INTRODUÇÃO

A Informática nasceu da idéia de auxiliar o homem nos trabalhos rotineiros e repetitivos, em geral de cálculo e gerenciamento. E hoje é conceituada como Ciência que estuda o tratamento automático da informação.

Assim, o elemento físico utilizado para o tratamento de dados e obtenção de informação, é o **computador**. E o processo de avaliação, contagem, ordenação e classificação de dados e informações é conhecido por **computação**.

Para nos comunicarmos com o computador devemos utilizar uma linguagem de programação, isto é, uma liguagem que o computador entenda. Temos dois tipos de linguagens:

- LINGUAGEM DE MÁQUINA: O computador, internamente, possui uma linguagem característica, baseada em grandezas matemáticas do sistema binário (0 e 1). Programar em linguagem de máquina significa comunicar-se com uma linguagem bem próxima a que a máquina entende.
- LINGUAGEM DE ALTO NÍVEL: São instruções semelhantes a linguagem natural que ao serem passadas para o computador são transferidas para linguagem de máquina para que possam ser atendidas.

Os **tradutores** são programas capazes de entender os programas escritos em linguagem de alto nível. Temos dois tipos de tradutores:

- COMPILADORES: Realizam a tradução de um conjunto completo de instruções criando um novo conjunto, todo em linguagem de máquina, que será usado na execução do programa. Exemplo de linguagens compiladas: Pascal, Fortran, Cobol, C e outras.
- INTERPRETADORES: Realizam a tradução e imediata execução de uma instrução não gerando um novo conjunto em linguagem de máquina. Exemplo de linguagem interpretada: Basic.

LINGUAGEM C/C++

Segundo Schildt (1996), Dennis Ritchie inventou a linguagem C e foi o primeiro a implementá-la utilizando um computador com sistema operacional UNIX.

C é resultante de um processo evolutivo de linguagens e seu marco inicial foi uma linguagem chamada BCPL, desenvolvida por Martin Richards, que teve fortes influências em uma linguagem chamada B, inventada por Ken Thompson. Na década de 70, B levou ao desenvolvimento de C.

Durante alguns anos o padrão da linguagem C foi aquele fornecido com a versão 5 do sistema operacional UNIX. Com a popularização dos microcomputadores várias implementações de C foram criadas gerando assim muitas discrepâncias. Para resolver este problema o ANSI (American National Standards Institute), estabeleceu em 1983 um comitê para definir um padrão que guiasse todas as implementações da linguagem C.

A linguagem C++ é uma extensão da linguagem C. As instruções que fazem parte da linguagem C representam um subconjunto de C++. Os incrementos encontrados na linguagem C++ foram feitos para dar suporte à programação orientada a objetos. A sintaxe da linguagem C++ é basicamente a mesma da linguagem C.

CAPÍTULO 1: CONCEITOS INICIAIS

1.1 Algoritmo

Algoritmo é o conjunto de ações primitivas (instruções) organizadas de forma lógica, estruturada e bem definida, expressa em linguagem natural, que tem por finalidade resolver um problema, ou seja, é a descrição em português das ordens que o programa deve realizar.

Exemplo: Algoritmo de um saque em um caixa eletrônico

1. Início
2. Passe o cartão
3. Abra a porta
4. Entre no caixa
5. Feche a porta
6. Insira o cartão no local indicado
7. Digite sua senha
8. Escolha a opção saque
9. Escolha o valor
10. Apanhe o dinheiro
11. Apanhe o comprovante e o cartão
12. Abra a porta
13. Saia do caixa
14. Feche a porta
15. Fim

Um algoritmo está completo se os seus comandos (instruções) forem do entendimento do seu destinatário, caso contrário, o comando não entendido deverá ser desdobrado em novos comandos, até que todos os comandos sejam entendidos. Este ato de desdobrar os comandos é chamado **refinamento sucessivo** do comando inicial.

Exemplo: A seqüência de Fibonacci se define como tendo os dois primeiros termos iguais a 1 e cada termo seguinte é igual a soma dos dois termos imediatamente anteriores. Fazer um algoritmo que escreva os termos de Fibonacci inferiores a L.

```
Início
    1 Escreva os termos de Fibonacci inferiores a L
Fim
```

Refinando **1** temos

```
1 Ref.   Escreva os termos de Fibonacci inferiores a L
         Receba o valor de L
         2 Processe os dois primeiros termos
         3 Processe os termos restantes                    Estrutura Seqüencial
Fim Ref.
```

Devemos refinar **2** e **3**

```
2 Ref.  Processe os dois primeiros termos
        Atribua o valor 1 ao primeiro termo
        Se ele for menor que L
            então escreva-o                                Estrutura Condicional
        Atribua o valor 1 ao segundo termo
        Se ele for menor que L
            então escreva-o
Fim Ref.
```

```
3 Ref.  Processe os termos restantes
        Repita
              Calcule o novo termo somando os 2 anteriores
              Escreva o novo termo                          Estrutura de Repetição
        Até que o novo termo seja maior ou igual a L
Fim Ref.
```

Supondo que o destinatário entenda os comandos de atribuição e adição de dois termos, o algoritmo está completo e tem a seguinte estrutura:

```
Início
                { Dado de Entrada }
        Receba o valor de L

                { Processamento dos dois primeiros termos }
        Atribua o valor 1 ao primeiro termo
        Se ele for menor que L
              então
                    escreva-o
        Atribua o valor 1 ao segundo termo
        Se ele for menor que L
              então
                    escreva-o

                { Processamento dos termos restantes  }
        Repita
                Calcule o novo termo somando os 2 anteriores
                Escreva o novo termo
        Até que o novo termo seja maior ou igual a L
Fim
```

Assim, um algoritmo e seus refinamentos são formados por **comandos** (determinam ações a serem executadas) e por **estruturas de controle** (determinam a ordem em que os comandos devem ser executados, se devem ou não ser executados e quando devem ser repetidos).

1.2 Estrutura de um Programa em C/C++

Um programa escrito na linguagem de programação C/C++ consiste em uma coleção de funções, sendo que a main() é a primeira função a ser executada. Além disso, C/C++ é *case-sensitive*, ou seja, diferencia letras maiúsculas de letras minúsculas (por exemplo, int ≠ Int). Um programa em C/C++ mínimo consiste em:

```
main ()
{
}
```

Este programa define a função *main*, que não possui argumentos e não faz nada. As chaves, { }, são usadas para expressar agrupamentos. No caso do exemplo, elas indicam o início e o fim do corpo da função *main*, que está vazia, isto é, não faz nada. Cada programa em C/C++ deve ter uma função *main*.

Assim, a estrutura genérica de um programa em C/C++ é a que se apresenta a seguir, podendo alguns dos elementos não existir:

- Comandos do pré-processador;
- Definições de tipos;
- Protótipos de funções: declaração dos tipos de retorno e dos tipos dos parâmetros das funções;
- Declaração de variáveis globais;
- Um bloco de instruções principais: main
- Blocos de funções;
- Documentação do programa: comentários;

Em C/C++, existem comandos que são processados durante a compilação do programa (pré-processador). Estes comandos são genericamente chamados de **diretivas de compilação**. Tais comandos informam ao compilador do C/C++ basicamente quais são as constantes simbólicas usadas no programa e quais bibliotecas devem ser anexadas ao programa executável.

A diretiva #include <filename> diz ao compilador para incluir na compilação do programa o arquivo **filename.** Geralmente estes arquivos contêm bibliotecas de funções ou rotinas do usuário. Bibliotecas são arquivos contendo várias funções que podem ser incorporadas aos programas. Os arquivos terminados em "**.h**" são chamados ***headers*** (ou cabeçalhos).

Estrutura Básica:

```
#include <nome da biblioteca>
void main( )
{
  bloco de comandos;
}
```

CAPÍTULO 2: CONCEITOS BÁSICOS DE PROGRAMAÇÃO

2.1 Identificadores

Em C/C++, os nomes de variáveis, funções, rótulos e vários outros objetos definidos pelo usuário são chamados de **identificadores**. A escolha dos nomes desses identificadores deve ser feita seguindo as regras:

- Um identificador deve iniciar por uma letra ou por um "_" (underscore);
- A partir do segundo caracter pode conter letras, números e underscore;
- Deve-se usar nomes significativos dentro do contexto do programa;
- Lembrar que C/C++ é uma linguagem case-sensitive (letras maiúsculas ≠ letras minúsculas);
- Costuma-se usar maiúsculas e minúsculas para separar palavras;
- Deve ser diferente das palavras reservadas;
- O padrão C ANSI (American National Standards Institute - comitê que definiu um padrão para a linguagem C/C++) determina que os identificadores podem ter qualquer tamanho, mas se for um nome externo (nome de função e variáveis globais compartilhadas entre arquivos), pelo menos os 6 primeiros caracteres devem ser significativos, e se o identificador for um nome interno os primeiros 31 caracteres são significativos;

Exemplos de identificadores:

Correto	Incorreto
cont	1cont
Teste23	Oi!
usuário_1	usuário...1
RaioDoCirculo	Raio Do Circulo

As palavras reservadas não podem ser utilizadas como identificadores, pois são de uso restrito da linguagem C/C++ (comandos, estruturas, declarações, etc.). O conjunto de palavras reservadas do C/C++ padrão é:

auto	**double**	**if**	**static**
break	**else**	**int**	**struct**
case	**entry**	**long**	**switch**
char	**extern**	**register**	**typedef**
continue	**float**	**return**	**union**
default	**for**	**sizeof**	**unsigned**
do	**goto**	**short**	**while**

2.2 Tipos Básicos de Dados

Cada linguagem de programação possui tipos de dados pré-definidos. Assim para desenvolvermos **algoritmos** utilizaremos os seguintes tipos de dados:

INTEIRO = armazena qualquer valor pertencente ao conjunto dos números inteiros.
REAL = armazena qualquer valor pertencente ao conjunto dos números reais.
CARACTER = armazena **apenas 1 elemento** pertencente a um dos conjuntos abaixo:
{A, ..., Z}, {0, ..., 9}, {*, ?, &, =, +, $, -, /, ', ",}
LITERAL = Armazena **1 seqüência de caracteres**.
BOOLEANO = Armazena apenas as constantes FALSO ou VERDADEIRO.

Em C/C++ existem cinco tipos básicos de dados: caracter, inteiro, ponto flutuante, ponto flutuante de precisão dupla e sem valor (**char**, **int**, **float**, **double** e **void**, respectivamente). O tipo void declara explicitamente uma função que não retorna valor algum ou cria ponteiros genéricos. Valores do tipo **char** são normalmente usados para conter valores definidos pelo conjunto de caracteres ASCII,

Esta linguagem não possui o tipo de dado **booleano**, pois considera qualquer valor diferente de zero como sendo verdadeiro. **C** não possui também um tipo especial para armazenar cadeias de caracteres (literais, strings). Deve-se, quando necessário, utilizar um vetor contendo vários elementos do tipo **char**. Os vetores serão abordados posteriormente.

Segundo Schildt, o tamanho e a faixa desses tipos de dados variam de acordo com o tipo de processador e com a implementação do compilador C. Um caracter ocupa geralmente 1 byte e um inteiro tem normalmente 2 bytes, mas não podemos fazer esta suposição se desejarmos que nossos programas sejam portáveis a uma grande quantidade de computadores. O padrão ANSI estipula apenas a **faixa mínima** de cada tipo de dado, não o seu tamanho em bytes.

A partir dos cinco tipos básicos podemos definir outros tipos. A tabela a seguir apresenta os tipos de dados definidos no padrão ANSI.

TIPO	FAIXA DE VALORES	TAMANHO EM BYTES
char	-127 a 127	1
unsigned char	0 a 255	1
signed char	O mesmo que **char**	1
int	-32.767 a 32767	2
unsigned int	0 a 65.535	2
signed int	O mesmo que **int**	2
short int	O mesmo que **int**	2
unsigned short int	0 a 65.535 (o mesmo que **unsigned int**)	2
signed short int	O mesmo que **short int** e **int**	2
long int	-2.147.483.648 a 2.147.483.647	4
unsigned long int	0 a 4.294.967.295	4
signed long int	O mesmo que **long int**	4
float	3,4 E-38 a 3,4E+38 (6 dígitos de precisão)	4
double	1,7 E-308 a 1,7E+308 (10 dígitos de precisão)	8
long double	3,4 E-4.932 a 1,1E+4.932 (10 dígitos de precisão)	10

Geralmente nos sistemas UNIX os tipos **int** e **long int** são equivalentes (inteiros de 4 bytes). No entanto noutros sistemas é possível que o tipo **int** seja equivalente a um **short int** (inteiro de 2 bytes). Aqui adotaremos o padrão ANSI.

Exceto o **void**, os tipos de dados básicos (como mostra a tabela) podem ter vários modificadores precedendo-os. Um **modificador** é utilizado para alterar o significado de um tipo básico para adaptá-lo mais precisamente às necessidades de diversas situações. A lista de modificadores é composta por: **signed, unsigned, long** e **short**. Todos estes podem ser aplicados aos tipos **char** e **int**, e o **long** também pode ser aplicado ao **double**.

O padrão ANSI elimina o **long float** porque é igual a **double** O uso de **signed** com **int** é permitido, mas redundante, pois a declaração de um **int** assume um número com sinal. O uso mais importante de **signed** é para modificar o **char**.

2.3 Variáveis

Uma **variável** é uma posição de memória que pode ser identificada através de um nome, e é usada para guardar um valor. O conteúdo de uma variável pode ser alterado através de um comando de atribuição, ou seja, após uma atribuição a variável muda de valor. É interessante comentar que não há uma inicialização implícita na declaração, mas a variável pode ser inicializada na declaração. Todas as variáveis em C/C++ devem ser declaradas antes de serem usadas. A forma geral de uma declaração é:

tipo lista_de_variáveis;

sendo:

tipo = tipo de dado válido em C/C++;
lista_de_variáveis = pode consistir em um ou mais nomes de identificadores separados por vírgula.

A declaração de variáveis pode ser feita em três lugares básicos: dentro de funções (variáveis locais), na definição dos parâmetros das funções (parâmetros formais) e fora de todas as funções (variáveis globais).

Exemplos:

Algoritmo	C
x : real	float x;
y, z : real	float y, z;
SomaGeral : inteiro	int SomaGeral;
sexo : caracter	char sexo;
nome : literal [40]	char nome[40];

Note que em C/C++ após cada comando **deve-se colocar**; (ponto e vírgula).

2.4 Constantes

São identificadores que não podem ter seus valores alterados durante a execução do programa. As constantes em C/C++ podem ser de qualquer um dos cinco tipos de dados básicos. Porém, a maneira como cada constante é representada depende do seu tipo.

As constantes de caracteres são envolvidas por apóstrofos (‘) e, as cadeias de caracteres são representadas entre aspas (“).

Para criar uma constante existe o comando **#define** que, em geral é colocado no início do programa fonte, ou seja, no seu cabeçalho.

Exemplos:

Algoritmo:	C:	
LARGURA_MÁXIMA = 50	#define LARGURA_MÁXIMA	50
FALSO = 0	#define FALSO	0
VERDADEIRO = 1	#define VERDADEIRO	1
CURSO = “BSI”	#define CURSO	“BSI”
TERMINO = ‘T’	#define TERMINO	‘T’
VALOR_DE_PI = 3,1415	#define VALOR_DE_PI	3.1415

Observe que **não se coloca** ponto e vírgula após o valor.

2.5 Comandos de Entrada e Saída

Os principais comandos (funções) de entrada e saída são utilizados para escrever no vídeo e ler do teclado, respectivamente, o valor de variáveis. Para que o programa possa fazer uso desses comandos é necessário utilizar a biblioteca **<stdio.h>.**

O comando de **entrada** é utilizando para receber dados digitados pelo usuário. Os dados recebidos são armazenados em variáveis. Os comandos de entrada mais utilizados são:

scanf: lê todos os tipos de dados;
gets: lê uma seqüência de caracteres (string) até que seja pressionado enter.

Apesar do comando **scanf** poder ser utilizado para qualquer tipo de dados, seu uso para **char** ou **string** pode resultar em problemas, devido ao fato de armazenar no buffer o comando <enter>. Este problema é resolvido se antes, ou logo após sua utilização seja acrescentado o comando **fflush(stdin);**.

O comando de **saída** é utilizado para mostrar os dados na tela ou na impressora, e seu principal comando é **printf**. Seus parâmetros são de dois tipos: o primeiro formado por caracteres que serão exibidos na tela, e o segundo contém comandos de formato que definem a maneira pela qual os argumentos subseqüentes serão mostrados. O comando de formato começa por **%** e é seguido pelo código de formato. Os principais códigos são:

scanf	**printf**
%c – caracter (char)	**%c** – char
%d – números inteiros (int)	**%d** – int
%f, %e, %g – números reais (float)	**%u** – inteiro sem sinal (unsigned int)
%lf, %le, %lg – números reais (double)	**%f** – float
%s – seqüência de caracteres	**%e** - double ou float no formato científico
	%g – double ou float no formato mais apropriado (%f ou %e)
	%s – seqüência de caracteres

Os comandos de formato podem ser alterados, por exemplo, para especificação da largura mínima de campo ou do número de casas decimais.

Exemplos:

1)

```
#include <stdio.h>
#include <conio.h>
float x;
main ()
{
   x=10.1234;
   printf("%f \n", x);
   printf("%5.2f \n", x);
   printf("Conteudo de x = %5.2f \n", x);
   getch();
}
```

Saída:

```
10.123400
10.2
Conteúdo de x = 10.12
```

2)

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

int idade;
char nome[30];

main ()
{
   printf("Digite sua idade: ");
   scanf("%d",&idade);
   printf("Digite seu nome: ");
   scanf("%s",nome);
   printf("%s sua idade eh %d anos. \n", nome, idade);
   getch();
}
```

Saída:

```
Digite sua idade: 20
Digite seu nome: Maria
Maria sua idade eh 20 anos.
```

Observe que em strings não se utiliza **&** na leitura do dado. As chaves **{ }** indicam início e fim de blocos de comando.

A função **getch()** esperá até que uma tecla seja pressionada e não mostra o caracter na tela. Aqui ela foi utilizada para ver o que estava escrito na tela.

É importante comentar que C/C++ possui constantes especiais de caracter de barra invertida para caracteres que não podem ser impressos. Os principais são:

- **\b** – retrocesso (BS);
- **\f** – salto de página (Form Feed - FF);
- **\n** – linha nova (line Feed - LF);
- **\r** – retorno de carro (CR);
- **\t** – tabulação horizontal (TAB)
- **\"** – aspas duplas;
- **\´** - aspas simples;
- **\0** – nulo;
- **** - barra invertida;
- **\v** – tabulação vertical;
- **\a** – alerta (beep);

2.6 Comentários

Comentários são textos que podem ser inseridos em programas com o objetivo de documentá-los e não são analisados pelo compilador. Podem ocupar uma ou várias linhas devendo ser inseridos nos programas utilizando **/*... */** ou **//.**

Exemplos:

1)

```
/*
  linhas de comentários
  linhas de comentários
*/
```

2)

```
// comentário
```

Em **(1)** a região de comentários é aberta com **/*** e fechada com ***/**.
Em **(2)** a região de comentário é aberta por **/** e encerrada automaticamente ao final da linha.

2.7 Comando de Atribuição

Este comando é utilizado para atribuir valores ou operações às variáveis, sendo representado por = (sinal de igualdade).

A linguagem C/C++ permite que você atribua o mesmo valor a muitas variáveis usando atribuições múltiplas em um único comando.

Exemplos:

```
x = 4;
x = x + 2;
y = z = 2.5;
sexo = 'F';
```

Caso seja necessário armazenar uma cadeia de caracteres dentro de uma variável, deve-se utilizar uma função para manipulação de caracteres:

```
strcpy ( nome, "João" );
```

Para que seja possível a utilização da função **strcpy** deve-se inserir ao programa, por meio da diretiva **#include,** a biblioteca **<string.h>**. As funções de manipulação de strings serão abordadas posteriormente.

2.7 Operadores

Em C/C++ temos quatro classes de operadores: aritméticos, relacionais, lógicos e bit a bit. Além disso, C tem alguns operadores especiais para tarefas particulares.

2.7.1 Operadores Aritméticos

Operador	Ação	Precedência
-- ++	Decremento e Incremento	Maior
* /	Multiplicação e Divisão	↓
%	Resto da Divisão Inteira	↓
- +	Subtração e Adição	Menor

Os operadores +, -, * e / trabalham em C/C++ da mesma maneira da maioria das outras linguagens.

Quando / é aplicado a um inteiro, qualquer resto é truncado.

O operador **%** retornará o resto da divisão **inteira**. Porém, **%** não pode ser usado nos tipos em ponto flutuante.

Exemplo:

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

int x,y;

main()
{
 x=5;
 y=2;
 printf ("%d", x/y); // mostrará 2
 printf ("\n%d", x%y); // mostrará 1 (resto da divisão)
 x=1;
 y=2;
 printf ("\n%d  %d", x/y, x%y); // mostrará 0 1
 printf ("\n%f", 1.0/2.0); // mostrará 0.500000
 getch();
}
```

Os operadores de incremento e decremento são muito úteis. Por exemplo: **x = x+1** é o mesmo que **++x** ou **x++**, assim como **x = x-1** é o mesmo que **--x** ou **x--**. Apesar dos operadores de incremento e decremento poderem ser utilizados como prefixo ou sufixo do operando, há uma diferença quando são usados em expressões. Quando um operador de incremento ou decremento **precede** seu operando, C executa a operação de incremento ou decrementos **antes** de usar o valor do operando. Se o operador estiver após seu operador, C usará o valor do operando antes de incrementá-lo ou decrementá-lo. Por exemplo:

```
#include <stdio.h>
#include <conio.h>
int x,y;
int a,b,c,i=3;
main ()
{
 x = 10;
 y = ++x;
 printf("%d %d", y,x); // y recebe 11, x recebe 11
 x = 10;
 y = x++;
 printf("\n%d %d", y,x); // y recebe 10, x recebe 11
 getch();
 clrscr(); // limpa a tela
 a = i++;
 printf("%d %d %d %d", a,b,c,i); // 3 0 0 4
 b = ++i;
 printf("\n%d %d %d %d", a,b,c,i); // 3 5 0 5
 c = --i;
 printf("\n%d %d %d %d", a,b,c,i); // 3 5 4 4
 getch();
}
```

2.7.2 Operadores Aritméticos de Atribuição

Operador	Exemplo	Comentário
+=	x += y	Equivale a x = x + y
-=	x -= y	Equivale a x = x - y
*=	x *= y	Equivale a x = x * y
/=	x /= y	Equivale a x = x / y
%=	x %= y	Equivale a x = x % y

Estes operadores realizam as operações aritméticas e posteriormente a operação de atribuição. Por exemplo: x += y, realiza a operação x + y e depois atribui seu resultado à variável x.

2.7.3 Operadores Relacionais

São operadores que comparam duas grandezas.

Operador	Operação
==	Igual
!=	Diferente
<	Menor
>	Maior
<=	Menor ou igual
>=	Maior ou igual

2.7.4 Operadores Lógicos

Os operadores lógicos e relacionais freqüentemente trabalham juntos.

A idéia de **verdadeiro** e **falso** é a base dos conceitos dos operadores lógicos e relacionais. Em C/C++, **verdadeiro** é qualquer valor **diferente de zero**. Falso é zero. As expressões que usam operadores lógicos ou relacionais devolvem zero para falso e **um para verdadeiro**.

Operador	Operação
&&	Conjunção (E)
||	Disjunção (OU)
!	Negação

Temos a seguinte tabela verdade para os operadores lógicos:

p	q	p && q	p || q	! p	! q
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

Obs: Existem os operadores **&** e **|** que são utilizados para manipular **ponteiros**. Cuidado para não confundir!!

A precedência para os operadores lógicos e relacionais é mostrada na tabela a seguir.

Operador	Precedência
!	Maior
> >= < <=	↓
== !=	↓
&&	↓
||	Menor

2.7.5 Algumas Funções Matemáticas Pré Definidas em C/C++

Estão listadas a seguir algumas funções matemáticas pré-definidas em C/C++. A linguagem C/C++ possui muitas outras que podem ser vistas detalhadamente na documentação da biblioteca **math.h**. Tal biblioteca deve ser incluída no programa que fará uso de alguma dessas funções.

Função	Exemplo	Comentário
ceil	ceil (x)	Arredonda o valor do número real x para cima, por exemplo, ceil(3.2) é 4.
cos	cos (x)	Calcula o cosseno de x (x deve estar representado em radianos).
exp	exp (x)	Obtém o logaritmo natural e elevado à potencia x (e^x).
fabs	fabs (x)	Obtém o valor absoluto de x.
floor	floor (x)	Arredonda um número real para baixo, por exemplo, floor(3.2) é 3.
log	log (x)	Obtém o logaritmo natural de x.
log10	log10 (x)	Obtém o logaritmo de base 10 de x.
modf	modf (x,y)	Decompõe um número real em duas partes: **x** recebe a parte fracionária e **y** recebe a parte inteira do número.
pow	pow (x,y)	Calcula a potência de **x** elevado a **y**.
sin	sin (x)	Calcula o seno de x (x deve estar em radianos).
sqrt	sqrt (x)	Calcula a raiz quadrada de x.
tan	tan (x)	Calcula a tangente de x (x deve estar em radianos).

2.7.6 Prioridades

A linguagem C/C++ permite que sejam combinadas diversas operações em uma expressão, como por exemplo: 10>5 && !(10<9) || 3 <= 4. Neste caso, o resultado é verdadeiro.

Para que as operações possam ser efetuadas sem problemas, devemos seguir certas prioridades. Operadores de mesmo nível de precedência são avaliados da esquerda para a direita. Obviamente parênteses podem ser usados para alterar a ordem de avaliação.

A tabela a seguir mostra a lista de precedência dos operadores mencionados anteriormente. Para uma tabela completa veja Schildt (páginas 39 a 60).

Operador	Precedência
++ --	Maior
* / %	
+ -	
Função pré definida	
< <= > >=	
== !=	
!	
&&	
||	
= += -= *= /= etc	Menor

Exercícios

1. Faça um programa que receba quatro números inteiros, calcule e mostre a soma desses números.

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

int n1,n2,n3,n4,soma;

main()
{
  printf("Digite 4 numeros inteiros \n");
  scanf("%d %d %d %d", &n1,&n2,&n3,&n4);
  soma = n1 + n2 + n3 + n4;

  printf("\nA soma dos numeros eh %d",soma);
  getch();   // para o programa e espera enter
}
```

2. Faça um programa que receba 3 notas e seus respectivos pesos, calcule e mostre a média ponderada dessas notas.

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

float n1,n2,n3,p1,p2,p3,media;

main()
{
 printf("Digite 3 notas \n");
 scanf("%f %f %f", &n1,&n2,&n3);
 printf("Digite o peso das 3 notas, respectivamente \n");
 scanf("%f %f %f", &p1,&p2,&p3);
 media = (n1*p1 + n2*p2 + n3*p3)/(p1+p2+p3);

 printf("\nA media ponderada das notas eh %f",media);
 getch();
}
```

3. Faça um programa que receba o salário de um funcionário, calcule e mostre o novo salário, sabendo-se que este sofreu um aumento de 25%.

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

float salario, NovoSalario;

main()
{
 printf("Digite o salario do funcionario: ");
 scanf("%f", &salario);
 NovoSalario = salario*1.25;

 /* NovoSalario = salario + salario*25/100; ou
   NovoSalario = salario + salario*0.25; */

 printf("\nO novo salario eh = %6.2f",NovoSalario);
 getch();
}
```

4. Faça um programa que calcule a área de um círculo. Sabe-se que Área = πR^2.

```
#include <stdio.h>
#include <conio.h>
#include <math> //se usar a funcao pow

float area, raio;

main()
{
 printf("Digite o raio do circulo: ");
 scanf("%f", &raio);
 area = 3.1415*raio*raio;

/* area = 3.1415*pow(raio,2); */

 printf("\nA area do circuloo eh = %6.2f",area);
 getch();   // para o programa e espera enter
}
```

5. Faça um programa que receba um número positivo e maior que zero, calcule e mostre:
a. o número digitado ao quadrado,
b. o número digitado ao cubo,
c. a raiz quadrada do número digitado,
d. a raiz cúbica do número digitado.

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

float x, x2, x3, raiz2, raiz3;

main()
{
 printf("Digite um numero: ");
 scanf("%f", &x);
 x2 = pow(x,2);
 x3 = pow(x,3);
 raiz2 = sqrt(x);
 raiz3 = exp(log(x) * 1/3);

 printf("\n %4.2f:",x);
 printf("\n\t ao quadrado = %4.2f",x2);
 printf("\n\t ao cubo = %4.2f",x3);
 printf("\n\t raiz quadrada = %4.2f",raiz2);
 printf("\n\t raiz cubica = %4.2f",raiz3);
 getch();
}
```

6. Sabe-se que: 1 pé = 12 polegadas, 1 jarda = 3 pés e 1 milha = 1.760 jardas. Faça um programa que receba uma medida em pés, faça as conversões para polegadas, jardas e milhas. Mostre seus resultados.

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

float pes, polegadas, jardas, milhas;

main()
{
 printf("Digite um numero: ");
 scanf("%f", &pes);
 polegadas = pes * 12;
 jardas = pes / 3;
 milhas = jardas / 1760;

 printf("\n %4.2f pes equivale a:", pes);
 printf("\n\t %10.5f polegadas",polegadas);
 printf("\n\t %10.5f jardas",jardas);
 printf("\n\t %10.5f milhas",milhas);
 getch();
}
```

CAPÍTULO 3: ESTRUTURAS DE CONTROLE

3.1 Estrutura Condicional (Comandos de Seleção)

A estrutura condicional permite a escolha de ações e estruturas a serem executadas quando determinadas condições (expressões lógicas) são ou não satisfeitas.

Esta estrutura pode ser representada como:

- condicional simples (if);
- condicional composta (if-else);
- múltipla escolha (swich);
- operador alternativo (?).

3.1.1 Estrutura Condicional Simples

ALGORITMO:

```
Se condição
   Então
      comando
```

```
Se condição
   Então
      Início
         comando1
         comando2
         ⋮
         comandoN
      Fim
```

C:

```
if condição
   comando;
```

```
if condição
   {
      comando1;
      comando2;
      ⋮
      comandoN;
   }
```

Neste tipo de estrutura o **comando** só será executado se a **condição** for verdadeira (a condição/expressão resulta em não zero). As condições devem estar entre parênteses.

Exemplos:

1) Leia dois números reais. Se o primeiro é maior que o segundo escreva uma mensagem.

Algoritmo
```
Início
    N1, N2: real
    Escreva (' Digite dois numeros: ')
    Leia (N1,N2)
    Se N1 > N2
        Então
            Escreva (' O primeiro numero é maior
            ')
Fim
```

C
```
#include <stdio.h>
#include <conio.h>
float N1,N2;
main ()
{
 printf("Digite 2 numeros: \n");
 scanf("%f %f", &N1,&N2);
 if (N1 > N2)
   printf("\nO 1o numero eh maior que o 2o");
 getch();   // para o programa e espera enter
}
```

2) Leia dois números reais. Se o primeiro é maior, a diferença entre os dois será armazenada numa variável. Escreva essa variável.

```
Início
      X, Y, Z : real;
      Escreva (' Digite dois números: ')
      Leia (X,Y)
      Se ( X > Y )
          Então
                Início
                      Z = X - Y
                      Escreva (' Z = ', Z)
                Fim
Fim
```

```
#include <stdio.h>
#include <conio.h>
float N1,N2,N3;
main ()
{
 printf("Digite 2 numeros: \n");
 scanf("%f %f", &N1,&N2);
 if (N1 > N2)
   {
    N3 = N1-N2;
    printf("\nA diferença entre %5.2f e %5.2f eh:
            %5.2f", N1,N2,N3);
   }
 getch();   // para o programa e espera enter
}
```

3.1.2 Estrutura Condicional Composta

ALGORITMO:

```
Se condição
   Então
      comando1
   Senão
      comando2;
```

C:

```
if condição
    comando1;
else
    comando2;
```

Neste tipo de estrutura o **comando1** será executado se a **condição** for verdadeira; caso contrário será executado o **comando2**.

Se, em alguma situação, existir mais de um comando devemos colocar INÍCIO/FIM no algoritmo, e { } em C/C++. Lembre-se que, em C/C++ **se existir mais de um comando** eles devem ser **separados por ponto e vírgula**;

Exemplo: Faça um programa que leia duas notas e calcule sua média. Se a média for maior que 5.0 escreva a media e a mensagem "Aluno Aprovado". Caso contrário escreva apenas "Aluno Reprovado".

```
Início
  N1, N2, M : real
  Escreva (' Digite as duas notas do aluno: ')
  Leia (N1,N2)
  M = (N1 + N2) / 2
  Se M ≥ 5.0
    Então
      Inicio
        Escreva (' Média = ', M:4:1)
        Escreva (' Aluno Aprovado ')
      Fim
    Senão
        Escreva (' Aluno Reprovado ')
Fim
```

```
#include <conio.h>
float N1,N2,M;
main ()
{
 printf("Digite 2 notas: \n");
 scanf("%f %f", &N1,&N2);
 M = (N1 + N2) / 2;
 if (M >= 5.0)
  {
   printf ("\nMedia = %4.1f", M);
   printf ("\nAluno Aprovado");
  }
 else
  printf ("\nAluno Reprovado");
 getch();
}
```

3.1.3 Estrutura Condicional Encadeada

Dentro de uma estrutura condicional podemos ter várias estruturas condicionais. Em C/C++, um comando else sempre se refere ao comando if mais próximo, que está dentro do mesmo bloco do else e não está associado a outro if.

Exemplos:

1)

```
if (i)
  {
     if (j)
        comando1;
     if (k)
        comando2;
    else  /* este else está associado ao if (k) */
        comando3;
  }
else
  comando4; /* este else está associado ao if(i) */
```

2)

```
if (expr_log)
{
      comando1; // executado se expr_log for VERDADEIRA
      comando11;
      if (expr_log2)
      {
          comando2; // executado se expr_log e expr_log2 forem ambas VERDADEIRAS
          comando21;
      }
      else
      {
          comando3; // executado se expr_log for VERDADEIRA e expr_log2 for FALSA
          comando31;
      }
      comando4; // executado se expr_log for VERDADEIRA
}
else
{
      comando5; // executado se expr_log for FALSA
      comando51;
      if (expr_log3)
      {
          comando6; // executado se expr_log FALSA e expr_log3 for VERDADEIRA
          comando61;
      }
      else
      {
          comando7; // executado se expr_log e expr_log3 forem ambas FALSAS
          comando71;
      }
      comando8; // executado sempre que expr_log for FALSA
}
comando9; // executado sempre
⋮
```

3) Faça um algoritmo que leia 3 valores inteiros, determine e imprima o menor deles.

```
Início
    A, B, C, MENOR : inteiro
    Escreva (' Entre com os valores de A, B e C. ')
    Leia (A,B,C)
    Se ( A < B ) e ( A < C )
        Então
                MENOR = A
        Senão
                Se ( B < C )
                    Então
                            MENOR = B
                    Senão
                            MENOR = C
    Escreva (' Menor valor = ', MENOR)
Fim
```

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

int A,B,C,menor;

main ()
{
 printf("Entre com os valores de A, B e C. \n");
 scanf("%d %d %d", &A,&B, &C);
 if (( A < B ) && ( A < C ))
    menor = A;
 else
   if ( B < C )
      menor = B;
   else
      menor = C;
 printf ("\nMenor valor = %d", menor);
 getch();
}
```

3.1.4 Estrutura de Múltipla Escolha

O comando **switch** é usado para várias seleções. Assim como o **if**, o **switch** divide uma seqüência de possíveis ações em seções de códigos individuais. Para a execução de um determinado comando **switch**, **somente uma** dessas seções será selecionada para execução. A seleção está baseada numa série de testes de comparação, sendo todos executados sobre um valor desejado. Sua estrutura é:

C:

```
switch (variável)
  {
    case valor1:
          lista de comandos;
          break;
    case valor2:
          lista de comandos;
          break;
          ⋮
    case valorN:
          lista de comandos;
          break;
    default:
          lista de comandos;
  }
```

O comando switch (variável) avalia o valor de uma variável para decidir qual case será executado. Esta variável deve ser do tipo **int** ou **char**.

Cada case está associado a um possível valor da variável.

O comando break deve ser utilizado para impedir que sejam executados os comandos definidos nos cases subseqüentes. Caso o comando break não seja encontrado, o código para os comandos case seguintes são executados.

Quando o valor da variável não coincidir com aqueles especificados nos cases, será executado então o default. Tal comando é opcional e, se não estiver presente nenhuma ação será realizada se todos os testes falharem.

O padrão ANSI C especifica que um switch pode ter pelo menos 257 comandos case.

O comando switch pode ser escrito como um conjunto de comandos if.

```
if (val ==1)
   <comando1>
else
   if (val ==2)
      <comando2>
   else
      <comando3>
```

O significado é o mesmo, mas preferencialmente utiliza-se o switch, pois o código fica mais legível. Ou seja, o comando switch difere do comando if porque só pode testar iguladade, enquanto if pode avaliar uma expressão lógica ou relacional.

Exemplo: Construa um algoritmo que, tendo como dados de entrada o preço de um produto e um código de origem, emita o preço junto de sua procedência. Caso o código não seja nenhum dos especificados, o produto deve ser encarado como importado.

Código de origem:
1 – Sul
2 – Norte
3 – Leste
4 – Oeste
5 – Nordeste
6 – Sudeste
7 a 10 – Centro-Oeste

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

int origem;
float preco;

main ()
{
   printf("Forneca o preco e o codigo de origem. \n");
   scanf("%f %d", &preco,&origem);
   clrscr(); // função que limpa a tela
   switch (origem)
     {
       case 1:
            printf("%5.2f - produto do Sul",preco);
            break;
       case 2:
            printf("%5.2f - produto do Norte",preco);
            break;
       case 3:
            printf("%5.2f - produto do Leste",preco);
            break;
       case 4:
            printf("%5.2f - produto do Oeste",preco);
            break;
       case 5:
            printf("%5.2f - produto do Nordeste",preco);
            break;
       case 6:
            printf("%5.2f - produto do Sudeste",preco);
            break;
       default:
            if((7 <= origem) && (origem <=10))
                printf("%5.2f - produto do Centro-Oeste",preco);
            else
                printf("%5.2f - produto importado",preco);
     }
   getch();
}
```

3.1.5 Operador Condicional ?

O operador **?** substitui certas sentenças da forma if-else. Sua forma geral é

Exp1 ? Exp2 : Exp3

A expressão 1 (Exp1) é avaliada; se ela for verdadeira, então Exp2 é avaliada e se torna o valor da expressão. Se Exp1 é falsa, então Exp3 é avaliada e se torna o valor da expressão.

Exemplo:

```
x = 10;
y = x>9 ? 100 : 200;
```

ou

```
x = 10;
if (x>9)
    y = 100;
else
    y = 200;
```

Exercício: Faça um programa que leia um número inteiro e eleva esse número ao quadrado preservando seu sinal (10^2=100, $(-10)^2$=-100). Utilize o operador **?**.

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

int aux, i;

main()
{
 printf("Digite um numero: ");
 scanf("%d", &i);
 aux = i>0 ? i*i : -(i*i);
 printf("%d ao quadrado eh %d", i, aux);
 getch();
}
```

Exercícios

1) Dados 3 números inteiros, apresente esses números em ordem crescente.

```
#include <stdio.h>
#include <conio.h>
int A,B,C,menor;

main ()
{
   printf("Entre com os valores de A, B e C. \n");
   scanf("%d %d %d", &A,&B, &C);

   if ( (A < B)  &&  (A < C) )
      if (B < C)
         printf ("\nOrdem crescente: %d, %d, %d", A,B,C);
      else
         printf ("\nOrdem crescente: %d, %d, %d", A,C,B);
   else
      if ( B<A && B<C )
         if (A < C)
            printf ("\nOrdem crescente: %d, %d, %d", B,A,C);
         else
            printf ("\nOrdem crescente: %d, %d, %d", B,C,A);
      else
         if (A < B)
            printf ("\nOrdem crescente: %d, %d, %d", C,A,B);
         else
            printf ("\nOrdem crescente: %d, %d, %d", C,B,A);
   getch();
}
```

2) Dado um número inteiro, verifique se ele é par ou ímpar.

```
#include <stdio.h>
#include <conio.h>
int num, resto;

main ()
{
 printf("Digite um numero. \n");
 scanf("%d", &num);
 resto = num % 2;
 if ( resto == 0 )
   printf ("\nO numero %d eh par", num);
 else
   printf ("\nO numero %d eh impar", num);
 getch();
}
```

ou

```
#include <stdio.h>
#include <conio.h>
int num;

main ()
{
 printf("Digite um numero. \n");
 scanf("%d", &num);
  if ( num % 2 == 0 )
    printf ("\nO numero %d eh par", num);
 else
    printf ("\nO numero %d eh impar", num);
 getch();
}
```

3) Dado 3 valores, x, y e z, verifique se formam um triângulo (o comprimento de cada lado é menor que a soma dos outros lados) e qual tipo de triângulo.

```
#include <stdio.h>
#include <conio.h>
float x,y,z;
main ()
{
   printf("Digite os lados do triangulo: \n");
   scanf("%f %f %f", &x,&y, &z);
   if (x<y+z && y<x+z && z<x+y)
     if (x==y && x==z)
        printf("%4.2f, %4.2f, %4.2f formam um triangulo equilatero", x, y,z);
     else
        if (x==y || x==z || y==z)
           printf("%4.2f, %4.2f, %4.2f formam um triangulo isosceles", x, y,z);
        else
           printf("%4.2f, %4.2f, %4.2f formam um triangulo escaleno", x, y,z);
   else
     printf("%4.2f, %4.2f, %4.2f nao formam um triangulo", x, y,z);
   getch ();
 }
```

4) Faça um programa que resolva uma equação do 2º grau. Escreva qual o tipo de raiz.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
float a,b,c,x1,x2,delta;

main ()
{
  printf("Digite os coeficientes da equacao \n");
  scanf("%f %f %f", &a,&b,&c);
  clrscr();
  if (a == 0)
     printf("\nA equacao nao eh do 2o grau.");
  else
     {
       delta = b*b - 4*a*c;
       if (delta < 0)
          printf("\nNao existe raiz real.");
       else
             if (delta == 0)
               {
                 x1 = -b / (2*a);
                 printf("\nExiste uma raiz real multipla:");
                 printf("\nX1 = X2 = %4.2f", x1);
               }
             else
               {
                 x1 = (-b + sqrt(delta))/ (2*a);
                 x2 = (-b - sqrt(delta))/ (2*a);
                 printf("\nExistem duas raizes reais distintas:");
                 printf("\nX1 = %4.2f e X2 = %4.2f", x1,x2);
               }
          }
    getch();
  }
```

5) Faça um programa que receba o código correspondente ao cargo de um funcionário e o seu salário atual e, mostre o cargo, o valor do aumento e seu novo salário. Os cargos são dados na tabela a seguir. Utilize switch.

Código	Cargo	Percentual
1	Escriturário	50%
2	Secretário	35%
3	Caixa	20%
4	Gerente	10%
5	Diretor	Não tem aumento

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
float salario, aumento, novo_sal;
int cargo;

main()
{
 clrscr();
 printf("Digite o cargo do funcionario (1,2,3,4 ou 5)");
 scanf("%d",cargo);
 printf("Digite o valor do salario atual: ");
 scanf("%f", &salario);
 switch(cargo)
  {
   case 1:
    printf("\nO cargo eh de Escriturario");
    aumento = salario * 50 / 100;
    printf("\nO valor do aumento eh %4.2f
            ",aumento);
    novo_sal = salario + aumento;
    printf("\nO novo salario eh %4.2f ",novo_sal);
    break;
   case 2:
    printf("\nO cargo eh de Secretario");
    aumento = salario * 35 / 100;
    printf("\nO valor do aumento eh %4.2f
            ",aumento);
    novo_sal = salario + aumento;
    printf("\nO novo salario eh %4.2f ",novo_sal);
    break;
   case 3:
    printf("\nO cargo eh de Caixa");
    aumento = salario * 20 / 100;
    printf("\nO valor do aumento eh %4.2f
            ",aumento);
    novo_sal = salario + aumento;
    printf("\nO novo salario eh %4.2f ",novo_sal);
    break;
   case 4:
    printf("\nO cargo eh de Gerente");
    aumento = salario * 10 / 100;
    printf("\nO valor do aumento eh %4.2f
            ",aumento);
    novo_sal = salario + aumento;
    printf("\nO novo salario eh %4.2f: ",novo_sal);
    break;
   case 5:
    printf("\nO cargo eh de Diretor");
    printf("\nNao tem aumento.");
    break;
   default:
    printf("\nCargo invalido !");
  }
 getch();
}
```

3.2 Estrutura De Repetição

Podemos classificar uma estrutura de repetição como:

- Repetição (laço) contada: quando se sabe previamente quantas vezes o comando composto no interior da construção será executado;
- Repetição condicional: quando não se sabe de antemão o número de vezes que os comandos no interior do laço serão repetidos, pois este número depende de uma condição sujeita à modificação pelas instruções do interior do laço. Este tipo de repetição pode ser com interrupção no início ou no final.

3.2.1 Repetição Contada (For)

for (i = valor_inicial**;** condição**;** incremento ou decremento de i)
comando ou seqüência de comandos**;**

A primeira parte (i = valor_inicial) atribui um valor inicial à variável de controle **i**, que tem por objetivo controlar o número necessário de repetições.

A segunda parte (condição) corresponde a uma expressão relacional que, quando assumir valor **falso**, determinará o fim da repetição.

A terceira parte (incremento ou decremento de i) é responsável por alterar o valor da variável **i** com o objetivo de, em algum momento, fazer com que a condição assuma valor falso.

Deve-se lembrar que, se houver uma seqüência de comandos a ser executada, esses comandos devem estar delimitados por { }.

Exemplo:

```
#include <stdio.h>
#include <conio.h>
int contador =1;

main ()
{
  for (contador=1; contador<=10; contador++)
      printf("%d\n", contador);
  getch();
}
```

3.2.2 Repetição Com Interrupção No Início (while)

É uma estrutura de repetição utilizada quando não sabemos quantas repetições serão necessárias, isto é, a interrupção do laço depende de uma (ou mais) condição. Neste caso, os comandos serão repetidos enquanto a condição for **verdadeira**.

Nesta estrutura, a verificação da condição é feita antes da execução dos comandos. Se logo no início a condição for falsa esta estrutura não será executada.

while (condição)
seqüência de comandos**;**

Lembre-se que pode existir mais de uma condição de controle. Além disso, se houver uma seqüência de comandos a serem executados, tais comandos devem ser delimitados ({}).

Para a execução terminar é necessário que a seqüência de comandos altere a condição de verdadeira para falsa, senão o programa ficar num laço infinito (loop).

```
#include <stdio.h>
#include <conio.h>
int contador =1;

main ()
{
    while (contador <= 10)
        {
            printf("%d\n", contador);
            contador ++;
        }
    getch();
}
```

3.2.3 Repetição Com Interrupção No Final (do-while)

Este comando trabalha da mesma maneira que **while,** exceto pelo fato de que o valor da condição é testado após a execução da seqüência de comandos, ao invés de ser testado antes. Isto implica que pelo menos uma vez esses comandos serão executados. Cada vez que o comando é executado a condição é testada. Se for **verdadeira** o processo é repetido, caso contrário é interrompido;

Este comando é útil quando a seqüência de comandos deve ser executada uma vez antes que a condição seja avaliada.

```
do
    {
        seqüência de comandos;
    }
while (condição);
```

O exemplo a seguir é equivalente aos anteriores.

```
#include <stdio.h>
#include <conio.h>
int contador =1;

main ()
{
    do
        {
            printf("%d\n", contador);
            contador ++;
        }
    while (contador <= 10);
    getch();
}
```

3.3 Interrupções com break e continue

A linguagem em C/C++ oferece ainda duas formas para a interrupção antecipada de um determinado laço.

O comando **break** tem duas utilizações: para terminar um **case** em um comando **switch**, ou para forçar o término imediato de um laço, evitando o teste condicional normal do laço. Quando utilizado dentro de um laço, interrompe e termina a execução do mesmo imediatamente. A execução prossegue com os comandos subseqüentes ao bloco.

```
#include <stdio.h>
#include <conio.h>
int i;
main ()
{
  for (i=0; i<10; i++)
  {
      if (i==5)
          break;
      printf("%d ",i);
  }
  printf("Fim\n");
  getch();
}
```

A saída deste programa será: 0 1 2 3 4 Fim, pois quando **i** tiver o valor **5**, o laço será interrompido e finalizado pelo comando **break**, passando o controle para o próximo comando após o laço, no caso uma chamada final de **printf**.

É importante comentar que da mesma forma que é possível sair de um laço, pode-se sair de um programa usando a função **exit()** da biblioteca padrão **<stdlib.h>**. Essa função provoca o término imediato do programa inteiro, forçando um retorno ao sistema operacional.

O comando **continue** também interrompe a execução dos comandos de um laço. A diferença básica em relação ao comando **break** é que o laço não é automaticamente finalizado. O comando **continue** força que ocorra a próxima iteração do laço, pulando qualquer comando intermediário. Para o laço **for**, **continue** faz com que o teste condicional e a porção de incremento do laço sejam executados. Assim,

```
#include <stdio.h>
#include <conio.h>
int i;
main ()
{
  for (i=0; i<10; i++)
  {
      if (i==5)
          continue;
      printf("%d ",i);
  }
  printf("Fim\n");
  getch();
}
```

gera a saída:

0 1 2 3 4 6 7 8 9 Fim

Devemos ter cuidado com a utilização do comando **continue** nos laços **while**, pois nesses laços ocontrole do programa passa para o teste condicional. O programa

```
/* INCORRETO */
#include <stdio.h>
#include <conio.h>
int i = 0;
main ()
{
  while (i<10)
  {
      if (i==5)
          continue;
      printf("%d ",i);
      i++;
  }
  printf("Fim\n");
  getch();
}
```

é um programa **INCORRETO**, pois o laço criado não tem fim. Isto porque a variável **i** nunca terá valor superior a **5**, o teste será sempre verdadeiro. O que ocorre é que o comando **continue** "pula" os demais comando do laço quando **i** vale **5**, inclusive o comando que incrementa a variável **i**.

Existem outros comandos de desvio, **return** e **goto**, que serão vistos posteriormente.

Exercícios

1) Construir um programa para calcular o fatorial de um número N fornecido pelo usuário.

```
#include<stdio.h>
#include<conio.h>
long int fat=1;
int i,n;

main ()
{
 printf("\nForneca o valor de N:");
 scanf("%d",&n);
 if (n >=0 && n<=12)
   {
     for (i=1; i<=n; i++)
         fat = fat*i;    // fat *= I;
     printf("\nFatorial de %d = %u", n,fat);
   }
 else
   {
     printf("\n\nEstouro no tipo de variavel.");
     printf(" Calculo Incorreto.");
   }
 getch();
}
```

2) Escreva um algoritmo para calcular a soma de dez números quaisquer fornecidos pelo usuário.

```
#include <stdio.h>
#include <conio.h>
int i;
float num, soma;

main ()
{
 soma = 0;
 for (i=1; i<=10; i++)
   {
     printf("\nForneca um numero: ");
     scanf("%f",&num);
     soma = soma + num;
   }
 printf("\nSoma = %4.2f", soma);
 getch();
}
```

3) Dado um número N>0, construir um programa que mostre os N primeiros números ímpares.

A) USANDO while

```
#include <stdio.h>
#include <conio.h>
int i, N, NovoTermo, TermoAnt;

main ()
{
 printf("\nDigite o valor de N: ");
 scanf("%d", &N);
 i = NovoTermo = 1;
 if (N>0)
   while (i<=N)
     {
      printf("%d, ", NovoTermo);
      TermoAnt = NovoTermo;
       NovoTermo = TermoAnt + 2;
      i = i + 1;//i++;
     }
 else
   printf("\nO programa soh determina impares
           maiores que zero");
 getch();
}
```

B) USANDO do-while

```
#include <stdio.h>
#include <conio.h>
int i, N, NovoTermo, TermoAnt;

main ()
{
 printf("\nDigite o valor de N: ");
 scanf("%d", &N);
 i = NovoTermo = 1;
 if (N >0)
  do
    {
     printf("%d, ", NovoTermo);
     TermoAnt = NovoTermo;
     NovoTermo = TermoAnt + 2;
     i = i + 1;//i++;
    }
  while (i<=N);
  else
    printf("\nO programa soh determina impares
            maiores que zero");
 getch();
}
```

4) Construir um programa que calcule o valor de S, usando os 20 primeiros termos.

$$S = \frac{x}{1} + \frac{x}{2} + \frac{x}{3} + \cdots + \frac{x}{20}$$

```
#include <stdio.h>
#include <conio.h>
float x, S;                    // foat x, S=0;
int i;
main ()
{
 printf("Digite o valor de x: \n");
 scanf("%f", &x);
 S = 0;                        // se S=0 na declaração, retirar essa linha
 for (i=1; i<=20; i++)
   S = S + x/i;                // S += x/i;
 printf("Soma = %4.2f",S);
 getch();
```

5) Faça um programa que leia um número indeterminado de linhas contendo cada uma a idade de um indivíduo. A última linha, que não entrará nos cálculos, contém o valor da idade menor ou igual a zero (flag). Calcule e mostre a idade média desse grupo de indivíduos.

```
#include <stdio.h>
#include <conio.h>
int idade, soma;
float contador,media;

main ()
{
   soma = contador = 0;
   printf("Digite a idade: ");
   scanf("%d", &idade);
   while (idade > 0)
    {
        soma += idade;
        contador ++;
        printf("\nDigite a idade: ");
        scanf("%d", &idade);
    }
   media = soma / contador;
   printf("\nA media das idades eh %4.2f", media);
   getch();
}
```

6) Elabore um algoritmo que efetue a soma de todos os números ímpares que se encontram no conjunto dos números de 1 até 500.

```
#include <stdio.h>
#include <conio.h>
int soma, termo, contador;

main ()
{
 soma = 0;
 termo = 1;
 while (termo<=500)
  {
    soma += termo;
    termo += 2;
  }
 printf("\nA soma dos numeros impares de 1 a 500 eh %d", soma);
 getch();
}
```

7) Faça um programa que leia um valor N inteiro e positivo, calcule e mostre a série dada por:

$$S = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots + \frac{1}{N!}$$

```
#include <stdio.h>
#include <conio.h>
float S;
int i, N, den;
main ()
{
 printf("Digite o valor de N: ");
 scanf("%d", &N);
 S = 1;
 den = 1;
 for (i=1; i<=N; i++)
    {
     den *= i;
     S = S + 1.0/den;  // S = S +(float)1/den; ⇒ cast = força a expressão 1/den ser real
    }
 printf("\nSoma = %4.3f",S);
 getch();
}
```

OBS: É possível forçar uma expressão a ser de um determinado tipo usando um **cast**. Sua forma genérica é:

<tipo> <expressão>

No exercício temos

```
S = S +(float)1/den;        // força a expressão 1/den ser do tipo float
```

8) Faça um programa que leia um valor x, calcule e mostre a série abaixo considerando os 15 primeiros termos.

$$S = 1 + \frac{x^2}{1} + \frac{x^3}{2} + \frac{x^4}{3} + \frac{x^5}{4} + \cdots$$

```
#include <stdio.h>
#include <conio.h>
float x, num, S=1;
int i;
main ()
{
 printf("Digite o valor de x: ");
 scanf("%f", &x);
 num = x*x;
 for (i=1; i<=15-1; i++)
    {
     S +=  num/i;
     num *= x;
    }
 printf("\nSoma = %4.2f",S);
 getch();
}
```

9) Faça um programa que leia um valor x, calcule e mostre a série abaixo considerando os 10 primeiros termos.

$$S = \frac{x}{1} - \frac{x}{2} + \frac{x}{3} - \frac{x}{4} + \cdots$$

```
#include <stdio.h>
#include <conio.h>
float x, S=0;
int i;
main ()
{
  printf("Digite o valor de x: ");
  scanf("%f", &x);
  for (i=1; i<=10; i++)
      {
          S +=  x/i;
          x *=-1;
      }
  printf("\nSoma = %4.2f",S);
  getch();
}
```

EXERCÍCIOS EXTRAS

1. Uma pesquisa sobre algumas características físicas da população de uma determinada região coletou os seguintes dados, referentes a cada habitante, para serem analisados:
   - Sexo (masculino, feminino)
   - Cor de cabelos (louros, castanhos, pretos)
   - Cor dos olhos (azuis, verdes, castanhos)
   - Idade em anos

   Para cada habitante, foi digitada uma linha com esses dados e a última linha que não corresponde a ninguém, conterá o valor de idade igual a –1.
   Escreva um programa que determine e escreva:
   a) A maior idade dos habitantes
   b) A porcentagem de indivíduos do sexo feminino cuja idade está entre 18 e 35 anos inclusive e que tenham olhos verdes e cabelos louros.

2. Para se determinar o número de lâmpadas necessárias para cada cômodo de uma residência, existem normas que dão o mínimo de potência de iluminação exigida por metro quadrado (m^2) conforme a utilização deste cômodo. Seja a tabela tomada como exemplo:

Utilização	Classe	Potência
Quarto	1	15
Sala de TV	1	15
Salas	2	18
Cozinha	2	18
Varandas	2	18
Escritório	3	20
Banheiro	3	20

   Supondo que só serão usadas lâmpadas de 60W, fazer um programa que:
   a) Leia um número indeterminado de linhas contendo cada uma:
      - Cômodo de uma residência
      - As duas dimensões do cômodo
      - Classe de iluminação deste cômodo
   b) calcule e escreva para cada cômodo
      - o cômodo
      - potência de iluminação
      - a área do cômodo
      - número de lâmpadas necessárias
   c) Calcule e escreva para toda a resistência
      - Total de lâmpadas
      - Total da potência

3. Fazer um programa que leia e escreva o nome e a altura das moças inscritas em um concurso de beleza. Para cada moça, existe uma linha contendo seu nome e sua altura. A última linha que não corresponde a nenhuma moça conterá a palavra "vazio" no lugar do nome. Calcule e escreva as duas maiores alturas e quantas moças as possuem.

4. Escrever um algoritmo para fazer uma tabela de senA, com A variando de 0 a 1,6 radiano de décimo em décimo de radiano, usando a série abaixo com erro inferior a 0,0001. Imprimir também o número de termos usados.

   $\text{Sen } A = A - A^3/3! + A^5/5! - ....$

5. Supondo que a população de um país A seja de ordem de 90 milhões de habitantes com uma taxa anual de crescimento de 3% e que a população de um país B seja, aproximadamente, de 200 milhões de habitantes com uma taxa anual de crescimento de 1,5%, fazer um programa que calcule e escreva o número de anos necessários para que a população do país A ultrapasse ou iguale a população do país B, mantidas essas taxas de crescimento.

6. Fazer um programa para calcular o número de dias decorridos entre duas datas (não considerar a ocorrência de anos bissextos), sabendo-se que:
   - cada par de datas é lido numa linha, a última linha contém o número do dia negativo;
   - a primeira data na linha é sempre a mais antiga.

   O ano está digitado com quatro dígitos.

7. Número primo é aquele que só é divisível por ele mesmo e pela unidade. Fazer um programa que lido um intervalo de números, escreva os números primos compreendidos nesse intervalo.

CAPÍTULO 4: VETORES

Um vetor é uma variável composta homogênea unidimensional formada por uma seqüência de variáveis, todas do mesmo tipo, com o mesmo identificador (mesmo nome) e alocadas seqüencialmente na memória.

Como as variáveis têm o mesmo nome, o que as distingue é um índice, que referencia sua localização dentro da seqüência.

Os índices utilizados na linguagem C/C++ para identificar as posições de um vetor começam **sempre em 0 (zero)** e vão até o tamanho do vetor menos uma unidade.

A linguagem C/C++ não tem verificação de limites em vetores (o compilador não acusa este tipo de erro), o programador é quem deve prover a verificação dos limites onde for necessário, pois a ultrapassagem deste limite pode resultar nos mais variados erros durante a execução do programa.

4.1 Declaração de Vetor

Em C/C++, os vetores são identificados pela existência de colchetes logo após o nome da variável no momento da declaração. Dentro dos colchetes deve-se colocar o número de posições do vetor.

Exemplos:

1)

```
float nota[6];
```

A variável **nota** contém seis posições, começando pela posição **0** e indo até **5** (6-1).
Como foi declarada sendo do tipo float, em cada posição poderão ser armazenados números reais, por exemplo:

nota

5.0	7.5	6.2	4.0	2.0	3.0
0	1	2	3	4	5

2)

```
char x[5];
```

A variável **x** contém cinco posições (de **0** a **4**). Em cada posição poderão ser armazenados caracteres, conforme especificado pelo tipo char na declaração:

x

A	!	5	@	f
0	1	2	3	4

4.2 Manipulação de Vetor

Atribuição

```
vet [0] = 1;

/* atribui o valor 1 ao primeiro elemento do vetor vet */
```

Entrada de dados:

```
for (i=0; i<10; i++)
   scanf("%f",&A[i]);

/* O usuário entrará com o valor dos 10 elementos do vetor A do tipo float*/
```

Escrevendo os elementos:

```
for (i=0; i<10; i++)
   printf("%5.2f ",A[i]);
```

Exemplo: Dados dois vetores a e c de dimensão 10, calcule o vetor c obtido da soma de a e b.

```
#include<stdio.h>
#include<conio.h>
float a[10],b[10],c[10];
int i;

main ()
{
  printf("\nDigite o vetor a:\n");
  for(i=0;i<4;i++)
      {
        printf("a[%d] = ", i);
        scanf("%f",&a[i]);
      }

  printf("\nDigite o vetor b:\n");
  for(i=0;i<4;i++)
      {
        printf("b[%d] = ", i);
        scanf("%f",&b[i]);
      }

  for(i=0;i<4;i++)
      c[i] = a[i] + b[i];

  printf("\nNovo vetor c = a + b:\n");
  for(i=0;i<4;i++)
      printf("c[%d] = %5.2f\n", i,c[i]);
  getch();
}
```

EXERCÍCIOS

1) Leia uma variável de 100 elementos reais e verifique se existem elementos iguais a 64. Se existirem, escreva as posições em que estão armazenados.

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

float vet[100];
int i;

main ()
{
  printf("\nDigite os elementos do vetor:\n");
  for(i=0;i<5;i++)
    {
      printf("vet[%d] = ", i);
      scanf("%f",&vet[i]);
    }

  printf("\nOs elementos que possuem valor igual a 64 estao nas posicoes: ");
  for(i=0;i<9;i++)
    if (vet[i] == 64)
      printf("%d ", i);
  getch();
}
```

2) Dado um vetor de N números inteiros ($N \geq 20$), calcule e escreva o somatório dos valores deste vetor.

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

int N,i,soma,vet[10];

main ()
{
  printf("\nDigite a dimensao do vetor, sendo no maximo 20: ");
  scanf("%d",&N);

  printf("\nDigite os elementos do vetor:\n");
  for(i=0;i<N;i++)
    {
      printf("vet[%d] = ", i);
      scanf("%d",&vet[i]);
    }

  soma = vet[0];
  for(i=1;i<N;i++)
    soma += vet[i];

  printf("\nA soma dos elementos do vetor eh %d.", soma);
  getch();
}
```

3) Construir um algoritmo que leia um conjunto A de 20 elementos, construa e imprima outro conjunto B da seguinte forma:

- os elementos de índices pares correspondem a A^3
- os elementos de índices ímpares correspondem a A/2

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int N,i,soma;
float a[20], b[20];

main ()
{
  printf("\nDigite os elementos do vetor a:\n");
  for(i=0;i<20;i++)
    {
      printf("a[%d] = ", i);
      scanf("%f",&a[i]);
    }
  for(i=0;i<20;i++)
    if (i % 2 == 0)
      b[i] =  pow(a[i],3);
    else
      b[i] = a[i]/2;
  printf("\nNovo vetor b:\n");
  for(i=0;i<20;i++)
    printf("b[%d] = %6.2f\n", i,b[i]);
  getch();
}
```

4) Construir um algoritmo que, dado um vetor A de 20 elementos, calcule e escreva:

$$S=(A_0-A_{19})^2+(A_1-A_{18})^2+\cdots+(A_9-A_{10})^2$$

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define N 20
int i, j;
float soma, a[N];

main ()
{
  printf("\nDigite os elementos do vetor a:\n");
  for (i=0;i<N;i++)
  {
    printf("a[%d] = ", i);
    scanf("%f",&a[i]);
  }
  soma = 0;
  j = N-1;
  for (i=0;i<N/2;i++)
    {
      soma += pow(a[i]-a[j],2);
      j--;
    }
  printf("\nSoma = %6.2f\n", soma);
  getch();
}
```

5) Leia uma variável de 50 elementos do tipo caracter e verifique se existem elementos iguais a um determinado caracter fornecido pelo usuário. Se existirem escreva as posições em que estão armazenados. **OBS:** Semelhante ao exerc.1, porém os elementos do vetor são do tipo "char".

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

char vet[50];
char caracter;
int i;

main ()
{
  printf("\nDigite os elementos do vetor:\n");
  for(i=0;i<50;i++)
    {
      printf("vet[%d] = ", i);
      gets(&vet[i]);
    }

  printf("\nDigite o caracter a ser procurado: ");
  gets(&caracter);

  printf("\nOs elementos com valor igual a %c
          estao nas posicoes: ", caracter);
  for(i=0;i<50;i++)
    if (vet[i] == caracter)
      printf("%d ", i);
  getch();
}
```

CAPÍTULO 5: MATRIZES

São estruturas de dados homogêneas multidimensionais que necessitam de mais de um índice para a individualização (referência) de seus elementos.

A linguagem C/C++ permite declarar matrizes unidimensionais (vetores), bidimensionais e multidimensionais. O padrão ANSI prevê, no mínimo, 12 dimensões; Entretanto, o limite de dimensões depende da quantidade de recurso disponível ao compilador. As matrizes mais utilizadas são as bidimensionais.

A declaração de uma matriz é feita da forma:

tipo nome_da_variável [dim 1] [dim 2] ... [dim N]

onde:

tipo: tipo a que pertencem todos os elementos da matriz;
nome_da_variável: nome dado à variável do tipo matriz;
[dim 1] [dim 2] ... [dim N]: representam as possíveis dimensões da matriz.

Exemplo:

```
float A [2] [4];
```

A é uma matriz com duas linhas e quatro colunas, cujos elementos são do tipo float. Devemos lembrar que, assim como ocorre com os vetores, os índices começam sempre em zero:

	0	1	2	3
0				
1				

A atribuição de valores a uma matriz é realizada como nos vetores.
Assim, A[1][2] = 6.5 representa:

	0	1	2	3
0				
1			6.5	

Para entrar com os dados de uma matriz e mostrá-los na tela também procedemos de forma análoga aos vetores, porém, como a matriz possui duas dimensões é necessária a utilização de dois contadores (um para a linha e outro para a coluna).

Para a matriz A declarada anteriormente temos:

```
/* Entrada de Dados */

for (i = 0; i<2; i++)
    for (j=0; j<4; j++)
      {
        printf("A[%d][%d] = ", i,j);
        scanf("%f",&A[i][j]);
      }
```

```
/* Visualização dos Dados */

for(i=0;i<2;i++)
    {
      for (j=0; j<4; j++)
          printf("A[%d][%d] = %5.2f  ", i,j,A[i][j]);
      printf("\n");
    }
```

5.1 Iniciação de Matrizes

A linguagem C/C++ permite a iniciação de matrizes (e vetores) no momento da declaração. A forma geral de uma iniciação de matriz é semelhante à de outras variáveis:

tipo nome_da_variável [dim 1] [dim 2] ... [dim N] = { lista_de valores };

A lista de valores é uma lista separada por vírgulas de constantes cujo tipo é compatível com o tipo especificado para a matriz. A primeira constante é colocada na primeira posição da matriz, a segunda na segunda posição, e assim por diante.

Exemplos:

1) Vetor

```
int x[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

Esta declaração representa: x[0] = 1, x[1] = 2, ..., x[9] = 10.

2) Matriz

```
int A[5][2] = {
                1, 5,
                2, 6,
                3, 7,
                4, 8,
                5, 9,
                };
```

ou seja, A[0][0]=1, A[0][1]=5, A[1][0]=2, A[1][1]=6, ..., A[4][0]=5, A[4][1]=9,

EXERCÍCIOS

1) Dadas as matrizes A e B, 5x3, determine a matriz C = A + B. Imprima A, B e C.

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

float A[5][3], B[5][3], C[5][3];
int i,j;

main ()
{
  printf("Digite os elementos da matriz A\n");
  for (i = 0; i<5; i++)
      for (j=0; j<3; j++)
        {
          printf("A[%d][%d] = ", i,j);
          scanf("%f",&A[i][j]);
        }

  printf("\nDigite os elementos da matriz B\n");
  for (i = 0; i<5; i++)
      for (j=0; j<3; j++)
        {
          printf("B[%d][%d] = ", i,j);
          scanf("%f",&B[i][j]);
        }

  for (i = 0; i<5; i++)
      for (j=0; j<3; j++)
        C[i][j] = A[i][j] + B[i][j];

  printf("\nMatriz C=A+B\n");
  for(i=0;i<5;i++)
      {
        for (j=0; j<3; j++)
          printf("C[%d][%d] = %6.2f  ", i,j,C[i][j]);
          printf("\n");
      }

  getch();
}
```

2) Dada uma matriz 3x6, calcule a soma de cada coluna dessa matriz e armazene esses valores num vetor. Imprima a matriz e o vetor.

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

float A[3][6], v[6];
int i,j;

main ()
{
  printf("Digite os elementos da matriz A\n");
  for (i = 0; i<3; i++)
      for (j=0; j<6; j++)
        {
          printf("A[%d][%d] = ", i,j);
          scanf("%f",&A[i][j]);
        }

  for (j = 0; j<6; j++)
      {
        v[j]=0;
          for (i=0; i<3; i++)
            v[j] += A[i][j];
      }

  printf("\nMatriz A\n");
  for(i=0;i<3;i++)
    {
      for (j=0; j<6; j++)
        printf("%6.2f  ", A[i][j]);
      printf("\n");
    }

  printf("\nVetor soma das colunas de A\n");
  for(i=0;i<6;i++)
    printf("v[%d]= %5.2f\n", i,v[i]);

  getch();
}
```

3) Dadas as matrizes A e B, determine a matriz C tal que C = A * B. Imprima A, B e C.

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

#define M 4
#define N 3
#define P 2

float A[M][N], B[N][P], C[M][P];
int i,j,k;

main ()
{
  printf("Digite os elementos da matriz A\n");
  for (i = 0; i<M; i++)
    for (j=0; j<N; j++)
      {
        printf("A[%d][%d]= ", i,j);
        scanf("%f",&A[i][j]);
      }

  printf("\nDigite os elementos da matriz B\n");
  for (i = 0; i<N; i++)
    for (j=0; j<P; j++)
      {
        printf("B[%d][%d]= ", i,j);
        scanf("%f",&B[i][j]);
      }

  for (i = 0; i<M; i++)
    for (j=0; j<P; j++)
      {
        C[i][j] = 0;
        for (k=0; k<N; k++)
          C[i][j] += A[i][k] * B[k][j];
      }

  printf("\nMatriz C=A*B\n");
  for(i=0;i<M;i++)
    {
      for (j=0; j<P; j++)
        printf("%6.2f  ", C[i][j]);
      printf("\n");
    }

  getch();
}
```

4) Dada A_{nxn} de elementos inteiros, calcule e imprima a soma dos elementos situados abaixo da diagonal principal de A, incluindo os elementos da própria diagonal principal.

```
#include <conio.h>
#include <alloc.h>

#define N 10

int A[N][N],i,j,dim,soma=0;

main ()
{

  do
    {
      printf("\nDigite a dimensao da matriz Anxn,
              n<=10: ");
      scanf("%d",&dim);
    }
  while (dim < 0 || dim > 10);

  printf("\nDigite os elementos da matriz A\n");
  for (i=0; i<dim; i++)
    for (j=0; j<dim; j++)
      {
        printf("A[%d][%d]= ", i,j);
        scanf("%d",&A[i][j]);
      }

  /*
  for (i=0; i<dim; i++)
    for (j=0; j<=i; j++)
      soma += A[i][j];
  */

  for (j=0; j<dim; j++)
    for (i=j; i<dim; i++)
      soma += A[i][j];

  printf("\nSoma = %d", soma);

  getch();
}
```

5) Dada uma matriz 10x3 com as notas de 10 alunos em 3 provas, faça um programa que mostre um relatório com o número do aluno (número da linha) e a prova em que cada aluno obteve menor nota. Ao final do relatório, mostre quantos alunos tiveram menor nota na P1, quantos alunos tiveram menor nota na P2 e quantos alunos tiveram menor nota na P3.

```
#include <stdio.h>
#include <conio.h>
float notas[5][3], menor;
int q1, q2, q3, prova_menor, i, j;

main()
{
    for (i=0;i<5;i++)
      for (j=0;j<3;j++)
        {
          printf("\nDigite a nota da P%d do aluno %d: ", j+1,i+1);
          scanf("%f", &notas[i][j]);
        }

    q1 = q2 = q3 = 0;

    for (i=0;i<5;i++)
      {
       printf("\nAluno numero %d: ",i+1);
       menor = notas[i][0];
       prova_menor = 1;
       for (j=1;j<3;j++)
           if (notas[i][j] < menor)
             {
               menor = notas[i][j];
               prova_menor = j+1;
             }
       printf("sua menor nota foi na P%d \n",prova_menor);

       switch (prova_menor)
           {
             case 1:
                   q1++;
                   break;
             case 2:
                   q2++;
                   break;
             case 3:
                   q3++;
                   break;
           }
      }

    printf("\nQuantidade de alunos com menor nota na P1 = %d",q1);
    printf("\nQuantidade de alunos com menor nota na P2 = %d",q2);
    printf("\nQuantidade de alunos com menor nota na P3 = %d",q3);
    getch();
}
```

EXERCÍCIOS EXTRAS DE VETORES E MATRIZES

1) Um elemento A_{ij} de uma matriz é chamado **ponto de sela** da matriz A se, e somente se, A_{ij} for ao mesmo tempo o menor elemento da linha i e o maior elemento da coluna j. Faça um programa que leia uma matriz de ordem 5x7, verifique se a matriz possui ponto de sela e, se possuir, mostre seu valor e localização,

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

 int A[3][4];
 int i, j, maior, menor, lin1, lin2, col1, col2, cont;

void main()
{
 clrscr();
 printf("Digite os elementos da matriz A\n");
 for (i = 0; i<3; i++)
  for (j=0; j<4; j++)
    {
     printf("A[%d][%d] = ", i,j);
     scanf("%d",&A[i][j]);
    }
 cont = 0;
 for (i=0;i<3;i++)
  {
   menor = A[i][0];
   lin1 = i;
   col1 = 0;
   for (j=0;j<4;j++)
     if (A[i][j] < menor)
       {
        menor = A[i][j];
        lin1 = i;
       col1 = j;
       }
   maior = A[0][col1];
   lin2 = 0;
   for (j=0;j<3;j++)
     if (A[j][col1] > maior)
       {
         maior = A[j][col1];
         lin2 = j;
       }
   if (lin1 == lin2)
     {
       printf("\nPonto de sela = %d na posicao %d - %d", maior,lin1,col1);
       cont++;
     }
  }
 if (cont == 0)
   printf("\nNao existe ponto de sela nesta matriz ");
 getch();
}
```

2) Faça um programa que leia um vetor V de 12 elementos. Distribua esses elementos em uma matriz 3x4 e mostre essa matriz gerada

5	14	8	9	14	6	58	46	78	91	10	20

5	14	8	9
14	6	58	46
78	91	10	20

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

int V[12], mat[3][4], i, j, lin, col;

void main()
{
 clrscr();
 printf("Digite os elementos do vetor V\n");
 for (i = 0; i<12; i++)
   {
    printf("V[%d]= ", i);
   scanf("%d",&V[i]);
   }
 lin = 0;
 col = 0;
 for (i=0;i<12;i++)
   {
    mat[lin][col] = V[i];
    col++;
    if (col > 3)
      {
       col = 0;
       lin++;
      }
   }
 printf("\n\nMatriz Gerada\n");
 for (i=0;i<3;i++)
   {
   for (j=0;j<4;j++)
     printf(" %d  ", mat[i][j]);
   printf("\n");
   }

 getch();
}
```

3) Faça um programa que receba as vendas de cinco produtos em três lojas diferentes e em dois meses consecutivos. Armazene essas vendas em duas matrizes 5x3. O bimestre é uma matriz 5x3, resultado da soma das duas matrizes anteriores. Calcule e mostre:

- As vendas de cada produto em cada loja no bimestre;
- A maior venda do bimestre;
- O total vendido por loja no bimestre;
- O total vendido de cada produto no bimestre.

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

int mes1[5][3], mes2[5][3], bim[5][3], i, j, tot_prod, tot_loja, maior;

void main()
{
 clrscr();
 printf("\nMes 1\n\n");
 for (i = 0; i<5; i++)
   {
    for (j=0; j<3; j++)
     {
      printf("Digite a venda do produto %d na loja %d referente ao mes 1: ", i+1,j+1);
      scanf("%d",&mes1[i][j]);
     }
    printf("\n");
   }

 clrscr();
 printf("\n\nMes 2\n\n");
 for (i = 0; i<5; i++)
  {
   for (j=0; j<3; j++)
     {
      printf("Digite a venda do produto %d na loja %d referente ao mes 2: ", i+1,j+1);
      scanf("%d",&mes2[i][j]);
     }
   printf("\n");
  }

 clrscr();
 printf("\n\nVenda bimestral do produto:\n\n");
 for (i=0;i<5;i++)
  {
   for (j=0;j<3;j++)
     {
      bim[i][j] = mes1[i][j] + mes2[i][j];
      printf(" %d na loja %d = %d \n", i+1,j+1,bim[i][j]);
      if ((i==0) && (j==0))
        maior = bim[i][j];
      else
        if ((bim[i][j] > maior))
           maior = bim[i][j];
     }
    printf("\n");
  }
```

```
printf("\n\nA maior venda do bimestre foi %d \n",maior);

printf("\n\nTotal vendido no bimestre pela loja\n\n");
for (i=0;i<3;i++)
  {
  tot_loja = 0;
  for (j=0;j<5;j++)
    tot_loja = tot_loja + bim[j][i];
  printf(" %d = %d " ,i+1,tot_loja);
  printf("\n");
  }

printf("\n\nTotal vendido do produto:\n");
for (i=0;i<5;i++)
  {
  tot_prod = 0;
  for (j=0;j<3;j++)
    tot_prod = tot_prod + bim[i][j];
 printf("\n %d = %d ",i+1,tot_prod);
  }
getch();
}
```

CAPÍTULO 6: MANIPULAÇÃO DE CADEIAS DE CARACTERES (Strings)

Como já mencionado, a linguagem C/C++ não possui um tipo de dados específico para cadeia de caracteres como ocorre em outras linguagens.

Em C/C++, para armazenar uma cadeia de caracteres (string) utiliza-se vetores de **char**, onde cada posição representa um caracter.

A declaração geral para uma string é:

char *nome_da_string [tamanho];*

Deve-se destacar que os compiladores C/C++ identificam o fim de uma string por meio do caracter nulo ('\0'). Assim, deve-se declarar o vetor sempre com uma posição a mais para o armazenamento deste caracter, que é feito automaticamente pelo compilador no momento da leitura (scanf). Porém, se a string for montada caracter por caracter é necessário acrescentar "manualmente" o caracter **'\0'** no final. Se isto não for feito, o conjunto de caracteres não será visto como uma string pelas funções de manipulação do mesmo.

Exemplo:

```
char palavra[7];
```

Índice	...	0	1	2	3	4	5	6	...
Valor	...	C	A	D	E	I	A	\0	...

Vale destacar que o armazenamento da string, assim como dos vetores e matrizes, é feito de forma contígua na memória, ou seja, os valores da string ocupam espaços de memória adjacentes, sendo que cada caracter ocupa 1 byte.

6.1 Manipulação de Strings

A biblioteca **string.h** possui funções específicas para a manipulação de strings. As funções listadas a seguir são apenas algumas delas.

- **strcpy** - copia a *str_origem* para a *str_destino*. Sua forma geral é:

  strcpy (str_destino, str_origem);

Exemplo 1:

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
char str1[100], str2[100], str3[100];

main ()
{
   printf ("Entre com uma string: ");
   gets (str1);
   strcpy (str2, str1); /* Copia str1 em str2 */
   strcpy (str3, "Voce digitou a string ");
   /* Copia "Voce digitou a string" em str3 */

   printf ("\n\n%s%s", str3, str2);
   getch();
}
```

- **strcat** – concatena *str_origem* ao final de *str_destino* sem alterar *str_origem*. A *str_destino* deve ser grande o suficiente. Sua forma geral é:

strcat (str_destino, str_origem);

Exemplo 2:

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

char str1[100], str2[100];

main ()
{
   printf ("Entre com uma string: ");
   gets (str1);

   strcpy (str2, "Voce digitou a string ");
   strcat (str2, str1);
   /* str2 armazenara' Voce digitou a string + o conteudo de str1 */

   printf ("\n\n%s", str2);
   getch();
}
```

- **strlen** - retorna o tamanho de uma cadeia de caracteres *str* (o caracter nulo '\0' não é considerado).Isto quer dizer que, de fato, o comprimento da string deve ser um a mais que o inteiro retornado por strlen().Sua forma geral é:

strlen (str);

Exemplo 3:

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

int size;
char str[100];

main ()
{
    printf ("Entre com uma string: ");
    gets (str);

    size = strlen (str);

    printf ("\n\n A string que voce digitou tem tamanho %d", size);
    getch();
}
```

- **strcmp** - compara a *str1* com a *str2* e devolve um valor inteiro:

  < 0 se str1 < str2
  == 0 se str1 = str2
  > 0 se str1 > str2

  Lembre-se que, em uma operação lógica qualquer valor diferente de zero é considerado verdadeiro e assim, esta função retorna **falso** se as strings **são iguais**. Portanto, use o operador **!** para reverter a condição se estiver testando igualdade.
  Esta função considera letras maiúsculas como sendo símbolos diferentes de letras minúsculas. Sua forma geral é:

  strcmp (str1, str2);

Exemplo 4:

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

char str1[20], str2[20];

main ()
{
  printf ("Entre com uma string: ");
  gets (str1);
  printf ("\nEntre com outra string: ");
  gets (str2);

  if (strcmp(str1,str2))
     printf ("\n\n As strings são diferentes.");
  else
     printf ("\n\n As strings são iguais.");

  getch();
}
```

- **stricmp** – análoga a **strcmp**. A diferença entre as duas funções é que esta considera letras maiúsculas ou minúsculas como sendo símbolos iguais. Sua forma geral é:

  stricmp (str1, str2);

Exemplo 5:

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

char str1[100], str2[100];

main ()
{
  printf ("Entre com uma string: ");
  gets (str1);
  printf ("\n Entre com outra string: ");
  gets (str2);

  if (stricmp(str1,str2))
     printf ("\n\n As strings são diferentes.");
  else
     printf ("\n\n As strings são iguais.");

  getch();
}
```

- **strchr** – retorna um ponteiro para a primeira ocorrência de *ch* em *str*. Sua forma geral é:

  strchr (str, ch);

- **strstr** – retorna um ponteiro para a primeira ocorrência de *str2* em *str1*. Sua forma geral é:

  strstr (str1, str2);

6.2 Iniciação de strings

A iniciação de strings pode ser feita de três maneiras:

a) No momento da declaração

```
char str[] = {´A´,´B´,´C´,´D´,´E´,´\0´};
char str[] = “ABCDE”;
char str_vect [3][10] = { "Joao", "Maria", "Jose" };
```

No primeiro caso **str** recebe as letras separadamente, inclusive o caracter nulo (identifica um caracter isoladamente).

No segundo caso **str** foi iniciada com uma palavra, recebendo automaticamente o caracter nulo (identifica uma cadeia de caracter).

No terceiro caso temos um vetor de três elementos cujos valores são strings de comprimento dez.

Nos dois primeiros casos, a quantidade de caracteres é definida automaticamente em função da iniciação. Isto ocorre na hora da compilação e não poderá mais ser mudado durante o programa, sendo muito útil, por exemplo, quando vamos iniciar uma string e não queremos contar quantos caracteres serão necessários.

É incorreto em “C”:

```
char str[16];
main()
{
  str= “ANA”;
}
```

b) Por atribuição (após declaração)

```
char str1[10], str2[5];

strcpy (str1, “Programa”);
ou
strcpy (str1, str2);
```

No primeiro caso **str1** recebe um valor constante (a palavra Programa). No segundo caso, o conteúdo de **str2** é copiado em **str1**.

c) Por meio do teclado

Neste caso utiliza-se a função gets da biblioteca <stdio.h>. Esta função armazena todos os símbolos digitados até a ocorrência do <ENTER> e acrescenta \0 automaticamente.

Sua forma geral é:

gets (nome_da_string);

A função scanf também pode ser utilizada, porém ela considere válidos os caracteres localizandos antes do “espaço”.

Exemplo 6:

```
#include<stdio.h>
#include<conio.h>
char nome[10];
main()
{
    printf("Digite um nome:");
    gets (nome);   //scanf("%s",nome);
    printf("Ola %s",nome);
    getch();
}
```

Exemplo geral:

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
char s1[20], s2[20];

main ()
{
  printf ("Entre com uma string: ");
  gets (s1);
  printf ("\nEntre com outra string: ");
  gets (s2);

  printf ("\n\nComprimentos: %d, %d", strlen(s1), strlen(s2));

  if (!strcmp(s1, s2))
      printf ("\nAs strings sao iguais.");

  strcat (s1, s2);
  printf ("\n%s", s1);

  strcpy (s1, "\nIsto eh um teste.");
  printf (s1);

  if (strchr ("alo", 'o'))
      printf ("\no estah em alo.");

  if (strstr ("ola aqui", "ola"))
      printf ("\nola encontrado");

  getch();
}
```

Se as strings digitadas forem “alo” e “alo”, a saída será:

```
Comprimentos: 3, 3
As strings são iguais.
aloalo
Isto eh um teste.
o estah em alo.
ola encontrado
```

Exercícios

1. Faça um programa que receba uma frase, calcule e mostre a quantidade de palavras da frase digitada.

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

char frase[100];
int qtde, tam, i;

main ()
{
  qtde = 0;
  printf ("Digite uma frase.\n");
  gets (frase);
  tam = strlen (frase); // tamanho da frase
  for (i=0; i<=tam; i++)  // percorre cada caracter da frase
    if (frase[i] == ' ')  // compara cada caracter com espaço em branco
      qtde++;
  qtde++;; // depois da última palavra não tem espaço
  printf ("\nQuantidade de palavras da frase = %d", qtde);
  getch();
}
```

2. Faça um programa que receba uma frase e troque as vogais desta frase por *.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

char frase[50], letra;
int tam, i;

main ()
{
  clrscr ();
  printf ("Digite uma frase\n");
  gets (frase);
  tam = strlen (frase);
  for (i=0;i<tam;i++)
    {
      letra = toupper (frase[i]); // toupper biblioteca ctype.h
      if ((letra=='A') || (letra=='E') || (letra=='I') || (letra=='O') || (letra=='U'))
        frase[i] = '*';
    }
  printf ("\n%s\n", frase);  // puts(frase);
  getch ();
}
```

3. Faça um programa que receba uma frase. Caso a frase possua meses por extenso, substitua-os pelo seu número correspondente.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
char frase[100], frase_nova[100], palavra[15];
int tam, i, cont;
main ()
{
  printf ("Digite uma frase.\n\n");
  gets (frase);
  tam = strlen (frase);
  i = cont = 0;
  for (i=0;i<=tam;i++)
     {
      if ((frase[i] != ' ') && (frase[i] != '\0'))
        {
         palavra[cont] = frase[i];
         cont++;
        }
      else
        {
         palavra[cont] = '\0';
         cont = 0;
         if (stricmp(palavra,"JANEIRO")==0)
           {
            strcat(frase_nova,"1");
            strcat(frase_nova," ");
           }
         else
           {
            if (stricmp(palavra,"FEVEREIRO")==0)
              {
               strcat(frase_nova,"2");
               strcat(frase_nova," ");
              }
            else
              {
               if (stricmp(palavra,"MARCO")==0)
                 {
                  strcat(frase_nova,"3");
                  strcat(frase_nova," ");
                 }
               else
                 {
                  if (stricmp(palavra,"ABRIL")==0)
                    {
                     strcat(frase_nova,"4");
                     strcat(frase_nova," ");
                    }
                  else
                    {
                     if (stricmp(palavra,"MAIO")==0)
                       {
                        strcat(frase_nova,"5");
```

```
        strcat(frase_nova," ");
      }
    else
      {
       if (stricmp(palavra,"JUNHO")==0)
         {
          strcat(frase_nova,"6");
          strcat(frase_nova," ");
         }
       else
         {
          if (stricmp(palavra,"JULHO")==0)
            {
             strcat(frase_nova,"7");
             strcat(frase_nova," ");
            }
          else
            {
             if (stricmp(palavra,"AGOSTO")==0)
               {
                strcat(frase_nova,"8");
                strcat(frase_nova," ");
               }
             else
               {
                if (stricmp(palavra,"SETEMBRO")==0)
                  {
                   strcat(frase_nova,"9");
                   strcat(frase_nova," ");
                  }
                else
                  {
                   if (stricmp(palavra,"OUTUBRO")==0)
                     {
                      strcat(frase_nova,"10");
                      strcat(frase_nova," ");
                     }
                   else
                     {
                      if (stricmp(palavra,"NOVEMBRO")==0)
                        {
                         strcat(frase_nova,"11");
                         strcat(frase_nova," ");
                        }
                      else
                        {
                         if (stricmp(palavra,"DEZEMBRO")==0)
                           {
                            strcat(frase_nova,"12");
                            strcat(frase_nova," ");
                           }
                         else
                           {
                            strcat(frase_nova,palavra);
                            strcat(frase_nova," ");
                           }
                        }
```

```
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
  } //fim for
 strcat (frase_nova,"\0");
 printf ("\n\n%s",frase_nova);
 getch ();
}
```

4. Faça um programa que receba uma frase e mostre as letras que se repetem, juntamente com o número de repetições.

```
#include <conio.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>
char frase[80], letras[26], letra;
int rep[26], tam, i, j;

main ()
{
  clrscr();
  printf("Digite uma frase.\n\n");
  gets(frase);

  for (i=0;i<26;i++)
      rep[i] = 0;
  tam = strlen(frase);

  for (i=0; i<=tam; i++)
      if (frase[i] != ' ')
        {
          j = 0;
          letra = toupper(frase[i]);
          while ((letras[j] != letra) && (letras[j] != '\0'))
              j++;
          if (letras[j] == letra)
            rep[j] = rep[j]+1;
          else
            {
              rep[j] = 1;
              letras[j] = letra;
              letras[j+1] = '\0';
            }
        }

  printf("\n\nLetras repetidas\n");
  for (i=0; i<=strlen(letras); i++)
    if (rep[i] > 1)
      printf("\n%c: apareceu %d
                vezes.\n",letras[i],rep[i]);
  getch();
}
```

5. Faça um programa que receba uma frase e uma palavra. Caso a frase contenha a palavra ESCOLA, substitua-a pela palavra digitada.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
char frase[50], palavra[10], pal_frase[50], frase_nova[50]={""};
int tam, i, cont, res;

main ()
{
  printf("Digite uma frase.\n");
  gets(frase);
  printf("Digite uma palavra.\n");
  gets(palavra);
  tam = strlen(frase);

  cont=0;
  for (i=0;i<=tam;i++)
      {
       if ((frase[i] != ' ') && (frase[i] != '\0') && (frase[i] != ',') && (frase[i] != '.'))
         {
          pal_frase[cont] = frase[i];
          cont++;
         }
       else
         {
          pal_frase[cont] = '\0';
          cont=0;
          if (!stricmp(pal_frase,"ESCOLA"))
            {
             strcat(frase_nova,palavra);
             strcat(frase_nova," ");
             strcpy(pal_frase,"\0");
            }
          else
            {
             strcat(frase_nova,pal_frase);
             strcat(frase_nova," ");
             strcpy(pal_frase,"\0");
            }
         }
      }
  strcat(frase_nova,"\0");
  puts(frase_nova);
  getch();
}
```

CAPÍTULO 7: MODULARIZAÇÃO: FUNÇÕES

Um importante recurso das linguagens de programação é a modularização, na qual um programa pode ser particionado em sub-torinas específicas. A linguagem C/C++ possibilita a modularização por meio de **funções**.

Um programa em linguagem C/C++ tem, no mínimo, uma função principal - a função main() - por onde a execução começa. Em C/C++ não existe o conceito de procedimento, como existia no Pascal. Os procedimentos em C/C++ são também funções, com a particularidade de não retornarem nada.

7.1 Definição

A sintaxe geral para a definição de uma função é:

```
tipo_de_retorno nome_da_função (parâmetro)
{
  variáveis_locais
  instruções
}
```

onde:

- tipo_de_retorno: tipo da informação que a função vai retornar (default é **int**);
- parâmetros (ou argumentos – args): lista de parâmetros com o formato:
  tipo nome1, tipo nome2, ..., tipo nomeN
- Os parênteses ao lado do nome da função são obrigatórios mesmo que a lista de parâmetros esteja vazia

Por exemplo, uma função para calcular o valor médio de dois números, poderia ser definida da seguinte forma:

```
float Media (float a, float b)
{
  float m;
  m = (a + b) / 2;
  return m;
}
```

Esta função poderia depois ser chamada na função main() como se mostra no exemplo seguinte:

```
main ()
{
  float a=5, b=15, result;
  result = Media(a, b);
  printf("Média = %f\n", result);
}
```

Neste exemplo, foi calculada a média entre dois valores do tipo real (**a** e **b**) que foram passados à função **Media** como parâmetros. O resultado é armazenado na variável **m** e será devolvido ao ponto em que a função foi chamada na função principal **main**, para ser utilizado conforme a necessidade do programa. Isso só é possível porque a função **Media** está preparada para tratar esses valores, conforme especificado na primeira linha da função (cabeçalho).

Repare na instrução de **return** na função, que além de terminá-la também é responsável pela definição do valor de retorno da mesma. Pode haver mais de um comando **return** em uma função. Sua forma é:

```
return valor_retornado;
ou
return;
```

Exemplo:

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

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

int main ()
{
 int num;
 scanf ("%d", &num);
 printf ("%d", quadrado (num));
 getch ();
}
```

Exemplo:

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

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

main ()
{
 int num, quad;
 scanf ("%d", &num);
 quad = quadrado (num);
 printf ("%d", quad);
 getch ();
 return (0);
}
```

7.2 Protótipo de Função

Toda função deve ser declarada antes de ser utilizada. Na definição da função está implícita a sua declaração, mas a linguagem C permite que se declare uma função antes de defini-la.

Isto é feito através do protótipo da função, que nada mais é do que o trecho de código que especifica o nome e os parâmetros da função (cabeçalho seguido de **;**).

A forma geral de uma definição de protótipos de função é:

```
tipo_retorno nome_da_função ( tipo param1, tipo param2, ..., tipo paramN);
```

O uso dos nomes dos parâmetros é opcional, porém eles habilitam o compilador a identificar qualquer incompatibilidade de tipos por meio do nome quando ocorre um erro.

Os protótipos permitem que a linguagem C forneça uma verificação mais forte de tipos, pois quando são utilizados o C pode encontrar e apresentar quaisquer conversões de tipos ilegais entre o argumento usado para chamar uma função e a definição de seus parâmetros. A linguagem C também encontra diferença entre o número de argumentos usados para chamar a função e o número de parâmetros da função.

Em C os protótipos são opcionais, porém são obrigatórios em C++.

Exemplo:

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

int quadrado (int x); // protótipo da função

int main ()
{
 int num;
 scanf ("%d", &num);
 num = quadrado (num);
 printf ("%d", num);
 getch ();
 return (0);
}

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

7.3 Funções do Tipo void

Um dos usos de void (vazio) é declarar explicitamente funções que não devolvem valores. Isso evita seu uso em expressões e ajuda a afastar um mau uso acidental.

Antes de poder usar qualquer função void, você deve declarar seu protótipo. Se isso não for feito, C assumirá que ela devolve um inteiro e, quando o compilador encontrar de fato a função, ele declarará um erro de incompatibilidade.

Antes que o padrão C ANSI definisse void, funções que não devolviam valores simplesmente eram assumidas como do tipo int por padrão. Portanto, é comum ver muitos erros deste tipo em códigos mais antigos.

Resumidamente temos:

- Protótipo de uma função que não retorna nada:
  void nome_função (parâmetros);
  Nesse caso return não é necessário.

- Uma função sem parâmetros:
  tipo_retorno nome_função (void);

- Uma função sem parâmetros e sem retorno:
  void nome_função (void);

Exemplo:

```
#include <stdio.h>

void msg (void);

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

```
void msg (void)
{
   printf("Boa noite!");
}
```

De acordo com o padrão ANSI, a função main() devolve um inteiro para o processo chamador, que é geralmente o sistema operacional. Devolver um valor em main() é equivalente a chamar exit() com o mesmo valor. Se main() não devolve explicitamente um valor, o valor vhamado para o processo chamador é tecnicamente indefinido. Na prática, a maioria dos compiladores C devolve 0 (zero), mas não conte com isso se há interesse em portabilidade.

Você também pode declarar main() como void se ela não devolve em valor. Alguns compiladores geram uma mensagem de advertência, se a função não é declarada como void e também não devolve um valor.

Lembre-se que na chamada de funções sem parâmetros (argumentos) é sempre obrigatório utilizar parênteses, sem nada lá dentro, como se vê acima.

Exemplo:

```
main ( )
{
   ⋮
   return 0; // return (0);
}
```

ou

```
void main (void)
{
   ⋮
}
```

Exemplo:

```
#include <stdio.h>

void Multiplicacao(void)
{
   int k;
   for (k=1; k<=10; k++)
       printf("%d\n", k*k);
}
```

```
void main(void)
{
   Multiplicacao();
}
```

Exemplo:

```
//erro: nenhum valor é retornado
int f1 ()
{
}


//correto
void f2 ()
{
}


//correto
int f3 ()
{
 return 1;
}
```

```
//erro: valor é retornado em uma função void
void f4 ()
{
 return 1;
}


//erro:falta valor de retorno
int f5 ()
{
 return;
}


//correto
void f6 ()
{
 return;
}
```

7.3 Escopo de Variáveis

Escopos são regras que determinam o uso e a validade de variáveis nas diversas partes do programa.

Como já vimos, a declaração de variáveis pode ser feita em três lugares básicos: dentro de funções (variáveis locais), na definição dos parâmetros das funções (parâmetros formais) e fora de todas as funções (variáveis globais).

Em C, todas as funções estão no mesmo nível de escopo. Isto é, não é possível definir uma função internamente a uma função. Esta é arazão de C não ser tecnicamente uma linguagem estruturada em blocos.

Variáveis Locais

São as variáveis declaradas dentro de uma função e só têm validade dentro do bloco no qual foi declarada.

As variáveis locais existem apenas enquanto o bloco de código em que foram declaradas está sendo executado. Ou seja, uma variável local é criada na entrada de seu bloco e destruída na saída.

Geralmente as variáveis usadas por uma função são declaradas imediatamente após o abre-chaves da função e antes de qualquer outro comando. Porém as variáveis locais podem ser declaradas dentro de qualquer bloco de código. O bloco definido por uma função é simplesmente um caso especial.

Exemplo:

```
int func1 (...)
{
  int a,x;
      ⋮
}

float func2 (...)
{
  float y;
      ⋮
}
```

```
void main ()
{
  int a,x,y;
      ⋮
  for ...
  {
    float a,b,c;
      ⋮
  }
}
```

Exemplo:

/*Esta função está errada. */

```
void f (void)
{
  int i;
  i = 10;
  int  j;  // esta linha provocará o
  erro
  j = 20;
}
```

/*Esta função está correta.*/

```
void f (void)
{
  int i;
  i = 10;
  {
    /* define j em seu próprio bloco
       de código */
  int j;
  j = 20;
  }
}
```

/* Esta função está correta.
Define j no início do bloco da função. */

```
void f (void)
{
  int i;
  int j;
  i = 10;
  j = 20;
}
```

Parâmetros Formais

Se uma função usa argumentos, ela deve declarar variáveis que receberão os valores dos argumentos. Essas variáveis são denominadas **parâmetros formais** da função. Elas se comportam como qualquer outra variável local dentro da função e suas declarações ocorrem dentro dos parênteses.

Uma vez declarada as variáveis como parâmetros formais, elas podem ser usadas dentro da função como variáveis locais normais, e assim, elas também sãodestruídas na saída da função.

Os parâmetros formais devem ser declarados do mesmo tipo dos argumentos utilizados para chamar a função, caso contrário resultados inesperados podem ocorrer.

Embora a linguagem C forneça os *protótipos de funções*, que podem ser usados para ajudar a verificar se os argumentos usados para chamar a função são compatíveis com os parâmetros, ainda podem ocorrer problemas (os protótipos de função não eliminam inteiramente incongruências de tipo de parâmetros). Além disso, os protótipos de funções devem ser incluídos explicitamente no programa para receber esse benefício extra.

Variáveis Globais

Essas variáveis são reconhecidas pelo programa inteiro e podem ser usadas por qualquer pedaço de código. Além disso, elas guardam seus valores durante toda a execução do programa e são declaradas fora de qualquer função. Elas podem ser acessadas por qualquer expressão independentemente de qual bloco de código contém a expressão.

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

int cont; // variável global

void func1 (void);
void func2 (void);


void main (void)
{
  cont = 100;
  func1();
}
```

```
void func1 (void);
{
  int x; // variável local
  x = cont;
  func2();
  printf("cont = %d, x);
  // imprimirá 100 após os 9 pontos de putchar
}

void func2 (void);
{
  int cont; // variável local
  for (cont = 1; cont < 10; cont++)
      putchar (' . ');
      // imprimirá nove pontos: ……..
}
```

Observe que, apesar de nem **main()** nem **func1()** terem declarado a variável **cont**, ambas podem usá-la. A **func2()**, porém, declarou uma variável local chamada **cont**. Quando **func2()** referencia **cont**, ela referencia apenas sua variável local, não a variável global.

Se uma variável global e uma variável local possuem o mesmo nome, todas as referências ao nome da variável dentro do bloco onde a variável local foi declarada dizem respeito à variável local e não têm efeito algum sobre a variável global.

As variáveis globais são úteis quando o mesmo dado é usado em muitas funções no programa. No entanto, deve-se evitar usar variáveis globais desnecessárias. Elas ocupam memória durante todo o tempo em que o programa está sendo executado, não apenas quando são necessárias. Além disso, as variáveis globais tornam o programa mais difícil de ser entendido e a função menos geral (a função depende de alguma coisa que deve ser definida fora dela). Finalmente, usar um grande número de variáveis globais pode levar a erros no programa devido à mudança acidental do valor de uma variável (em consequencia de ela ter sido usada em algum outro lugar do programa).

7.4 Passagem de Parâmetros por Valor

Numa passagem de parâmetros por valor serão geradas cópias dos valores de cada um dos parâmetros e a função atua sobre essa cópia sem poder modificar a variável original que foi passada.

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

int soma(int a, int b);

void main()
{
   int x, y, res;
   printf("\nDigite dois numeros: \n");
   scanf("%d %d", &x, &y);      // 2 3
   res = soma (x,y);
   printf("\nA soma do dobro dos numeros %d e
           %d eh %d.", x,y,res );
   // A soma do dobro dos números 2 e 3 eh 10.
   getch();
}
```

```
int soma(int a, int b)
{
   int s;
   a = 2*a;
   b = 2*b;
   s = a + b;
   return s;
}
```

Exercícios:

1) Faça uma função que retorne 1 se o número digitado for positivo ou 0 se o número for negativo.

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

int verifica(int num);

void main()
{
  int num, x;
  clrscr();
  printf("\nDigite um numero: ");
  scanf("%d", &num);
  x = verifica(num);  // retirar essa linha
  if (x==1)    //  if (verifica(num)==1)
     printf("\nNumero positivo\n");
  else
     printf("\nNumero negativo\n");
  getch();
}
```

```
int verifica(int num)
{
 int res;
 if (num >= 0)
   res = 1;
 else
   res = 0;
 return res;
}
```

2) Faça uma função que receba dois números positivos por parâmetro e retorne a soma dos N números inteiros existentes entre eles.

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

int somar(int num1, int num2);

void main()
{
 int num1, num2, s;
 clrscr();
 printf("\nDigite dois numeros inteiros: \n");
 scanf("%d %d", &num1,&num2);
 s = somar (num1, num2);
 printf("\nSoma = %d",s);
 getch();
}
```

```
int somar(int num1, int num2)
{
 int i, s;
 s = 0;
 for (i=num1+1;i<=num2-1;i++)
   s = s + i;
 return s;
}
```

3) Faça uma função que receba três números inteiros: *a, b* e *c*, onde *a* é maior que 1. A função deve somar todos os inteiros entre *b* e *c* que sejam divisíveis por *a* (inclusive *b* e *c*) e retorne o resultado para a função principal.

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

int divisores(int a, int b, int c);

void main()
{
 int a, b, c, result;
 clrscr();
 do
   {
    printf("\nDigite o valor de a: ");
    scanf("%d",&a);
   }
 while (a<=1);
 printf("\nDigite o valor de b: ");
 scanf("%d",&b);
 printf("\nDigite o valor de c: ");
 scanf("%d",&c);
 result = divisores (a, b, c);
 printf("\nSoma dos inteiros entre %d e %d ", b,c);
 printf("que sao divisiveis por %d = %d",a,result);
 getch();
}
```

```
int divisores(int a, int b, int c)
{
 int i, s, r;
 s = 0;
 for (i=b;i<=c;i++)
   {
    r = (i % a);
    if (r == 0)
      s = s + i;
   }
 return s;
}
```

4) Faça um procediemnto que receba as três notas de um aluno como parâmetros e uma letra. Se a letra for A o procedimento calcula a média aritmética das notas do aluno, se for P calcula a média ponderada com pesos 5, 3 e 2 respectivamente. A média calcuada deve ser devolvida ao programa principal para ser mostrada.

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

float calcula_media(float n1, float n2, float n3, char l);

void main()
{
 float n1, n2, n3, m;
 char letra;
 clrscr();
 printf("\nDigite as tres notas n1, n2 e n3 respectivamente: \n");
 scanf("%f %f %f", &n1,&n2, &n3);
 do
   {
    printf("\nDigite uma letra (A: Media Aritmetica, P: Media Ponderada) ");
    fflush(stdin);       //stdin:associado ao teclado para entrada de dados
    scanf("%c",&letra);
    letra = toupper(letra);
   }
 while ((letra != 'A') && (letra != 'P'));

 m = calcula_media(n1, n2, n3, letra);
 if (letra=='A')
   printf("\nMedia Aritmetica = %5.2f", m);
 else
   printf("\nMedia Ponderada = %5.2f",m);
 getch();
}

float calcula_media(float n1, float n2, float n3, char l)
{
 float media;
 if (l == 'A')
   media = (n1+n2+n3)/3;
 else
   media = (n1*5+n2*3+n3*2)/(5+3+2);
 return media;
}
```

7.5 Passagem de Parâmetros por Referência

Neste método os parâmetros passados para uma função correspondem a endereços de memória ocupados pelas variáveis utilizadas na chamada da função. Esta referência ao endereço de uma variável é feita declarando os parâmetros formais como ponteiros e, conseqüentemente as alterações feitas no parâmetro afetam a variável usada para chamar a função.

Observação:

1. operador *: faz acesso ao conteúdo de uma área de memória indicada por um ponteiro (parâmetros formais e utilização);
2. operador **&**: retorna o endereço de uma variável (parâmetros reais).

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

void troca (int *a, int *b);

void main (void)
{
  int i, j;
  i = 10;
  j = 20;
  troca (&i, &j); // passa os endereços de i e j
  printf("%d %d", i, j);
  getch();
}
```

```
void troca (int *a, int *b)
{
  int temp;
  temp = *a; // conteúdo de a
  *a = *b;    // conteúdo de b
  *b = temp;
}
```

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

int soma (int *a, int *b);

void main ()
{
   int x, y, res;
   printf("\nDigite dois numeros: \n");
   scanf("%d %d", &x, &y);          // 2 3
   res = soma (&x, &y);
   printf("\nSoma dos numeros = %d", res );
       // Soma dos números = 10
   printf("\n\n x=%d,  y=%d", x,y);
       // x=4, y=6
   getch();
   }
```

```
int soma (int *a, int *b)
{
   int s;
   *a = 2*(*a);
   *b = 2*(*b);
   s = *a + *b;
   return s;
}
```

7.6 Passagem de Vetores e Matrizes como Parâmetros

Quando um vetor ou matriz é usado como parâmetros para uma função, apenas o seu endereço é passado como parâmetro (passagem por **referência**). Assim, os elementos do vetor ou da matriz que forem modificados na função mantêm essas modificações, mesmo depois da função terminar.

Em C um nome de matriz sem qualquer índice é um ponteiro para o primeiro elemento na matriz. Quando uma função com um nome de matriz é chamada, um ponteiro para o seu primeiro elemento é passado para a função. Isto significa que a declaração de parâmetros deve ser de um tipo de ponteiro compatível.

Existem três maneiras de declarar um parâmetro que receberá um ponteiro para uma matriz:

1) Especificando o valor do vetor ou matriz que será passado:

```
void levet(int vet[10])
{
}
```

O compilador C converte vet para um ponteiro de inteiros, pois nenhum parâmetro pode receber um vetor inteiro.

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

void escreve (int num[10]);

void main ()
{
 int i, t[10];
 for (i=0; i<10; i++)
   t[i] = i;
 escreve (t);
 getch();
}

void escreve (int num[10])
{
 int i;
 for (i=0; i<10; i++)
   printf("%d ", num[i]);      // 0 1 2 3 4 5 6 7 8 9
}
```

2) Especificar como um vetor/matriz sem dimensão

```
void levet(int vet[])
{
}
```

A variável vet é um vetor de inteiros de tamanho desconhecido. No caso de uma matriz multidimensional é necessário indicar todas as dimensões com exceção da primeira.

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

void escreve (int num[]);

void main ()
{
 int i, t[10];
 for (i=0; i<10; i++)
   t[i] = i;
 escreve (t);
 getch();
}

void escreve (int num[])
  /* num é uma matriz de inteiros de tamanho desconhecido. C
     não fornece nenhuma verificação de limites em matrizes */
{
   int i;
   for (i=0; i<10; i++)
     printf("%d ", num[i]);      // 0 1 2 3 4 5 6 7 8 9
}
```

3) Como um ponterio. Esta é a forma mais comum.

```
void levet(int *vet)
{
}
```

Esta estrutura é permitida porque qualquer ponteiro pode ser indexado usando [ ], como se fosse uma matriz.

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

void escreve (int *num);

void main ()
{
   int i, t[10];
   for (i=0; i<10; i++)
     t[i] = i;
   escreve (t);
   getch();
}
```

```
void escreve (int *num)
  /* Permitido porque qualquer ponteiro pode er
     indexado usando [ ], como se fosse uma matriz */
{
 int i;
 for (i=0; i<10; i++)
   printf("%d ", num[i]);

 // 0 1 2 3 4 5 6 7 8 9
}
```

A passagem por referência de vetor/matriz é facilitada criando-se um novo tipo de dados. Isso é feito atavés do comando **typedef**. Em seguida basta utilizar um ponteiro para fazer a passagem por referência. A sintaxe do comando **typedef** é dada por:

typedef tipo novo_nome;

Exemplos:

1)

```
typedef int inteiro;        //define o tipo inteiro a partir de int
inteiro x;                  // x é uma variável do tipo int
```

2)

```
typedef int vetor[20];      //define o tipo vetor com 20 elementos do tipo int
vetor X, Y;                 // X e Y são variáveis do tipo vetor
```

3)

```
typedef int matriz[10][5];
matriz A, B;
```

Exemplo: Faça um programa que trabalhe com matrizes inteiras MxN, contendo rotinas para:

- ler uma matriz inteira PxQ, usando passagem de parâmetro por referência;
- escrever uma matriz inteira PxQ;
- somar todos os elementos de uma matriz PxQ.

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

#define M 10
#define N 10

typedef int matriz[M][N];

void Le_Matriz(int linha, int coluna, matriz A);
void Escreve_Matriz(int linha, int coluna, matriz A);
int soma_toda_matriz(int linha, int coluna, matriz A);

void main()
{
 matriz A;
 int linhas, colunas, soma;

 clrscr();
 printf("\nForneca o numero de linhas (<=10): ");
 scanf("%d", &linhas);
 printf("\nForneca o numero de colunas (<=10): ");
 scanf("%d", &colunas);

 printf("\n\nForneca os elementos da matriz\n\n");
 Le_Matriz (linhas, colunas, A);

 printf("\n\nEscreve a matriz\n\n");
 Escreve_Matriz (linhas, colunas, A);

 soma = soma_toda_matriz(linhas,colunas,A);
 printf("\n\nSoma de todos os elementos = %d ", soma);

 getch();
}
```

```
void Le_Matriz(int linha, int coluna, matriz A)
{
 int i, j;
 for (i = 0; i<linha; i++)
   {
    for (j=0; j<coluna; j++)
      {
       printf("Matriz[%d][%d] = ", i,j);
       scanf("%d",&A[i][j]);
      }
    printf("\n");
   }
}

void Escreve_Matriz(int linha, int coluna, matriz A)
{
 int i, j;
 for(i=0; i<linha; i++)
   {
    for (j=0; j<coluna; j++)
      printf("Matriz[%d][%d] = %d     ", i,j,A[i][j]);
    printf("\n");
   }
}

int soma_toda_matriz(int linha, int coluna, matriz A)
{
 int i, j, soma;
 soma=0;
 for (i=0;i<linha;i++)
   for (j=0;j<coluna;j++)
     soma += A[i][j];
 return soma;
}
```

Exercícios

1) Faça duas rotinas (funções): uma para entrar com os dados do tipo inteiro de um vetor e outra para escrever este vetor. Esta função tem como parâmetro a dimensão N ($N \leq 50$) do vetor.

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

void Le_Vetor (int n, int v[50]);
void Escreve_Vetor (int n, int v[50]);

void main()
{
 int n, x[50];

 do
   {
    printf("\nDigite a dimensao do vetor, n<=50:
");
    scanf("%d", &n);
   }
 while ( n<=0 || n>=50);
 Le_Vetor (n,x);
 Escreve_Vetor (n,x);
 getch();
}
```

```
void Le_Vetor(int n, int v[50])
{
 int i;
 printf("\n\nForneca os elementos do vetor\n");
 for (i=0; i<n; i++)
  {
   printf("v[%d] = ", i);
   scanf("%d",&v[i]);
  }
}


void Escreve_Vetor(int n, int v[50])
{
 int i;
 printf("\nVetor\n");
 for(i=0;i<n;i++)
   printf("[%d] = %d\n", i,v[i]);
}
```

2) Faça um programa que dados dois vetores calcule um terceiro a partir da soma desses dois vetores. O programa deve conter uma rotina para entrada de dados de um vetor do tipo inteiro, outra rotina para escrever este vetor e uma terceira rotina que efetua a soma destes dois vetores. A dimensão dos vetores é fornecida no programa principal. *(Ex_2_Geral.cpp)*

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

void Le_Vetor(int n, int v[50]);
void Escreve_Vetor(int n, int v[]);
void Soma_Vetores(int n, int *a, int *b, int *c);

int main()
{
 int n, x1[50], x2[50], x3[50];
 do
   {
   printf("\nDigite a dimensao do vetor, N<=50: ");
   scanf("%d", &n);
   }
 while ( n<=0 || n>=50);
 printf("\n\nVetor x1\n");
 Le_Vetor (n,x1);
 printf("\n\nVetor x2\n");
 Le_Vetor (n,x2);
 Soma_Vetores (n,x1,x2,x3);
 printf("\n\nVetor x3 = x1 + x2\n");
 Escreve_Vetor (n,x3);
 getch();
}
```

```
void Le_Vetor(int n, int v[50])
{
 int i;
 for (i=0; i<n; i++)
  {
   printf("v[%d] = ", i);
   scanf("%d",&v[i]);
  }
}

void Escreve_Vetor(int n, int v[50])
{
 int i;
 for(i=0;i<n;i++)
   printf("v[%d] = %d\n", i,v[i]);
}

void Soma_Vetores (int n, int *a, int *b, int*c)
{
 int i;
 for(i=0;i<n;i++)
   c[i] = a[i] + b[i];
}
```

3) Faça uma função que receba, por parâmetro, um vetor A de 10 elementos inteiros positivos. Ao final desa função, o vetor B deve conter o fatorial de cada elemento de A. O vetor B deve ser mostrado no programa principal.

A	2	1	0	3	4	...
B	2	1	1	6	24	...

```
#include <stdio.h>
#include <conio.h>
void Le_Vetor(int n, int *v);
void Escreve_Vetor(int n, int *v);
void fatorial(int n, int *v1, int *v2);

int main()
{
 int n, a[10], b[10];
 n=10;
 printf("\nVetor a\n");
 Le_Vetor (n,a);
 fatorial (n, a, b);
 printf("\n\nVetor b\n");
 Escreve_Vetor (n,b);
 getch();
}

void Le_Vetor(int n, int *v)
{
 int i;
 for (i=0; i<n; i++)
     {
    printf("[%d] = ", i);
    scanf("%d",&v[i]);
    }
}
```

```
void Escreve_Vetor(int n, int *v)
{
 int i;
 for(i=0;i<n;i++)
  printf("[%d] = %d\n", i,v[i]);
}


void fatorial(int n, int *a, int *b)
{
 int i, j;
 for (i=0;i<n;i++)
  if ((a[i] == 0) || (a[i] == 1))
    b[i] = 1;
  else
   {
     b[i]=1;
     for (j=1;j<=a[i];j++)
       b[i] = b[i] * j;
   }
}
```

4) Faça um programa completo contendo uma função que recebe, por parâmetro, uma matriz A_{5x5} e retorne a soma de seus elementos.

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

void Le_Matriz(float A[][5]);
float soma_matriz(float A[][5]);


int main()
{
 float A[5][5], resultado;
 printf("\nMatriz A\n\n");
 Le_Matriz (A);
 resultado = soma_matriz(A);
 printf("\nSoma=%6.2f\n", resultado);
 getch();
}
```

```
void Le_Matriz(float A[][5])
{
 int i, j;
 for (i = 0; i<5; i++)
   {
    for (j=0; j<5; j++)
      {
       printf("n[%d][%d] = ", i,j);
       scanf("%f",&A[i][j]);
      }
    printf("\n");
   }
}
```

```
float soma_matriz(float A[][5])
{
 int i, j;
 float soma;
 soma=0;
 for (i=0;i<5;i++)
   for (j=0;j<5;j++)
     soma = soma + A[i][j];
 return soma;
}
```

5) Faça uma rotina que receba um vetor de N elementos inteiros e devolva um outro vetor onde os valores negativos do primeiro vetor foram substituídos por zero.

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

#define M 10

typedef int vetor[M];

void Le_Vetor (int linha, vetor x);
void Escreve_Vetor (int linha, vetor x);
void zerar (int n, vetor A, vetor B);

void main ()
{
 vetor A, B;
 int dim;

 clrscr ();
 printf ("\nForneca a dimensao do vetor (<=10): ");
 scanf ("%d",&dim);
 printf ("\n\nForneca os elementos do vetor\n\n");

 Le_Vetor (dim,A);
 zerar (dim,A,B);

 printf ("\n\nEscreve o vetor A\n\n");
 Escreve_Vetor (dim,A);

 printf ("\n\nEscreve o vetor B\n\n");
 Escreve_Vetor (dim,B);

 getch ();
}
```

```
void Le_Vetor (int linha, vetor x)
{
 int i;
 for (i=0; i<linha; i++)
  {
   printf ("vetor[%d] = ", i);
   scanf ("%d",&x[i]);
  }
}

void Escreve_Vetor (int linha, vetor x)
{
 int i;
 for (i=0; i<linha; i++)
   printf ("vetor[%d] = %d\n", i,x[i]);
}

void zerar (int n, vetor A, vetor B)
{
 int i;
 for (i=0; i<n; i++)
   {
    B[i] = A[i];
    if (A[i]<0)
     B[i] = 0;
   }
}
```

6) Faça um programa que contém uma rotina que recebe por parâmetro, um vetor A de 30 elementos inteiros e retorne dois vetores B e C. O vetor B deve conter os elementos pares de A e C os elementos ímpares. Escreva os três vetores.

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

#define M 10
typedef int vetor[M];

void Le_Vetor (int linha, vetor x);
void Escreve_Vetor (int linha, vetor x);
void separa (int nA, vetor A, int *dimB, vetor B,int
                *dimC, vetor C);

void main()
{
 vetor A, B, C;
 int dim, dimB, dimC;

 clrscr();
 do
  {
  printf ("\nForneca a dimensao do vetor (<=30): ");
  scanf ("%d",&dim);
  }
 while (dim<0 || dim >30);

 printf ("\n\nForneca os elementos do vetor\n\n");
 Le_Vetor (dim,A);

 separa (dim,A,&dimB,B,&dimC,C );

 printf ("\n\nEscreve o vetor A\n\n");
 Escreve_Vetor(dim,A);

 printf ("\n\nEscreve o vetor B\n\n");
 Escreve_Vetor (dimB,B);

 printf ("\n\nEscreve o vetor C\n\n");
 Escreve_Vetor (dimC,C);

 getch ();
}
```

```
void Le_Vetor (int linha, vetor x)
{
 int i;

 for (i=0; i<linha; i++)
  {
  printf ("vetor[%d] = ", i);
  scanf ("%d",&x[i]);
  }
}

void Escreve_Vetor (int linha, vetor x)
{
 int i;

 for (i=0; i<linha; i++)
  printf ("vetor[%d] = %d\n", i,x[i]);
}

void separa (int dimA, vetor A, int *dimB, vetor B,
                int *dimC, vetor C)
{
 int i;

 *dimB = *dimC = 0;
 for (i=0; i<dimA; i++)
  if (A[i]%2 == 0)
    {
     B[*dimB]= A[i];
     (*dimB)++;
    }
  else
    {
     C[*dimC]= A[i];
     (*dimC)++;
    }
}
```

Bubble Sort

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

#define falso 0
#define verdadeiro 1
#define max 15

typedef int vetor[max];

int dimensao ();
void Le_Vetor( int n, vetor v);
void Bubble (int n, vetor v);
void Escreve_Vetor (int n, vetor v);

void main()
{
   vetor x;
   int dim;

   dim = dimensao();
   Le_Vetor (dim,x);
   Bubble (dim,x);
   printf("\nVetor Ordenado\n");
   Escreve_Vetor(dim,x);
   getch();
}

int dimensao ()
    /* controle para que o número de elementos do vetor a
       ser digitado não ultrapasse o máximo declarado */
{
   int n;
   do
     {
      printf("Digite a dimensao do vetor (max= 15):
                ");
      scanf("%d",&n);
     }
   while(n<1 || n>max);
   return n;
}
```

```
void Le_Vetor(int n, vetor v)
{
    int i;
    printf("\n\nForneca os elementos do vetor\n");
    for (i=0; i<n; i++)
      {
       printf("[%d] = ", i);
       scanf("%d",&v[i]);
      }
}

void Escreve_Vetor(int n, vetor v)
{
 int i;
 for(i=0;i<n;i++)
   printf("[%d] = %d\n", i,v[i]);
}

void Bubble (int n, vetor v)
{
   int controle, i, aux, troca;
    /* ordenação crescente dos elementos do vetor*/
    controle = n;
    do
      {
       troca = falso;
       for(i=0; i<(controle-1); i++)
        if ( v[i] > v[i+1] )
         {
          aux = v[i];
          v[i] = v[i+1];
         v[i+1] = aux;
          troca = verdadeiro;
         }
       controle--;
      }
    while ( troca == verdadeiro );
}
```

Teoria Complementar de Strings

Observação: Parte deste material foi cedido pela Profa. Dra. Simone das Graças Domingues Prado.

A linguagem C utiliza o conceito de *streams* (canais) tanto para realizar E/S (Entrada/Saída) como para acessar arquivos. Os canais pré-definidos são:

- **stdin:** associado ao teclado para entrada de dados;
- **stdout:** associado à tela para exibição de dados;
- **stderr:** mensagens de erro, enviadas à tela por default.

1) E/S de Caracteres

```
int getchar (void);      // stdio.h
int getche (void);       // conio.h
int getch (void);        // conio.h
int putchar (int c);     // stdio.h
```

As três primeiras funções lêem caracteres do teclado e a última escreve um caracter na tela.

Exemplo 1: Uso de putchar( ) para escrita de caracteres.

A)

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

void main ()
{
 int ch;
  printf("\nCaracteres da Tabela ASCII de 65 a 90. \n");
 for (ch=65; ch<=90; ch++)
  {
     putchar(ch);
     putchar(' ');
  }
 getche();
}
```

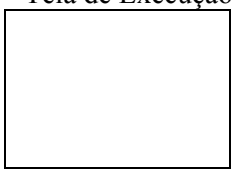
Tela de execução:

```
Caracteres da Tabela ASCII de 65 a 90.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

B)

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

/* define some box-drawing characters */

#define LEFT_TOP  0xDA
#define RIGHT_TOP 0xBF
#define HORIZ     0xC4
#define VERT      0xB3
#define LEFT_BOT  0xC0
#define RIGHT_BOT 0xD9

void main(void)
{
  char i, j;

  /* draw the top of the box */
     putchar(LEFT_TOP);
     for (i=0; i<10; i++)
       putchar(HORIZ);
     putchar(RIGHT_TOP);
     putchar('\n');


     /* draw the middle */
     for (i=0; i<4; i++)
     {
       putchar(VERT);
       for (j=0; j<10; j++)
         putchar(' ');
       putchar(VERT);
       putchar('\n');
     }

     /* draw the bottom */
     putchar(LEFT_BOT);
     for (i=0; i<10; i++)
       putchar(HORIZ);
     putchar(RIGHT_BOT);
     putchar('\n');
     getch();
}
```

Tela de Execução:



Exemplo 2: Uso de getchar( ) e putchar( ) para leitura e escrita de caracteres.

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

void main ()
{

 char ch;

 printf("\nDigite um caracter.");
 printf("\nDigite ponto (.) para parar.
\n\n");

 do
  {
   ch = getchar();
   if ( islower(ch))
      ch = toupper(ch);
   else
      ch = tolower(ch);
   putchar(ch);
   printf("\n");
  }
 while (ch != '.');
}
```

O **exemplo 2** lê caracteres até que seja digitado o caracter ponto (‘.’). O problema é que a função getchar() espera que você digite ENTER para que ela comece a pegar os caracteres. Assim, se você escrever uma cadeia de caracteres, ela será armazenada no buffer e só depois será lida pelo getchar( ). Isso gera um resultado estranho.

Para corrigir esse problema usa-se as outras funções getche( ) ou getch( ). A função getche( ) pega o caracter assim que é pressionado e o mostra na tela, já o getch( ) não mostra o caracter que foi pressionado. Teste o **exemplo 3** para ver como fica a entrada de dados usando essas funções.

Exemplo 3:

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

void main ()
{

 char ch;

 printf("\nDigite um caracter.");
 printf("\nDigite ponto (.) para parar.
\n\n");

 do
  {
   ch = getche();          // ou ch = getch();
   if ( islower(ch))
      ch = toupper(ch);
   else
      ch = tolower(ch);
   putchar(ch);
   printf("\n");
  }
 while (ch != '.');
}
```

Como já visto, a função scanf( ) faz a leitura de qualquer tipo de dado e a função printf( ) faz a escrita. Tanto scanf( ) como getchar( ) guardam a entrada de dados num buffer. A diferença é que getchat( ) não lê o retorno de carro (‘\n’) e o scanf( ) lê e guarda-o na variável. Veja o **exemplo 4**.

Exemplo 4:

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

void main ()
{
 char ch1, ch2, ch3;
 clrscr();
 do
 {
  printf("\nDigite um caracter: ");
  scanf("%c %c %c", &ch1, &ch2, &ch3);
  if ( ch1==' ')
    ch1 = '*';  // espaço em branco
  if ( ch2==' ')
    ch2 = '*';  // espaço em branco
  if ( ch3==' ')
    ch3 = '*';  // espaço em branco
  if ( ch1=='\n')
    ch1 = '@';  // retorno de carro, final de linha
  if ( ch2=='\n')
    ch2 = '@';  // retorno de carro, final de linha
  if ( ch3=='\n')
    ch3 = '@';  // retorno de carro, final de linha

  printf("\nImpressao das variaveis: ch1=%c,
            ch2=%c, ch3=%c\n", ch1,ch2,ch3);
 }
 while (ch1!='.' && ch2!='.' && ch3!='.');
 getche();
}
```

A seguir é mostrada a execução do programa do **exemplo 4** com várias entradas. Observe o que é guardado em cada variável em vários momentos.

```
Digite um caracter: a b c

Impressao das variaveis: ch1=a, ch2=b, ch3=c

Digite um caracter: defg

Impressao das variaveis: ch1=@, ch2=d, ch3=e

Digite um caracter: h i j

Impressao das variaveis: ch1=f, ch2=g, ch3=h

Digite um caracter:
Impressao das variaveis: ch1=*, ch2=i, ch3=j

Digite um caracter: k.mm

Impressao das variaveis: ch1=@, ch2=k, ch3=.
```

Perceba que o caracter branco (‘ ‘) e o de final de linha (‘\n’) em algumas situações foi armazenado e em outras descartado.

Para evitar que o scanf( ) ou o getchar( ) pegue o que sobrou no buffer, pode-se usar a função:

int fflush(File *stream);

Veja agora o mesmo **exemplo 4** com a adição do comando *fflush(stdin);* no início do comando do-while.

```
Digite um caracter: a b c

Impressao das variaveis: ch1=a, ch2=b, ch3=c

Digite um caracter: defg

Impressao das variaveis: ch1=d, ch2=e, ch3=f

Digite um caracter: h i j

Impressao das variaveis: ch1=h, ch2=i, ch3=j

Digite um caracter: k.mm

Impressao das variaveis: ch1=k, ch2=., ch3=m.
```

2) E/S de Strings

char *gets (char *s);
int puts (const char *sc);

A função gets( ) lê uma string a partir do teclado. A leitura ocorre até ser digitado ENTER. No final da string é colocado o caracter nulo (‘\0’).

A função puts( ) escreve uma string na tela. Se a string contiver caracteres de tabulação (‘\t’) ou de retorno de carro (‘\n’), esses são executados, ou seja, acontece uma tabulação e volta a linha, respectivamente. Veja o **exemplo 5.**

Exemplo 5:

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

void main ()
{
 char str1[30];

 clrscr();
 printf("\nDigite uma string: ");
 gets(str1);
 printf("\n\n");
 puts(str1);
 printf("\n\n");
 strcat(str1,"\ttabulacao\nretorno de carro");
 puts(str1);
 getche();
    }
```

Tela de execução:

```
Digite uma string: Testando ...

Testando ...

Testando ...    tabulacao
retorno de carro
```

3) Matrizes de Strings (Schildt, H. – C Completo e Total)

Para criar uma matriz de strings, use uma matriz bidimensional de caracteres. O tamanho do índice esquerdo indica o número de strings e o tamanho do índice do lado direito especifica o comprimento máximo de cada string. O código seguinte declara uma matriz de 30 strings, cada qual com um comprimento máximo de 79 caracteres:

```
char str_array[30][80];
```

Para acessar uma string individualmente, basta especificar apenas o índice esquerdo. Por exemplo, o comando a seguir chama **gets( )** com a terceira string em **str_array**.

```
gets (str_array[2]);
```

Exemplo 6: Faça um programa que leia:

- Um vetor de 8 elementos com os nomes das lojas;
- Um outro vetor de 4 elementos com os nomes dos produtos;
- Uma matriz com os preços de todos os produtos em cada loja (4x8).

Mostre todas as relações (nome do produto – nome da loja) nas quais o preço não ultrapasse R$120,00.

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

char lojas[8][10];      // matriz de string: 8 elementos de 9 caracteres
char produtos[4][10];
float pre[4][8];
int i, j;

void main()
{
 clrscr();
 for (i=0;i<8;i++)
   {
    printf("\nDigite o nome da loja %d \n",i+1);
    gets(lojas[i]);
   }
 for (i=0;i<4;i++)
   {
    printf("\nDigite o nome do produto %d: ",i+1);
    gets(produtos[i]);
   }
 for (i=0;i<4;i++)
   for (j=0;j<8;j++)
     {
       printf("\nDigite o preco do produto %s na loja %s: ", produtos[i], lojas[j]);
      scanf("%f", &pre[i][j]);
     }
 printf("\n\nProdutos e respectivas lojas cujos precos nao ultrapassam R$ 120,00\n");
 for (i=0;i<4;i++)
   for (j=0;j<8;j++)
     if (pre[i][j] < 120)
       printf( "\n %s - %s",produtos[i],lojas[j]);
 getch();
}
```

Exemplos Extras

As teclas **<enter>**, **<esc>**, setas, e funções (**F1** a **F12**) possuem uma leitura diferente. As teclas **<enter>** e **<esc>** ocupam somente um byte quando são lidas. Os códigos ASCII correspondentes são respectivamente 13 e 27. As setas e as funções são lidas e armazenadas em dois bytes, no primeiro byte fica armazenado o valor 0 e no outro o código que identifica a tecla. Assim,

Tecla	Código	Caracter
←	75	K
↑	72	H
→	77	M
↓	80	P
F1	59	;
F2	60	<
F3	61	=
F4	62	>
F5	63	?
F6	64	@
F7	65	A
F8	66	B
F9	67	C
F10	68	D
F11	69	E
F12	70	F

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

void main ()
{

int tecla, tecla2;

 clrscr();
 printf("Digite uma tecla especial: (F ou
setas.\n");
 printf("Tecle ENTER (13) para encerrar\n\n");

 do
 {
  tecla = tecla2 = 0;
  tecla = getch();
  if ( tecla == 0)
    tecla2 = getch();

  switch (tecla2)
   {
     case 75:
       printf("Seta Esquerda");
       break;
     case 72:
       printf("Seta para Cima");
         break;
     case 77:
       printf("Seta Direita");
       break;
     case 80:
       printf("Seta para Baixo");
       break;
     case 59:
       printf("F1");
       break;
     case 60:
       printf("F2");
       break;
     case 61:
       printf("F3");
       break;
     case 62:
       printf("F4");
       break;
     case 63:
       printf("F5");
       break;
     case 64:
       printf("F6");
       break;
     case 65:
       printf("F7");
       break;
     case 66:
       printf("F8");
       break;
     case 67:
       printf("F9");
       break;
     case 68:
       printf("F10");
       break;
     case 133:
       printf("F11");
       break;
     case 134:
       printf("F12");
       break;
     default:
       if (tecla == 27)
        printf("Esc");
    }

  printf("\n\n");
  }

  while (tecla != 13);

 }
```

Faça um programa que receba um caracter e escreva o número correspondente a esse caracter na tabela ASCII.

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

int b1, b2;

main ()
{
  clrscr();
  printf("Codigo ASCII");
  printf("\nPressione uma tecla para obter seu codigo\n");
  printf("Tecle ENTER (13) para encerrar\n\n");

do
   {
     b1 = b2 = 0;
     b1 = getch();
     if (b1 == 0)
       b2 = getch();
     printf("\n\nCaracter = %c",  b1);
     printf("\nCodigo na tabela ASCII:");
     printf(" Byte 1 = %d; Byte 2 = %d \n", b1,b2);
   }
}
```

Tabela ASCII

Dec	Hex	Sinal	Caract.	Dec	Hex	Caract.	Dec	Hex	Caract.	Dec	Hex	Caract.
0	0	NULL		32	20	SPACE	64	40	@	96	60	`
1	1	SOH	☺	33	21	!	65	41	A	97	61	a
2	2	STX	☻	34	22	"	66	42	B	98	62	b
3	3	ETX	♥	35	23	#	67	43	C	99	63	c
4	4	EOT	♦	36	24	$	68	44	D	100	64	d
5	5	ENQ	♣	37	25	%	69	45	E	101	65	e
6	6	ACK	♠	38	26	&	70	46	F	102	66	f
7	7	BEL		39	27	'	71	47	G	103	67	g
8	8	BS		40	28	(	72	48	H	104	68	h
9	9	HT		41	29	)	73	49	I	105	69	i
10	A	LF		42	2A	*	74	4A	J	106	6A	j
11	B	VT	♂	43	2B	+	75	4B	K	107	6B	k
12	C	FF	♀	44	2C	,	76	4C	L	108	6C	l
13	D	CR		45	2D	-	77	4D	M	109	6D	m
14	E	SO	♫	46	2E	.	78	4E	N	110	6E	n
15	F	SI	☼	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	►	48	30	0	80	50	P	112	70	p
17	11	DC1	◄	49	31	1	81	51	Q	113	71	q
18	12	DC2	↕	50	32	2	82	52	R	114	72	r
19	13	DC3	‼	51	33	3	83	53	S	115	73	s
20	14	DC4	¶	52	34	4	84	54	T	116	74	t
21	15	NAK	§	53	35	5	85	55	U	117	75	u
22	16	SYN	▬	54	36	6	86	56	V	118	76	v
23	17	ETB	↨	55	37	7	87	57	W	119	77	w
24	18	CAN	↑	56	38	8	88	58	X	120	78	x
25	19	EM	↓	57	39	9	89	59	Y	121	79	y
26	1A	SUB		58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	←	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	∟	60	3C	<	92	5C	\	124	7C	¦
29	1D	GS	↔	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	▲	62	3E	>	94	5E	^	126	7E	~
31	1F	US	▼	63	3F	?	95	5F	_	127	7F	⌂

Dec	Hex	sinal	descrição	Dec	Hex	sinal	descrição
0	0	**NULL**	null	16	10	**DLE**	data link escape
1	1	**SOH**	start of heading	17	11	**DC1**	device control 1
2	2	**STX**	start of text	18	12	**DC2**	device control 2
3	3	**ETX**	end of text	19	13	**DC3**	device control 3
4	4	**EOT**	end of transmission	20	14	**DC4**	device control 4
5	5	**ENQ**	enquiry	21	15	**NAK**	native acknowledge
6	6	**ACK**	acknowledge	22	16	**SYN**	synchronous idle
7	7	**BEL**	bell	23	17	**ETB**	end of transmission block
8	8	**BS**	backspace	24	18	**CAN**	cancel
9	9	**HT**	horizontal tabulation	25	19	**EM**	end of medium
10	A	**LF**	linefeed	26	1A	**SUB**	substitute
11	B	**VT**	vertical tabulation	27	1B	**ESC**	escape
12	C	**FF**	form feed	28	1C	**FS**	file separator
13	D	**CR**	carriage return	29	1D	**GS**	group separator
14	E	**SO**	shift out	30	1E	**RS**	record separator
15	F	**SI**	shift in	31	1F	**US**	unit separator

Dec	Hex	Caract.	Dec	Hex	Caract.	Dec	Hex	Caract.	Dec	Hex	Caract.
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	∈
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	