style="list-style-type: none">• Com esta declaração está sendo definida uma matriz de índices de 0 a 99 de elementos do tipo estrutura registro.• Para se acessar um estrutura específica, deve-se indexar o nome da estrutura. Por exemplo para se imprimir o código de CEP da estrutura 3, escreva: printf ("%s", matriz [2].cep);
---	--

Passando Estruturas inteiras para funções

Quando uma estrutura é passada para uma função, a estrutura inteira é passada usando o método de chamada por valor.

	<pre>/* Exemplo 22 - Estruturas e funcoes */ #include <stdio.h> struct reg { int a, b; char c; }; void funcao (struct reg param); main () { struct reg arg; arg.a = 1000; funcao (arg); printf ("\n%d", arg.a); } void funcao (struct reg param) { printf ("\n%d", param.a); param.a = 200; }</pre>
---	--

No exemplo anterior o argumento **arg** não terá seu valor alterado apesar do parâmetro **param** ter tido seu valor mudado dentro da **função**, pois as estruturas serão sempre passadas por valor.

Ponteiros para Estruturas

Como outros ponteiros, você declara ponteiros para estruturas colocando * na frente do nome da estrutura.


	<pre>struct conta { float saldo; char nome [30]; } cliente; struct conta *p; p = &cliente;</pre>
---	--

A instrução acima põe o endereço da estrutura cliente no ponteiro p. Para acessar os campos de uma estrutura usando um ponteiro para estrutura deve-se usar o operador ->. Exemplo: **p->nome**.



Palavra Chave: typedef

C admite que se defina explicitamente novos nomes aos tipos de dados utilizando a palavra chave typedef. Por exemplo:

	<pre>/* Exemplo 23 - Estruturas e funcoes com TYPEDEF */ #include <stdio.h> typedef struct { int a, b; char c; }reg; void funcao (reg param); main () { reg arg; arg.a = 1000; funcao (arg); printf ("\n%d", arg.a); } Void funcao (reg param) { printf ("\n%d", param.a); param.a = 200; }</pre>
---	---

4.4. Alocação Dinâmica de Memória

Existe em C, uma forma poderosa de se gerar espaço de memória dinamicamente, durante a execução de um programa. Isto é necessário quando nos programas temos quantidades de armazenamento variáveis. Por exemplo, uma pilha dentro de um programa qualquer.

A pilha deve crescer dinamicamente, conforme forem sendo empilhados elementos. Se usarmos o vetor como mecanismo para manutenção desta pilha, estaremos limitando o número de elementos que poderão ser mantidos na pilha. Se usarmos alocação dinâmica para os elementos da pilha, a única limitação que teremos será devido ao espaço de memória disponível (no computador) quando da execução do programa.



São basicamente duas as funções de alocação dinâmica de memória em C:

- **Malloc:** devolve o endereço do primeiro byte da memória alocado para uma variável tipo ponteiro qualquer.



```
char *p;  
p = malloc (1000);
```

Foram alocados com esta instrução 1000 bytes de memória sendo que **p** aponta para o primeiro destes 1000 bytes. Um string foi alocado dinamicamente. Sempre que se alocar memória deve-se testar o valor devolvido por malloc (), antes de usar o ponteiro, para garantir que não é nulo. Assim:



```
if (p=malloc (1000)) {  
    puts ("sem memoria\n");  
    exit (1);  
}
```

- **free:** A função free é a função simétrica de malloc, visto que ela devolve para o sistema uma porção de memória que foi alocada dinamicamente



```
/* Exemplo 24 - Alocacao dinamica de memoria */  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
main ()  
{  
    char *s, tamstr[6];  
    int t;  
    printf ("\n\nQual o tamanho do string ? ");  
    fgets (tamstr, 5, stdin);  
    t = atoi (tamstr);  
    s = malloc (t);  
    if (!s)  
    {  
        puts ("Sem memoria!");  
        exit (1);  
    }  
    printf ("Entre um string qualquer : ");  
    fgets (s, t, stdin);  
    printf ("Ao contrario fica..... : ");  
    for (t = strlen (s) - 1; t >= 0; t--)  
        putchar (s[t]);  
    free (s);  
    printf ("\n");  
}
```



Verifique o exemplo completo que implementa a estrutura de dados árvore binária por alocação dinâmica em linguagem C (esses dados são detalhados na disciplina de Estrutura de Dados):

E

/* Exemplo 25 - Arvore Binaria em linguagem C */

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

struct arv
{
    char info[30];
    struct arv *dir;
    struct arv *esq;
};

struct arv *constroi (void);
void preordem (struct arv *no);
void posordem (struct arv *no);
void ordeminter (struct arv *no);
void destroi (struct arv *no);
void mostra (struct arv *no, int nivel);
int menu (void);
void escreve (char *s);

main ()
{
    struct arv *ARVORE = NULL;
    int escolha;
    do
    {
        escolha = menu ();
        switch (escolha)
        {
            case 0:
                puts ("Constroi arvore\n\r");
                destroi (ARVORE);
                ARVORE = constroi ();
                break;
            case 1:
                puts ("Pre'-ordem\n\r");
                preordem (ARVORE);
                break;
            case 2:
                puts ("Pos ordem\n\r");
                posordem (ARVORE);
                break;
            case 3:
                puts ("Ordem intermediaria\n\r");
                ordeminter (ARVORE);
                break;
            case 4:
                puts ("Mostra arvore\n\r");
                mostra (ARVORE, 0);
                break;
        }
        puts ("\n\nDigite qualquer tecla...");
    }
```



```
        getchar ();
    }
    while (escolha != 5);
    destroi (ARVORE);
}

int menu (void)
{
    int opcao;
    puts ("Opcoes:");
    puts ("-----");
    puts ("0. Constroi arvore");
    puts ("1. Mostra arvore em Pre'-ordem");
    puts ("2. Mostra arvore em Pos-ordem");
    puts ("3. Mostra arvore em Ordem-intermediaria");
    puts ("4. Desenha a arvore");
    puts ("5. Fim de operacoes\n\n");
    do
    {
        printf ("Escolha [0,1,2,3,4 ou 5]: ");
        opcao = getchar () - 48;
        getchar ();
        puts ("\n\n");
    }
    while ((opcao < 0) && (opcao > 5));
    return opcao;
}

struct arv * constroi (void)
{
    struct arv *no;
    char auxstr[30];
    fgets (auxstr, 29, stdin);
    auxstr[strlen (auxstr) - 1] = '\0';
    if (strcmp (auxstr, ".") == 0)
        return NULL;
    else
    {
        no = malloc (sizeof (struct arv));
        strcpy (no->info, auxstr);
        no->esq = constroi ();
        no->dir = constroi ();
        return no;
    }
}

Void preordem (struct arv *no)
{
    if (no)
    {
        puts (no->info);
        preordem (no->esq);
        preordem (no->dir);
    }
}

Void posordem (struct arv *no)
```




```
{
    if (no)
    {
        posordem (no->esq);
        posordem (no->dir);
        puts (no->info);
    }
}

Void ordeminter (struct arv *no)
{
    if (no)
    {
        ordeminter (no->esq);
        puts (no->info);
        ordeminter (no->dir);
    }
}

void destroi (struct arv *no)
{
    if (no)
    {
        destroi (no->esq);
        destroi (no->dir);
        free (no);
        no = NULL;
    }
}

void mostra (struct arv *no, int nivel)
{
    int i;
    if (no)
    {
        mostra (no->dir, nivel + 1);
        for (i = 0; i < nivel; i++)
            printf (" ");
        puts (no->info);
        mostra (no->esq, nivel + 1);
    }
}
```

4.5. Exercícios

1. Usando vetor e aritmética de ponteiros desenvolva as seguintes funções:
 - copia (char *s1, char *s2) - função que copia s2 em s1.
 - void conc (char *s1, char *s2) - função que concatena s2 ao final de s1.
 - int tam (char *s) - função que retorna o tamanho do string s.
 - int compara (char *s1, char *s2) - função que retorna 0 se s1 <> de s2 e retorna 1 se s1 = a s2.
 - int pos (char *s, char c) - função que devolve a posição + 1 (POS) da primeira ocorrência de C no string S. Se o caracter C não está em S o valor 0 será retornado.



2. Desenvolver um programa em C para ler uma lista de números e calcular qual o maior dos números. OBS: desenvolver uma função que encontra o maior elemento num vetor de números que é passado para a função e fazer a chamada desta função no programa principal.
3. Desenvolver uma função em C que faz a fusão de dois vetores ordenados, dando o resultado no terceiro vetor. OBS: O terceiro vetor deve estar ordenado ao final da função.
4. Usando aritmética de ponteiros, faça uma função em C que reverte uma cadeia de caracteres. Assim: Antes: "TESTE" Depois: "ETSET"



5. Mecanismos de Entrada e Saída Avançados

Neste capítulo veremos a Entrada e Saída utilizando arquivos e a entrada e saída com modo gráfico. Com a entrada e saída em arquivos será possível armazenarmos de forma definitiva os dados trabalhados em nossos programas. A entrada e saída em modo gráfico permitirão desenvolver aplicações mais agradáveis e fáceis para o usuário final.

5.1. Entrada e Saída com Arquivo

O sistema de arquivo em C é definido para trabalhar com uma série de dispositivos: terminais, acionadores de disco, etc. Estes dispositivos são vistos como arquivos lógicos em C denominados **STREAM**. (abstração do dispositivo).

O dispositivo real é denominado **ARQUIVO** (impressora, disco, console). Um **STREAM** é associado a um **ARQUIVO** por uma operação de abertura do arquivo. Verifique as funções mais comuns no sistema de E/S ANSI:

Função	Descrição
fopen ()	abre um arquivo
fclose ()	fecha um arquivo
putc () e fputc ()	escreve um caracter num arquivo
getc () e fgetc ()	lê um caracter num arquivo
fseek ()	posiciona numa posição do arquivo
fprintf ()	impressão formatada em arquivo
fscanf ()	leitura formatada em arquivo
feof ()	verdadeiro se o fim do arquivo foi atingido
fwrite ()	escreve tipos maiores que 1 byte num arquivo
fread ()	lê tipos maiores que 1 byte num arquivo

Tabela 12 – Funções de manipulação de Arquivos



Ponteiro para Arquivo


O ponteiro de arquivo une o sistema de (entrada/saída) com um *buffer*. O ponteiro não aponta diretamente para o arquivo em disco e sim contém informações sobre o arquivo, incluindo nome, status (aberto, fechado) e posição atual sobre o arquivo.

Para se definir uma variável ponteiro de arquivo usa-se o seguinte comando:


	Sintaxe: <code>FILE *arq;</code>
---	--

Observação: com este comando passa a existir uma variável de nome **arq** que é ponteiro de arquivo.

Abrindo Arquivo

	Sintaxe: <code>fopen("nomedoarquivo", "mododeabertura");</code>
---	---

A função que abre arquivo em C é a função **fopen ()**. Ela devolve NULL (nulo) ou um ponteiro associado ao arquivo e deve ser passado para função o nome físico do arquivo e o modo como este arquivo deve ser aberto.

	<pre>if ((arq = fopen ("teste", "w")) == NULL) { puts ("Arquivo nao pode ser aberto"); exit (1); }</pre>
---	--

Observação: está sendo aberto um arquivo de nome “teste” para escrita (w – write).

Valores legais para **Mododeabertura**:

Modo	Significado
r	abre um arquivo texto para leitura
w	cria um texto para escrita
a	anexa um arquivo texto
rb	abre um arquivo binário para leitura
wb	cria um arquivo binário para escrita
ab	anexa um arquivo binário
r+	abre texto para leitura/escrita



w+	cria texto para leitura/escrita
a+	anexa texto para leitura/escrita
r+b	abre um arquivo binário para leitura/escrita
w+b	cria um arquivo binário para leitura/escrita
a+b	anexa um arquivo binário para leitura/escrita

Tabela 13 – Modo de abertura de arquivo

Fechando Arquivo

A função que fecha e esvazia o buffer de um arquivo é a função **fclose ()**. Exemplo:



```
/* Exemplo 26 - Le teclado, grava arquivo */
#include <stdio.h>
#include <stdlib.h>

main ()
{
    FILE * arq;
    char c, nome[20];
    printf ("Editor em C - digite ESC e ENTER para terminar.\n");
    printf ("Nome do arquivo? ");
    scanf ("%s", nome);
    if ((arq = fopen (nome, "w")) == NULL)
    {
        printf ("Erro abertura\n");
        printf ("Arquivo : %s\n", nome);
        exit (1);
    }
    do
    {
        c = getchar ();
        if (c == 13)
        {
            putchar ('\n', arq);
            puts ("");
        }
        else if (c != 27)
            putchar (c, arq);
    }
    while (c != 27);
    fclose (arq);
}
```





```
/* Exemplo 27 - Le de arquivo, mostra na tela */
#include <stdio.h>
#include <stdlib.h>

main ()
{
    FILE *arq;
    char nome[20];
    signed char c;
    int contalinha = 1;
    printf ("Nome do arquivo? ");
    scanf ("%s", nome);
    if ((arq = fopen (nome, "rt")) == NULL)
    {
        printf ("Erro de abertura\n");
        printf ("Arquivo: %s", nome);
        exit (1);
    }
    c = getc (arq);
    printf ("%4d:", contalinha);

    while (c != EOF)
    {
        if (c == 10)
        {
            contalinha++;
            printf ("\n%4d:", contalinha);
        }
        else putchar (c);
        c = getc (arq);
    }
    fclose (arq);
}
```

Funções fread() e fwrite()

Para ler e escrever estruturas maiores que 1 byte, usa-se as funções fread () e fwrite().



Sintaxe:

```
fread (buffer, tamanhoembytes, quantidade, ponteirodearquivo)
fwrite (buffer, tamanhoembytes, quantidade, ponteirodearquivo)
```

Onde:

- **buffer:** é um endereço na memória da estrutura para onde deve ser lido ou de onde serão escritos os valores (**fread ()** e **fwrite ()** respectivamente).
- **tamanhoembytes:** é um valor numérico que define o tamanho em bytes da estrutura que deve ser lida/escrita.



- **Quantidade:** número de estrutura que serão lidas ou escritas em cada processo de **fread** ou **fwrite**.
- **ponteirodearquivo:** é o ponteiro do arquivo onde será lida ou escrita uma estrutura.

Exemplo: programa que lê estruturas de dois campos: nome e telefone e vai gravando num arquivo denominado "ARQUIVO". Este processo é terminado com a digitação do campo nome vazio, quando então o programa apresenta todos os registros do arquivo que acaba de ser gerado.

E

```
/* Exemplo 28 - fread e fwrite */  
  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
#include <string.h>  
  
typedef struct  
{  
    char nome[30];  
    char telefone[20];  
} registro;  
  
main ()  
{  
    registro reg;  
    FILE * arq;  
    char numstr[5];  
    if ((arq = fopen ("arquivo", "w+b")) == NULL)  
    {  
        puts ("Erro na abertura do arquivo");  
        exit (1);  
    }  
    puts ("Digite enter no NOME para terminar");  
  
    do  
    {  
        printf ("\nEntre um nome..... : ");  
        fgets (reg.nome, 29, stdin);  
        reg.nome[strlen (reg.nome) - 1] = '\0';  
        printf ("Entre com telefone : ");  
        if (reg.nome[0])  
        {  
            fgets (reg.telefone, 19, stdin);  
            reg.telefone[strlen (reg.telefone) - 1] = '\0';  
            fwrite (&reg, sizeof (registro), 1, arq);  
        }  
    } while (reg.nome[0]);  
    rewind (arq);
```



```
printf ("\n\nListagem do arquivo\n");
puts ("-----\n");
printf ("%30s %20s\n", "NOME", "TELEFONE");
while (!feof (arq))
{
    fread (&reg, sizeof (registro), 1, arq);
    if (!feof (arq))
        printf ("%30s %20s\n", reg.nome, reg.telefone);
}
```

Manutenção de arquivo seqüencial em C



```
/* Exemplo 29 - Manutencao de arquivo em C */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct
{
    char nome[30];
    char telefone[15];
    int status;
} registro;
FILE * arq;
int insere (registro r);
void lista (void);
int retira (registro r);
void entra (registro * r);
int consulta (registro * r);
char menu (void);
void msg (char *s);
void escreve (char *s);

main ()
{
    registro reg;
    char opcao;
    if ((arq = fopen ("arquivo", "rb+")) == NULL)
        if ((arq = fopen ("arquivo", "wb+")) == NULL)
        {
            puts ("Erro na abertura do arquivo");
            exit (1);
        }
    do
    {
        opcao = menu ();
        printf ("\n\n");
        switch (opcao)
        {
            case '1':
                puts ("Insercao");
                entra (&reg);
                if (insere (reg))
```




```
        puts ("Insercao OK");
    else puts ("ERRO na insercao");
    break;
case '2':
    puts ("Remocao");
    printf ("\n Entre com um nome.... :");
    fgets (reg.nome, 29, stdin);
    reg.nome[strlen (reg.nome) - 1] = '\0';
    if (retira (reg))
        puts ("Remocao OK\n");
    else puts ("Erro na remocao\n");
    break;
case '3':
    puts ("Lista");
    lista ();
    break;
case '4':
    puts ("Consulta");
    printf ("\n Entre com um nome.... :");
    fgets (reg.nome, 29, stdin);
    reg.nome[strlen (reg.nome) - 1] = '\0';
    if (consulta (&reg))
    {
        printf ("\nNOME:%s, TEL:%s\n", reg.nome, reg.telefone);
        msg ("");
    }
    else msg ("NOME nao encontrado\n");
}

while (opcao != '5');
puts ("fim do programa...");
fclose (arq);
}

char menu (void)
{
    char opcao;
    puts ("\nMENU :      ");
    puts ("1. Insere  ");
    puts ("2. Retira  ");
    puts ("3. Lista   ");
    puts ("4. Consulta");
    puts ("5. Fim");
    printf ("\nEscolha ==>");
    opcao = getchar ();
    getchar ();
    return opcao;
}

void msg (char *s)
{
    puts (s);
    puts ("Digite qualquer tecla...");
    getchar ();
}

void entra (registro * r)
```



```
{
    printf ("\n Entre com um nome.... :");
    fgets (r->nome, 29, stdin);
    r->nome[strlen (r->nome) - 1] = '\0';
    printf (" Entre com um telefone :");
    fgets (r->telefone, 14, stdin);
    r->telefone[strlen (r->telefone) - 1] = '\0';
    r->status = 1;
}

int insere (registro r)
{
    if (!fseek (arq, 0, SEEK_END))
    {
        fwrite (&r, sizeof (registro), 1, arq);
        return 1;
    }
    return 0;
}

int retira (registro r)
{
    registro reg;
    fseek (arq, 0, SEEK_SET);
    while (!feof (arq))
    {
        fread (&reg, sizeof (registro), 1, arq);
        if (!strcmp (reg.nome, r.nome) && (reg.status))
        {
            reg.status = 0;
            fseek (arq, -1L * sizeof (registro), SEEK_CUR);
            if (fwrite (&reg, sizeof (registro), 1, arq))
                return 1;
        }
    }
    return 0;
}

void lista (void)
{
    int i;
    registro r;
    rewind (arq);
    printf ("\n%-30s %-15s\n", "Nome", "Telefone");
    for (i = 0; i < 46; i++)
        printf ("-");
    printf ("\n");
    while (!feof (arq))
    {
        fread (&r, sizeof (registro), 1, arq);
        if ((r.status) && (!feof (arq)))
            printf ("\n%-30s %-15s", r.nome, r.telefone);
    }
    printf ("\n\n");
}

int consulta (registro * r)
```



```
{
    registro reg;
    fseek (arq, 0, SEEK_SET);
    while (!feof (arq))
    {
        fread (&reg, sizeof (registro), 1, arq);
        if (!strcmp (reg.nome, r->nome) && (reg.status))
        {
            *r = reg;
            return 1;
        }
    }
    return 0;
}
```

5.2. Entrada e Saída com Modo Gráfico

Nada como um gráfico para facilitar o entendimento de coisas abstratas. Em Computação, assim como em Engenharia e outras áreas, usa-se modelos gráficos como DFD, plantas e outros, para facilitar o planejamento de sistemas, construções e outros.

Cada compilador C tem sua biblioteca gráfica específica. Isto se deve ao fato da resolução gráfica ser uma característica dependente da máquina. Seria muito complicado tratar todos os tipos de dispositivos gráficos num único pacote. A biblioteca para as versões Borland Turbo C, é a "graphics.h".

Funções da Biblioteca Gráfica

- **initgraph**: inicializa o modo gráfico e necessita de 3 parâmetros. O 'graphdriver' indica o driver gráfico de cada monitor (EGA,VGA,SVGA e outros). Estes driver's estão disponíveis em arquivos com extensão **.BGI**. O parâmetro 'graphmode' define o modo gráfico mais indicado para cada monitor. O parâmetro 'pathstring' indica o caminho nos diretórios onde se encontra o driver gráfico.
- **DETECT**: macro que efetua a autodetecção de modo gráfico, garantindo a portabilidade.
- **graphresult**: função que retorna um código de erro para a última operação gráfica executada num programa.



- **setviewport:** define uma região da tela gráfica como área ativa.
- **settextstyle:** define o estilo (tamanho, tipo, direção) dos textos (strings) que serão exibidos no modo gráfico.
- **setcolor:** especifica uma cor (paleta de cores) para trabalho no modo gráfico.
- **setfillstyle:** ajusta um padrão e cor de preenchimento de figuras.
- **line:** desenha uma linha entre dois pontos especificados. Desenha a linha de (X1, Y1) a (X2, Y2) usando a cor e estilo correntes.
- **putpixel:** acende um pixel (picture element) num ponto específico.
- **getpixel:** retorna a cor de um pixel (X,Y) da tela gráfica.
- **rectangle:** desenha um retângulo em modo gráfico, utilizando a cor corrente.
- **lineto:** desenha uma linha entre a posição corrente do indicador gráfico e os pontos (X,Y) definidos, utilizando a cor corrente.
- **moveto:** move o indicador gráfico para a posição (X,Y) definida.
- **bar:** desenha uma barra (retângulo preenchido) em modo gráfico, utilizando a cor corrente.
- **circle:** traça um círculo, utilizando a cor corrente, de acordo com um raio especificado.
- **floodfill:** preenche uma figura fechada a partir de um ponto.
- **outtextxy:** escreve um string numa certa posição da tela, em modo gráfico.
- **outtext:** escreve um string na posição corrente do indicador de tela, em modo gráfico.
- **getmaxx:** retorna a resolução (número máximo de pixels) na horizontal (X), da tela gráfica.
- **getmaxy:** retorna a resolução (número máximo de pixels) na vertical (Y), da tela gráfica.
- **closegraph:** sai do modo gráfico.





```
/* Exemplo 30 - Apresenta o uso das funções gráficas */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>

void DefineJanela(int X1, int Y1, int X2, int Y2,
                 unsigned char CorFundo, unsigned char CorBorda);

void main (void)
{
    int DriverGrafico, ModoGrafico, Erro;
    int X,Y;
    int TrocaCor,ContadorPixels;
    unsigned char Cor;
    DriverGrafico = DETECT;
    initgraph (&DriverGrafico,&ModoGrafico,"c:\\tc\\bgi");
    Erro = graphresult ();
    if (Erro != grOk)
    {
        printf ("Erro de Abertura do Modo Grafico");
        exit (1);
    }

    /* Definindo uma primeira janela (area grafica ativa) */
    /* e utilizando comandos graficos cartesianos */
    DefineJanela(0,0,300,220,7,15);
    setfillstyle (1,0); setcolor(0);
    circle(50,50,20);
    setfillstyle (1,4); setcolor(4);
    circle(100,50,20); floodfill(100,50,4);
    setfillstyle(1,2); rectangle(200,30,280,70);
    setfillstyle(1,14); setcolor(14);
    line(30,140,30,200);
    line(30,200,130,200);
    line(130,200,30,140);
    setfillstyle(1,4); setcolor(4);
    moveto(30,140); lineto(100,140);
    setfillstyle(1,3); setcolor(3);
    bar(200,130,280,170);
    getch();
    /* Definindo uma segunda janela (area grafica ativa) */
    DefineJanela(100,100,400,320,6,15);
    setfillstyle(1,15); setcolor(15);
    outtextxy(40,30,"Usando recursos graficos da");
    outtextxy(40,42,"Linguagem C...");
    setfillstyle(1,11); setcolor(11);
    outtextxy(40,80,"Usando recursos graficos da");
    outtextxy(40,92,"Linguagem C...");
    setfillstyle(1,3); setcolor(3);
    outtextxy(40,130,"Usando recursos graficos da");
    outtextxy(40,142,"Linguagem C...");
    setfillstyle(1,14); setcolor(14);
    moveto(156,142); outtext("e C++");
    setttextstyle(0,HORIZ_DIR,3);
    outtextxy(30,170,"OK");
    setttextstyle(1,HORIZ_DIR,3);
```



```
outtextxy(80,170,"OK");
settextstyle(2,HORIZ_DIR,3);
outtextxy(130,170,"OK");
settextstyle(3,HORIZ_DIR,3);
outtextxy(180,170,"OK");
settextstyle(4,HORIZ_DIR,3);
outtextxy(230,170,"OK");
getch();
/* Definindo uma terceira janela (area grafica ativa) */
DefineJanela(200,200,500,420,1,15);
randomize();
ContadorPixels = 0; TrocaCor = 0;
do
{
    X = random(getmaxx());
    Y = random(getmaxy());
    if (!TrocaCor)
        Cor = 15;
    else
        Cor = 8;
    if ((getpixel(X,Y) != 8) && (getpixel(X,Y) != 15))
    {
        setfillstyle(1,Cor); setcolor(Cor); putpixel(X,Y,Cor);
    }
    if ((getpixel(X,Y) != 15) && (getpixel(X,Y) != 8))
    {
        setfillstyle(1,Cor); setcolor(Cor); putpixel(X,Y,Cor);
    }
    ContadorPixels++;

    if (ContadorPixels == 100)
    {
        TrocaCor = !TrocaCor;
        ContadorPixels = 0;
    }
} while (!kbhit());
getch();

closegraph();
}

void DefineJanela(int X1, int Y1, int X2, int Y2,
    unsigned char CorFundo, unsigned char CorBorda)
{
    setviewport(0,0,getmaxx(),getmaxy(),1);
    setfillstyle(1,CorFundo); setcolor(CorFundo);
    bar(X1,Y1,X2,Y2);
    setfillstyle(1,CorBorda); setcolor(CorBorda);
    rectangle(X1,Y1,X2,Y2);
    setviewport(X1,Y1,X2,Y2,1);
}
```



Rotinas de Mouse integradas ao modo Gráfico

Mesmo com toda a capacidade de C, existem momentos em que você precisa acessar diretamente os recursos do sistema operacional e do ambiente hospedeiro. Frequentemente, um dispositivo específico, ou mesmo uma função do sistema operacional, precisa ser utilizado sem que haja uma forma de fazê-lo com as rotinas existentes na biblioteca.

Cada processador, sistema operacional e ambiente têm seus próprios métodos de acesso aos recursos do sistema. Quando se deseja uma forma mais portátil de manipulação destes recursos é aconselhado o uso de rotinas pré-definidas do sistema, que se encontram disponíveis a partir de um vetor de interrupção. Para acessar estas rotinas basta carregar os registradores de uso geral com os parâmetros de chamada da rotina, e então executar a chamada da rotina via interrupção.

Em C temos as seguintes funções para chamada de interrupção: **bdos()**, **int86()** e **intr()**. Para o programa exemplo vamos acessar as rotinas de manipulação de mouse, via interrupção 0x33, usando a função **intr()** pré-definida em C para acesso aos recursos do sistema operacional.



```
/* Exemplo 31 - Apresenta Rotinas de Mouse (Modo Gráfico) */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>
#include <graphics.h>

unsigned int IniciaMouse (void);
int CoordX (void);
int CoordY (void);
void IndicadorMouse (void);
void SemIndicadorMouse (void);
int BotaoCentral (void);
int BotaoDireita (void);
int BotaoEsquerda (void);

void main (void)
{
    int DriverGrafico, ModoGrafico, Erro;
    int X,Y;
    unsigned int MouseInstalado, Finaliza;
    int BotaoEsq, BotaoDir;
```





```
DriverGrafico = DETECT;
initgraph (&DriverGrafico,&ModoGrafico,"c:\\tc\\bgi");
Erro = graphresult ();
if (Erro != grOk)
{
    printf ("Erro de Abertura do Modo Grafico");
    exit (1);
}
MouseInstalado = IniciaMouse();
if (!MouseInstalado)
    exit(1);

setfillstyle(1,15); setcolor(15);
rectangle(0,0,100,50);
rectangle(0,60,100,110);
rectangle(0,120,100,170);
setfillstyle(1,11); setcolor(11);
outtextxy(20,22,"Mensagem");
outtextxy(20,82," Avisos ");
outtextxy(20,142,"Finaliza");
IndicadorMouse();
Finaliza = 0;
do
{
    BotaoEsq = 0;
    BotaoDir = 0;
    BotaoEsq = BotaoEsquerda();
    BotaoDir = BotaoDireita();

    if ((CoordX() >= 0) && (CoordX() <= 100) &&
        (CoordY() >= 0) && (CoordY() <= 50) &&
        (BotaoEsq != 0))
    {
        setfillstyle(1,7); setcolor(7);
        SemIndicadorMouse();
        bar(200,80,460,100);
        setfillstyle(1,0); setcolor(0);
        outtextxy(215,88,"Bem vindo ao mundo grafico...");
        IndicadorMouse();
    }

    if ((CoordX() >= 0) && (CoordX() <= 100) &&
        (CoordY() >= 60) && (CoordY() <= 110) &&
        (BotaoEsq != 0))
    {
        setfillstyle(1,7); setcolor(7);
        SemIndicadorMouse();
        bar(200,80,460,100);
        setfillstyle(1,4); setcolor(4);
        outtextxy(210,88,"Programe com cautela e atencao");
        IndicadorMouse();
    }

    if ((CoordX() >= 0) && (CoordX() <= 100) &&
        (CoordY() >= 120) && (CoordY() <= 170) &&
        (BotaoEsq != 0))
        Finaliza = 1;
```



E

```
} while ((!Finaliza) && (BotaoDir == 0));
IndicadorMouse();
closegraph ();
}

unsigned int IniciaMouse (void)
{
    struct REGPACK Registradores;
    Registradores.r_ax = 0;
    intr (0x33, &Registradores);
    return Registradores.r_ax;
}

int CoordX (void)
{
    struct REGPACK Registradores;
    Registradores.r_ax = 03;
    intr (0x33, &Registradores);
    return Registradores.r_cx;
}

int CoordY (void)
{
    struct REGPACK Registradores;
    Registradores.r_ax = 03;
    intr (0x33, &Registradores);
    return Registradores.r_dx;
}

void IndicadorMouse (void)
{
    union REGS Registradores;
    Registradores.x.ax = 0x01;
    Registradores.x.bx = 2;
    int86(0x33,&Registradores,&Registradores);
}

void SemIndicadorMouse (void)
{
    union REGS Registradores;
    Registradores.x.ax = 0x02;
    int86 (0x33,&Registradores,&Registradores);
}

int BotaoCentral (void)
{
    union REGS Registradores;
    Registradores.x.ax = 0x05;
    Registradores.x.bx = 2;
    int86(0x33,&Registradores,&Registradores);
    return Registradores.x.bx;
}

int BotaoDireita (void)
{
    union REGS Registradores;
    Registradores.x.ax = 0x05;
```



```
Registradores.x.bx = 1;
int86(0x33, &Registradores, &Registradores);
return Registradores.x.bx;
}

int BotaoEsquerda (void)
{
    union REGS Registradores;
    Registradores.x.ax = 0x05;
    Registradores.x.bx = 0;
    int86(0x33, &Registradores, &Registradores);
    return Registradores.x.bx;
}
```

5.3. Exercícios

1. Modifique o exemplo 29 para manutenção de um arquivo seqüencial com os seguintes campos: nome (string de 20 posições), endereco (string de 20 posições), cidade (string de 20 posições), estado (string de 2 posições) e cep (string de 9 posições).
2. Implementar o exemplo 30 do modo gráfico.
3. Implementar o exemplo 31 do modo gráfico.



REFERÊNCIAS

FARRER, Harry. Pascal Estruturado. Ed. LTC. Rio de Janeiro, 1999.

SCHILDT, HERBERT. C completo e total. São Paulo. MacGrawHill, 1990.

SCHILDT, HERBERT. Linguagem C: Guia do Usuário. São Paulo. MacGrawHill, 1986.

SWAN, TOM. Aprendendo C++. São Paulo. MacGrawHill, 1993.

BERRY, JOHN THOMAS. Programando em C++. São Paulo, MacGrawHill, 1991.

TANENBAUM, AARON M. Estruturas de dados usando C. São Paulo, MAKRON BOOKS, 1995.

Web Sites:

<http://ead1.eee.ufmg.br/cursos/C>

<http://www.portalc.na-web.net>

