Apostila de Linguagem C

PROFESSORES:
- Claudio Passos
- Flavio Costa

Tópico 1 - INTRODUÇÃO

A linguagem C foi criada por Dennis Ritchie, em 1972, no centro de Pesquisas da Bell Laboratories. Sua primeira utilização importante foi a reescrita do Sistema Operacional UNIX, que até então era escrito em assembly. Em meados de 1970 o UNIX saiu do laboratório para ser liberado para as universidades. Foi o suficiente para que o sucesso da linguagem atingisse proporções tais que, por volta de 1980, já existiam várias versões de compiladores C oferecidas por várias empresas, não sendo mais restritas apenas ao ambiente UNIX, porém compatíveis com vários outros sistemas operacionais. O C é uma linguagem de propósito geral, sendo adequada à programação estruturada. No entanto é mais ut iliz ada escrever compiladores, analisadores léxicos, bancos de dados, editores de texto, etc.. A linguagem C pertence a uma família de linguagens cujas características são: portabilidade, modularidade, compilação separada, recursos de baixo nível, geração de código eficiente, confiabilidade, regularidade, simplicidade e fac ilidade de uso.

TÓPICO 2 - Primeiros Passos

O C é "Case Sensitive"

Vamos começar o nosso curso ressaltando um ponto de suma importância: o C é "Case Sensitive", isto é, *maiúsculas e minúsculas fazem diferença*. Se declarar uma variável com o nome soma ela será diferente de **Soma**, **SOMA**, **SoMa** ou **sOmA**. Da mesma maneira, os comandos do C **if** e **for**, por exemplo, só podem ser escritos em minúsculas pois senão o compilador não irá interpretá-los como sendo comandos, mas sim como variáveis.

Dois Primeiros Programas

Vejamos um primeiro programa em C:

```
#include <stdio.h> #include <conio.h>
/* Um Primeiro Programa */
int main ()
{
    printf ("Ola! Eu estou vivo!\n");
    getch(); return(0);
}
```

Compilando e executando este programa você verá que ele coloca a mensagem *Ola! Eu estou vivo!* na tela.

Vamos analisar o programa por partes.

A linha **#include <stdio.h> #include <conio.h>#include <conio.h>**diz ao compilador que ele deve incluir o arquivo-cabeçalho **stdio.h**. Estes arquivos as vezes são chamados de bibliotecas, por conterem uma coleção de funções. Neste arquivo existem declarações de funções úteis para entrada e saída de dados (std = standard, padrão em inglês; io = Input/Output, entrada e saída ==> stdio = Entrada e saída padronizadas). Toda vez que você quiser usar uma destas funções deve-se incluir este comando. O C possui diversas bibliotecas.

Quando fazemos um programa, uma boa idéia é usar comentários que ajudem a elucidar o funcionamento do mesmo. No caso acima temos um comentário: **/* Um Primeiro Programa */**. O compilador C desconsidera qualquer coisa que esteja começando com **/*** e terminando com ***/**. Um comentário pode, inclusive, ter mais de uma linha.

A linha **int main()** indica que estamos definindo uma função de nome **main**. Todos os programas em C têm que ter uma função **main**, pois é esta função que será chamada quando o programa for executado. O conteúdo da função é delimitado por chaves **{ }**. O código que estiver dentro das chaves será executado seqüencialmente quando a função for chamada. A palavra int indica que esta função retorna um inteiro. O que significa este retorno será visto posteriormente, quando estudarmos um pouco mais detalhadamente as funções do C. A última linha do programa, **getch(); return(0);** , indica o número inteiro que está sendo retornado pela função, no caso o número 0. Fazemos isto para garantir que na compilação não de erro dizendo que a função não está retornando nada.

A única coisa que o programa *realmente* faz é chamar a função **printf()**, passando a string (uma string é uma seqüência de caracteres, como veremos brevemente) **"Ola! Eu estou vivo!\n"** como argumento. É por causa do uso da função **printf()** pelo programa que devemos incluir o arquivo- cabeçalho **stdio.h** . A função **printf()** neste caso irá apenas colocar a string na tela do computador. O **\n** é uma constante chamada de *constante barra invertida*. No caso, o **\n** é a constante barra invertida de "new line" e ele é interpretado como um comando de mudança de linha, isto é, após imprimir *Ola! Eu estou vivo!* o cursor passará para a próxima linha. É importante observar também que os *comandos* do C terminam com **;** .

Podemos agora tentar um programa mais complicado:

```
 #include <stdio.h>
 #include <conio.h>
 int main ()
{int Dias;           /* Declaracao de Variaveis */
 float Anos;
 printf ("Entre com o número de dias: "); /* Entrada de
Dados */
 scanf ("%d",&Dias);
 Anos=Dias/365.25;      /* Conversao Dias->Anos */
```

```
 printf ("\n\n%d dias equivalem a %f anos.\n",Dias,Anos);
 getch(); return(0);
}
```

Vamos entender como o programa acima funciona. São declaradas duas variáveis chamadas **Dias** e **Anos**. A primeira é um **int** (inteiro) e a segunda um **float** (ponto flutuante). As variáveis declaradas como ponto flutuante existem para armazenar números que possuem casas decimais, como 5,1497.

É feita então uma chamada à função **printf()**, que coloca uma mensagem na tela.

Queremos agora ler um dado que será fornecido pelo usuário e colocá-lo na variável inteira **Dias**. Para tanto usamos a função **scanf()**. A string **"%d"** diz à função que iremos ler um inteiro. O segundo parâmetro passado à função diz que o dado lido deverá ser armazenado na variável **Dias**. É importante ressaltar a necessidade de se colocar um **&** antes do nome da variável a ser lida quando se usa a função **scanf()**. O motivo disto só ficará claro mais tarde. Observe que, no C, quando temos mais de um parâmetro para uma função, eles serão separados por vírgula.

Temos então uma expressão matemática simples que atribui a **Anos** o valor de **Dias** dividido por 365.25 (365.25 é uma constante ponto flutuante 365,25). Como **Anos** é uma variável **float** o compilador fará uma conversão automática entre os tipos das variáveis (veremos isto com detalhes mais tarde).

A segunda chamada à função **printf()** tem três argumentos. A string **"\n\n%d dias equivalem a %f anos.\n"** diz à função para pular duas linhas, colocar um inteiro na tela, colocar a mensagem **" dias equivalem a "**, colocar um valor **float** na tela, colocar a mensagem **" anos."** e pular outra linha. Os outros parâmetros são as variáveis, **Dias** e **Anos**, das quais devem ser lidos os valores do inteiro e do **float**, respectivamente.

Introdução às Funções

Uma função é um bloco de código de programa que pode ser usado diversas vezes em sua execução. O uso de funções permite que o programa fique mais legível, mais bem estruturado. Um programa em C consiste, no fundo, de várias funções colocadas juntas.

Abaixo o tipo mais simples de função:

```
#include <stdio.h>
#include <conio.h>
     int mensagem ()  /* Funcao simples: so imprime Ola! */
     {
      printf ("Ola! ");
      return(0);
     }
     int main ()
```

```
{
 mensagem();
 printf ("Eu estou vivo!\n");
 getch();
 return(0); }
```

Este programa terá o mesmo resultado que o primeiro exemplo da seção anterior. O que ele faz é definir uma função **mensagem()** que coloca uma string na tela e retorna 0. Esta função é chamada a partir de **main()** , que, como já vimos, também é uma função. A diferença fundamental entre main e as demais funções do problema é que main é uma função especial, cujo diferencial é o fato de ser a primeira função a ser executada em um programa.

- Argumentos

Argumentos são as entradas que a função recebe. É através dos argumentos que passamos *parâmetros* para a função. Já vimos funções com argumentos. As funções **printf()** e **scanf()** são funções que recebem argumentos. Vamos ver um outro exemplo simples de função com argumentos:

```
#include <stdio.h>
 #include <conio.h>
int square (int x)  /* Calcula o quadrado de x */
{
  printf ("O quadrado e %d",(x*x));
 return(0);
}
int main ()
{
    int num;
    printf ("Entre com um numero: ");
    scanf ("%d",&num);
    printf ("\n\n");
    square(num);
    getch();
    return(0);
}
```

Na definição de **square()** dizemos que a função receberá um argumento inteiro **x**. Quando fazemos a chamada à função, o inteiro **num** é passado como argumento. Há alguns pontos a observar. Em primeiro lugar temos de satisfazer aos requisitos da função quanto ao tipo e à quantidade de argumentos quando a chamamos. Apesar de existirem algumas conversões de tipo, que o C faz automaticamente, é importante ficar atento. Em segundo lugar, não é importante o nome da variável que se passa como argumento, ou seja, a variável **num**, ao ser passada como argumento para **square()** é copiada para a variável **x**. Dentro de **square()** trabalha-se apenas com **x**. Se mudarmos o valor de **x** dentro de **square()** o valor de **num** na função **main()** permanece inalterado.

Vamos dar um exemplo de função de mais de uma variável. Repare que, neste caso, os argumentos são separados por vírgula e que deve-se explicitar o tipo de cada um dos argumentos, um a um. Note, também, que os argumentos passados para a função não necessitam ser todos variáveis porque mesmo sendo constantes serão copiados para a variável de entrada da função.

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

 int mult (float a, float b,float c)    /* Multiplica
3 numeros */
{
    printf ("%f",a*b*c);

}

int main ()
{
   float x,y;
   x=23.5;
   y=12.9;
   mult (x,y,3.87);
    getch(); return(0);
}
```

- Retornando valores

Muitas vezes é necessário fazer com que uma função retorne um valor. As funções que vimos até aqui estavam retornando o número 0. Podemos especificar um tipo de retorno indicando-o antes do nome da função. Mas para dizer ao C *o que* vamos retornar precisamos da palavra reservada **return**. Sabendo disto fica fácil fazer uma função para multiplicar dois inteiros e que retorna o resultado da multiplicação. Veja:

```
 #include <stdio.h>

#include <conio.h>

 int prod (int x,int y)
 {
    return (x*y);
 }

int main ()
{   int saida;
    saida=prod (12,7);
    printf ("A saida e: %d\n",saida);
    getch(); return(0);
}
```

Veja que, como prod retorna o valor de 12 multiplicado por 7, este valor pode ser usado em uma expressão qualquer. No programa fizemos a atribuição deste resultado à variável saida, que posteriormente foi impressa usando o printf. Uma observação adicional: se não especificarmos o tipo de retorno de uma função, o compilador C automaticamente suporá que este tipo é inteiro. Porém, não é uma boa prática não se especificar o valor de retorno e, neste curso, este valor será sempre especificado.

Com relação à função main, o retorno sempre será inteiro. Normalmente faremos a função main retornar um zero quando ela é executada sem qualquer tipo de erro.

Mais um exemplo de função, que agora recebe dois floats e também retorna um float::

```
#include <stdio.h> #include <conio.h>
float prod (float x,float y)
{
     return (x*y);
}

int main ()
{
     float saida;
     saida=prod (45.2,0.0067);
     printf ("A saida e: %f\n",saida);
     getch(); return(0);
}
```

- Forma geral

Apresentamos aqui a forma geral de uma função:

tipo_de_retorno nome_da_função (lista_de_argumentos)
{
código_da_função
}

Introdução Básica às Entradas e Saídas

- Caracteres

Os caracteres são um tipo de dado: o **char**. O C trata os caracteres ('a', 'b', 'x', etc ...) como sendo variáveis de um *byte* (8 *bit*s). Um *bit* é a menor unidade de armazenamento de informações em um computador. Os inteiros (**int**s) têm um número maior de *bytes*. Dependendo da implementação do compilador, eles podem ter 2 *bytes* (16 *bits*) ou 4 *bytes* (32 *bits*). Isto será melhor explicado na Tópico 3. Na linguagem C, também podemos usar um **char** para armazenar valores numéricos inteiros, além de usá-lo para armazenar

caracteres de texto. Para indicar um caractere de texto usamos apóstrofes. Veja um exemplo de programa que usa caracteres:

```
#include <stdio.h> #include <conio.h>
int main ()
{
      char Ch;
      Ch='D';
      printf ("%c",Ch);
      getch(); return(0);
}
```

No programa acima, **%c** indica que **printf()** deve colocar um caractere na tela. Como vimos anteriormente, um **char** também é usado para armazenar um número inteiro. Este número é conhecido como o código ASCII correspondente ao caractere. Veja o programa abaixo:

```
  #include <stdio.h> #include <conio.h>
  int main ()
  {
    char Ch;
    Ch='D';
    printf ("%d",Ch);  /* Imprime o caracter como inteiro
*/
    getch(); return(0);
  }
```

Este programa vai imprimir o número 68 na tela, que é o código ASCII correspondente ao caractere 'D' (d maiúsculo).

Muitas vezes queremos ler um caractere fornecido pelo usuário. Para isto as funções mais usadas, são **getch()** e **getche()**. Ambas retornam o caractere pressionado. **getche()** imprime o caractere na tela antes de retorná-lo e **getch()** apenas retorna o caractere pressionado sem imprimí-lo na tela. Ambas as funções podem ser encontradas no arquivo de cabeçalho **conio.h**. Podem ser substituídas pela função scanf(), porém sem as mesmas funcionalidades. Em muitos casos usamos a função getch() para manter nosso programa parado em uma linha do código, até que o usuário pressione qualquer tecla.

```
#include <stdio.h>

#include <conio.h>
/* Este programa usa conio.h . Se você não tiver a conio,
ele não funcionará no Unix */
int main ()
{
      char Ch;
      Ch=getch();
      printf ("Voce pressionou a tecla %c",Ch);
      getch(); return(0);
```

```
}
```

Equivalente ANSI-C para o ambiente Unix do programa acima, sem usar getch():

```
#include <stdio.h> #include <conio.h>
int main ()
{
       char Ch;
       scanf("%c", &Ch);
       printf ("Voce pressionou a tecla %c",Ch);
       getch(); return(0);
}
```

- Strings

No C uma string é um vetor de caracteres terminado com um caractere nulo. O caracter nulo é um caractere com valor inteiro igual a zero (código ASCII igual a 0). O terminador nulo também pode ser escrito usando a convenção de barra invertida do C como sendo '\0'. Embora o assunto vetores seja discutido posteriormente, veremos aqui os fundamentos necessários para que possamos utilizar as strings. Para declarar uma string, podemos usar o seguinte formato geral:

char nome_da_string[tamanho];

Isto declara um vetor de caracteres (uma string) com número de posições igual a *tamanho*. Note que, como temos que reservar um caractere para ser o terminador nulo, temos que declarar o comprimento da string como sendo, no mínimo, um caractere maior que a maior string que pretendemos armazenar. Vamos supor que declaremos uma string de 7 posições e coloquemos a palavra João nela. Teremos:

J	o	a	o	\0	...	...

No caso acima, as duas células não usadas têm valores indeterminados. Isto acontece porque o C *não* inicializa variáveis, cabendo ao programador esta tarefa. Portanto as únicas células que são inicializadas são as que contêm os caracteres 'J', 'o', 'a', 'o' e '\0' .

Se quisermos ler uma string fornecida pelo usuário podemos usar a função **gets()**. Um exemplo do uso desta função é apresentado abaixo. A função **gets()** coloca o terminador nulo na string, quando você aperta a tecla "Enter".

```
#include <stdio.h> #include <conio.h>
int main ()
{
    char string[100];
    printf ("Digite uma string: ");
    gets (string);
```

```
    printf ("\n\nVoce digitou %s",string);
    getch(); return(0);
}
```

Neste programa, o tamanho máximo da string que você pode entrar é uma string de 99 caracteres. Se você entrar com uma string de comprimento maior, o programa irá aceitar, mas os resultados podem ser desastrosos. Veremos porque posteriormente.

Como as strings são vetores de caracteres, para se acessar um determinado caracter de uma string, basta "indexarmos", ou seja, usarmos um índice para acessarmos o caracter desejado dentro da string. Suponha uma string chamada *str*. Podemos acessar a **segunda** letra de *str* da seguinte forma:

```
str[1] = 'a';
```

Por quê se está acessando a segunda letra e não a primeira? Na linguagem C, o índice ***começa em zero.*** Assim, a primeira letra da string sempre estará na posição 0. A segunda letra sempre estará na posição 1 e assim sucessivamente. Segue um exemplo que imprimirá a segunda letra da string "Joao", apresentada acima. Em seguida, ele mudará esta letra e apresentará a string no final.

```
#include <stdio.h> #include <conio.h>
int main()
{
    char str[10] = "Joao";
    printf("\n\nString: %s", str);
    printf("\nSegunda letra: %c", str[1]);
    str[1] = 'U';
    printf("\nAgora a segunda letra eh: %c", str[1]);
    printf("\n\nString resultante: %s", str);
    getch(); return(0);
}
```

Nesta string, o terminador nulo está na posição 4. Das posições 0 a 4, sabemos que temos caracteres válidos, e portanto podemos escrevê-los. Note a forma como inicializamos a string **str** com os caracteres 'J' 'o' 'a' 'o' e '\0' simplesmente declarando char str[10] = "Joao". Veremos, posteriormente que "Joao" (uma cadeia de caracteres entre aspas) é o que chamamos de string constante, isto é, uma cadeia de caracteres que está pré-carregada com valores que não podem ser modificados. Já a string str é uma string variável, pois podemos modificar o que nela está armazenado, como de fato fizemos.

No programa acima, **%s** indica que **printf()** deve colocar uma string na tela. Vamos agora fazer uma abordagem inicial às duas funções que já temos usado para fazer a entrada e saída.

- printf

A função **printf()** tem a seguinte forma geral:

printf (string_de_controle,lista_de_argumentos);

Teremos, na string de controle, uma descrição de tudo que a função vai colocar na tela. A string de controle mostra não apenas os caracteres que devem ser colocados na tela, mas também quais as variáveis e suas respectivas posições. Isto é feito usando-se os códigos de controle, que usam a notação **%**. Na string de controle indicamos quais, de qual tipo e em que posição estão as variáveis a serem apresentadas. É muito importante que, para cada código de controle, tenhamos um argumento na lista de argumentos. Apresentamos agora alguns dos códigos **%**:

Código	Significado
%d	Inteiro
%f	Float
%c	Caractere
%s	String
%%	Coloca na tela um %

Vamos ver alguns exemplos de **printf()** e o que eles exibem:

```
printf ("Teste %% %%") -> "Teste % %"
printf ("%f",40.345) -> "40.345"
printf ("Um caractere %c e um inteiro %d",'D',120) -> "Um
caractere D e um inteiro 120"
printf ("%s e um exemplo","Este") -> "Este e um exemplo"
printf ("%s%d%%","Juros de ",10) -> "Juros de 10%"
```

Maiores detalhes sobre a função **printf()** (incluindo outros códigos de controle) serão vistos posteriormente, mas podem ser consultados de antemão pelos interessados.

- scanf

O formato geral da função **scanf()** é:

scanf (string-de-controle,lista-de-argumentos);

Usando a função **scanf()** podemos pedir dados ao usuário. Um exemplo de uso, pode ser visto acima. Mais uma vez, devemos ficar atentos a fim de colocar o mesmo número de argumentos que o de códigos de controle na string de controle. Outra coisa importante é lembrarmos de colocar o **&** antes das variáveis da lista de argumentos. É impossível justificar isto agora, mas veremos depois a razão para este procedimento. Maiores detalhes sobre a função **scanf()** serão vistos posteriormente, mas podem ser consultados de antemão pelos interessados.

Introdução a Alguns Comandos de Controle de Fluxo

Os comandos de controle de fluxo são aqueles que permitem ao programador alterar a sequência de execução do programa. Vamos dar uma breve introdução a dois comandos de controle de fluxo. Outros comandos serão estudados posteriormente.

- if

O comando **if** representa uma tomada de decisão do tipo "SE isto ENTÃO aquilo". A sua forma geral é:

if (condição) declaração;

A condição do comando **if** é uma expressão que será avaliada. Se o resultado for zero a declaração não será executada. Se o resultado for qualquer coisa diferente de zero a declaração será executada. A declaração pode ser um bloco de código ou apenas um comando. É interessante notar que, no caso da declaração ser um bloco de código, não é necessário (e nem permitido) o uso do **;** no final do bloco. Isto é uma regra geral para blocos de código. Abaixo apresentamos um exemplo:

```
#include <stdio.h> #include <conio.h>
int main ()
{
       int num;
       printf ("Digite um numero: ");
       scanf ("%d",&num);
       if (num>10) printf ("\n\nO numero e maior que 10");
       if (num==10)
       {
              printf ("\n\nVoce acertou!\n");
              printf ("O numero e igual a 10.");
         }
       if (num<10) printf ("\n\nO numero e menor que 10");
       getch()
       return (0);
}
```

No programa acima a expressão **num>10** é avaliada e retorna um valor diferente de zero, se verdadeira, e zero, se falsa. No exemplo, se num for maior que 10, será impressa a frase: "O número e maior que 10". Repare que, se o número for igual a 10, estamos executando dois comandos. Para que isto fosse possível, tivemos que agrupa-los em um bloco que se inicia logo após a comparação e termina após o segundo printf. Repare também que quando queremos testar igualdades usamos o operador **==** e não **=**. Isto porque o operador **=** representa *apenas* uma atribuição. Pode parecer estranho à primeira vista, mas se escrevêssemos

```
if (num=10) ...        /* Isto esta errado */
```

o compilador iria *atribuir* o valor 10 à variável **num** e a expressão **num=10** iria retornar 10, fazendo com que o nosso valor de **num** fosse modificado e fazendo com que a declaração fosse executada sempre. Este problema gera erros frequentes entre iniciantes e, portanto, muita atenção deve ser tomada.

Os operadores de comparação são: **==** (igual)**, !=** (diferente de), **>** (maior que)**, <** (menor que)**, >=** (maior ou igual**), <=** (menor ou igual)**.**

- for

O loop (laço) **for** é usado para repetir um comando, ou bloco de comandos, diversas vezes, de maneira que se possa ter um bom controle sobre o loop. Sua forma geral é:

for (inicialização;condição;incremento) declaração;

A declaração no comando for também pode ser um bloco ({ }) e neste caso o **;** é omitido. O melhor modo de se entender o loop **for** é ver de que maneira ele funciona "por dentro". O loop **for** é equivalente a se fazer o seguinte:

inicialização;
if (condição)
{
declaração;
incremento;
"Volte para o comando if"
}

Podemos ver que o **for** executa a inicialização incondicionalmente e testa a condição. Se a condição for falsa ele não faz mais nada. Se a condição for verdadeira ele executa a declaração, o incremento e volta a testar a condição. Ele fica repetindo estas operações até que a condição seja falsa. Abaixo vemos um programa que coloca os primeiros 100 números na tela:

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

int main ()
{
 int count;
 for (count=1;count<=100;count=count+1)
    printf ("%d ",count);
 getch(); return(0);
}
```

Outro exemplo interessante é mostrado a seguir: o programa lê uma string e conta quantos dos caracteres desta string são iguais à letra 'c'

```
#include <stdio.h> #include <conio.h>
int main ()
{
    char string[100];     /* String, ate' 99 caracteres */
    int i, cont;
    printf("\n\nDigite uma frase: ");
    gets(string); /* Le a string */
    printf("\n\nFrase digitada:\n%s", string);
    cont = 0;
    for (i=0; string[i] != '\0'; i=i+1)
    {
        if ( string[i] == 'c' )      /* Se for a letra 'c'
*/
        cont = cont +1;          /* Incrementa o contador
de caracteres */
    }
    printf("\nNumero de caracteres c = %d", cont);
    getch(); return(0);
}
```

Note o teste que está sendo feito no for: o caractere armazenado em string[i] é comparado com '\0' (caractere final da string). Caso o caractere seja diferente de '\0', a condição é verdadeira e o bloco do for é executado. Dentro do bloco existe um if que testa se o caractere é igual a 'c'. Caso seja, o contador de caracteres c é incrementado.

Mais um exemplo, agora envolvendo caracteres:

```
/* Este programa imprime o alfabeto: letras maiúsculas  */
#include <stdio.h> #include <conio.h>
int main()
{
    char letra;
    for(letra = 'A' ; letra <= 'Z' ; letra =letra+1)
    printf("%c ", letra);
      getch();
}
```

Este programa funciona porque as letras maiúsculas de A a Z possuem código inteiro sequencial.

Palavras Reservadas do C

Todas as linguagens de programação têm palavras reservadas. As palavras reservadas não podem ser usadas a não ser nos seus propósitos originais, isto é, não podemos declarar funções ou variáveis com os mesmos nomes. Como o C é "case sensitive" podemos declarar uma variável **For**, apesar de haver uma palavra reservada **for**, mas isto não é uma coisa recomendável de se fazer pois pode gerar confusão.

Apresentamos a seguir as palavras reservadas do ANSI C. Veremos o significado destas palavras chave à medida em que o curso for progredindo:

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

TÓPICO 3 - VARIÁVEIS, CONSTANTES, OPERADORES E EXPRESSÕES

Nomes de Variáveis

As variáveis no C podem ter qualquer nome se duas condições forem satisfeitas: o nome deve começar com uma letra ou sublinhado (_) e os caracteres subsequentes devem ser letras, números ou sublinhado (_). Há apenas mais duas restrições: o nome de uma variável não pode ser igual a uma palavra reservada, nem igual ao nome de uma função declarada pelo programador, ou pelas bibliotecas do C. Variáveis de até 32 caracteres são aceitas. Mais uma coisa: é bom sempre lembrar que o C é "case sensitive" e portanto deve-se prestar atenção às maiúsculas e minúsculas.

Dicas quanto aos nomes de variáveis...

- É uma prática tradicional do C, usar letras minúsculas para nomes de variáveis e maiúsculas para nomes de constantes. Isto facilita na hora da leitura do código;
- Quando se escreve código usando nomes de variáveis em português, evita-se possíveis conflitos com nomes de rotinas encontrados nas diversas bibliotecas, que são em sua maioria absoluta, palavras em inglês.

Os Tipos do C

O C tem 5 tipos básicos: **char**, **int**, **float**, **void, double**. Destes não vimos ainda os dois últimos: O **double** é o ponto flutuante duplo e pode ser visto como um ponto flutuante com muito mais precisão. O void é o tipo vazio, ou um "tipo sem tipo". A aplicação deste "tipo" será vista posteriormente.

Para cada um dos tipos de variáveis existem os modificadores de tipo. Os modificadores de tipo do C são quatro: **signed**, **unsigned**, **long** e **short**. Ao **float** não se pode aplicar nenhum e ao **double** pode-se aplicar apenas o **long**.

Os quatro modificadores podem ser aplicados a inteiros. A intenção é que **short** e **long** devam prover tamanhos diferentes de inteiros onde isto for prático. Inteiros menores (**short**) ou maiores (**long**). **int** normalmente terá o tamanho natural para uma determinada máquina. Assim, numa máquina de 16 bits, **int** provavelmente terá 16 bits. Numa máquina de 32, **int** deverá ter 32 bits. Na verdade, cada compilador é livre para escolher tamanhos adequados para o seu próprio hardware, com a única restrição de que **shorts ints** e **ints** devem ocupar pelo menos 16 bits, **longs ints** pelo menos 32 bits, e **short int** não pode ser maior que **int**, que não pode ser maior que **long int**. O modificador **unsigned** serve para especificar variáveis sem sinal. Um **unsigned int** será um inteiro que assumirá apenas valores positivos. A seguir estão listados os tipos de dados permitidos e seu valores máximos e mínimos em um compilador típico para um hardware de 16 bits. Também nesta tabela está especificado o formato que deve ser utilizado para ler os tipos de dados com a função scanf():

Tipo	Num de bits	Formato para leitura com scanf	Intervalo	
			Inicio	Fim
char	8	%c	-128	127
unsigned char	8	%c	0	255
signed char	8	%c	-128	127
int	16	%i	-32.768	32.767
unsigned int	16	%u	0	65.535
signed int	16	%i	-32.768	32.767
short int	16	%hi	-32.768	32.767
unsigned short int	16	%hu	0	65.535
signed short int	16	%hi	-32.768	32.767
long int	32	%li	-2.147.483.648	2.147.483.647
signed long int	32	%li	-2.147.483.648	2.147.483.647
unsigned long int	32	%lu	0	4.294.967.295
float	32	%f	3,4E-38	3.4E+38
double	64	%lf	1,7E-308	1,7E+308
long double	80	%Lf	3,4E-4932	3,4E+4932

O tipo **long double** é o tipo de ponto flutuante com maior precisão. É importante observar que os intervalos de ponto flutuante, na tabela acima, estão indicados em faixa de *expoente*, mas os números podem assumir valores tanto positivos quanto negativos.

Declaração e Inicialização de Variáveis

As variáveis no C devem ser declaradas antes de serem usadas. A forma geral da declaração de variáveis é:

tipo_da_variável lista_de_variáveis;

As variáveis da lista de variáveis terão todas o mesmo tipo e deverão ser separadas por vírgula. Como o tipo default do C é o **int**, quando vamos declarar variáveis **int** com algum dos modificadores de tipo, basta colocar o nome do modificador de tipo. Assim um **long** basta para declarar um **long int**.

Por exemplo, as declarações

```
char ch, letra;
long count;
float pi;
```

declaram duas variáveis do tipo **char** (ch e letra), uma variavel **long int** (count) e um **float** pi.

Há três lugares nos quais podemos declarar variáveis. O primeiro é fora de todas as funções do programa. Estas variáveis são chamadas **variáveis globais** e podem ser usadas a partir de qualquer lugar no programa. Pode-se dizer que, como elas estão fora de todas as funções, todas as funções as vêem. O segundo lugar no qual se pode declarar variáveis é **no início** de um bloco de código. Estas variáveis são chamadas **locais** e só têm validade dentro do bloco no qual são declaradas, isto é, só a função à qual ela pertence sabe da existência desta variável, dentro do bloco no qual foram declaradas. O terceiro lugar onde se pode declarar variáveis é na **lista de parâmetros** de uma função. Mais uma vez, apesar de estas variáveis receberem valores externos, estas variáveis são conhecidas apenas pela função onde são declaradas.

Veja o programa abaixo:

```
#include <stdio.h>
   int contador;
   int func1(int j) {
   /* aqui viria o código da funcao
      ...
*/
   }
   int main()
   {
     char condicao;
     int i;
     for (i=0; i<100; i=i+1)
     {              /* Bloco do for */
       float f2;
       /* etc ...
       ...
    */
       func1(i);
     }
```

```
    /* etc  ...  */
getch(); return(0);
  }
```

A variável *contador* é uma variável global, e é acessível de qualquer parte do programa. As variáveis *condição* e *i,* só existem dentro de main(), isto é são variáveis locais de main. A variável float *f2* é um exemplo de uma variável de bloco, isto é, ela somente é conhecida dentro do bloco do for, pertencente à função main. A variável inteira *j* é um exemplo de declaração na lista de parâmetros de uma função (a função *func1*).

As regras que regem *onde* uma variável é válida chamam-se regras de *escopo* da variável. Há mais dois detalhes que devem ser ressaltados. Duas variáveis globais não podem ter o mesmo nome. O mesmo vale para duas variáveis locais de uma mesma função. Já duas variáveis locais, de funções diferentes, podem ter o mesmo nome sem perigo algum de conflito.

Podemos inicializar variáveis no momento de sua declaração. Para fazer isto podemos usar a forma geral

tipo_da_variável nome_da_variável = constante;

Isto é importante pois quando o C cria uma variável ele *não* a inicializa. Isto significa que até que um primeiro valor seja atribuído à nova variável ela tem um valor *indefinido* e que não pode ser utilizado para nada. *Nunca* presuma que uma variável declarada vale zero ou qualquer outro valor. Exemplos de inicialização são dados abaixo :

```
  char ch='D';
  int count=0;
float pi=3.141;
```

Ressalte-se novamente que, em C, uma variável tem que ser declarada no início de um bloco de código. Assim, o programa a seguir não é válido em C (embora seja válido em C++).

```
int  main()
{
        int i;
        int j;
        j = 10;
        int k = 20;  /* Esta declaracao de variável não é
válida, pois não está sendo feita no início do bloco */
      getch(); return(0);
}
```

Constantes

Constantes são valores que são mantidos fixos pelo compilador. Já usamos constantes neste curso. São consideradas constantes, por exemplo, os números e caracteres como 45.65 ou 'n', etc...

- Constantes dos tipos básicos

Abaixo vemos as constantes relativas aos tipos básicos do C:

Tipo de Dado	Exemplos de Constantes
char	'b' '\n' '\0'
int	2 32000 -130
long int	100000 -467
short int	100 -30
unsigned int	50000 35678
float	0.0 23.7 -12.3e-10
double	12546354334.0 -0.0000034236556

- Constantes hexadecimais e octais

Muitas vezes precisamos inserir constantes hexadecimais (base dezesseis) ou octais (base oito) no nosso programa. O C permite que se faça isto. As constantes hexadecimais começam com 0x. As constantes octais começam em 0.

Alguns exemplos:

Constante	Tipo
0xEF	Constante Hexadecimal (8 bits)
0x12A4	Constante Hexadecimal (16 bits)
03212	Constante Octal (12 bits)
034215432	Constante Octal (24 bits)

Nunca escreva portanto 013 achando que o C vai compilar isto como se fosse 13. Na linguagem C 013 é diferente de 13!

- Constantes strings

Já mostramos como o C trata strings. Vamos agora alertar para o fato de que uma string **"Joao"** é na realidade uma constante string. Isto implica, por exemplo, no fato de que **'t'** é diferente de **"t"**, pois **'t'** é um **char** enquanto que **"t"** é uma constante string com dois **chars** onde o primeiro é **'t'** e o segundo é **'\0'**.

- Constantes de barra invertida

O C utiliza, para nos facilitar a tarefa de programar, vários códigos chamados códigos de barra invertida. Estes são caracteres que podem ser usados como qualquer outro. Uma lista com alguns dos códigos de barra invertida é dada a seguir:

Código	Significado
\b	Retrocesso ("back")
\f	Alimentação de formulário ("form feed")
\n	Nova linha ("new line")
\t	Tabulação horizontal ("tab")
\"	Aspas
\'	Apóstrofo
\0	Nulo (0 em decimal)
\\	Barra invertida
\v	Tabulação vertical
\a	Sinal sonoro ("beep")
\N	Constante octal (N é o valor da constante)
\xN	Constante hexadecimal (N é o valor da constante)

Operadores Aritméticos e de Atribuição

Os operadores aritméticos são usados para desenvolver operações matemáticas. A seguir apresentamos a lista dos operadores aritméticos do C:

Operador	Ação
+	Soma (inteira e ponto flutuante)
-	Subtração ou Troca de sinal (inteira e ponto flutuante)
*****	Multiplicação (inteira e ponto flutuante)
/	Divisão (inteira e ponto flutuante)
%	Resto de divisão (de inteiros)
++	Incremento (inteiro e ponto flutuante)
--	Decremento (inteiro e ponto flutuante)

O C possui operadores unários e binários. Os unários agem sobre uma variável apenas, modificando ou não o seu valor, e retornam o valor final da variável. Os binários usam duas variáveis e retornam um terceiro valor, sem alterar as variáveis originais. A soma é um operador binário pois pega duas variáveis, soma seus valores, sem alterar as variáveis, e retorna esta soma. Outros operadores binários são os operadores **-** (subtração), *****, **/** e **%**. O

operador **-** como troca de sinal é um operador unário que não altera a variável sobre a qual é aplicado, pois ele retorna o valor da variável multiplicado por -1.

O operador / (divisão) quando aplicado a variáveis inteiras, nos fornece o resultado da divisão inteira; quando aplicado a variáveis em ponto flutuante nos fornece o resultado da divisão "real". O operador % fornece o resto da divisão de dois inteiros.

Assim seja o seguinte trecho de código:

```
 int a = 17, b = 3;
int x, y;
float z = 17. , z1, z2;
x = a / b;
y = a % b;
z1 = z / b;
z2 = a/b;
```

ao final da execução destas linhas, os valores calculados seriam x = 5, y = 2, z1 = 5.666666 e z2 = 5.0 . Note que, na linha correspondente a z2, primeiramente é feita uma divisão inteira (pois os dois operandos são inteiros). Somente após efetuada a divisão é que o resultado é atribuído a uma variável float.

Os operadores de incremento e decremento são unários que alteram a variável sobre a qual estão aplicados. O que eles fazem é incrementar ou decrementar, a variável sobre a qual estão aplicados, de 1. Então

```
x++;
x--;
```

são equivalentes a

```
x=x+1;
x=x-1;
```

Estes operadores podem ser pré-fixados ou pós- fixados. A diferença é que quando são pré-fixados eles incrementam e retornam o valor da variável já incrementada. Quando são pós-fixados eles retornam o valor da variável sem o incremento e depois incrementam a variável. Então, em

```
x=23;
y=x++;
```

teremos, no final, **y=23** e **x=24**. Em

```
x=23;
y=++x;
```

teremos, no final, **y=24** e **x=24**. Uma curiosidade: a linguagem de programação C++ tem este nome pois ela seria um "incremento" da linguagem C padrão. A

linguagem C++ é igual à linguagem C só que com extensões que permitem a programação orientada a objeto, o que é um recurso extra.

O operador de atribuição do C é o **=**. O que ele faz é pegar o valor à direita e atribuir à variável da esquerda. Além disto ele retorna o valor que ele atribuiu. Isto faz com que as seguintes expressões sejam válidas:

```
x=y=z=1.5;              /* Expressao 1 */

  if (k=w) ...          /* Expressão 2 */
```

A expressão 1 é válida, pois quando fazemos **z=1.5** ela retorna 1.5, que é passado adiante, fazendo y = 1.5 e posteriormente x = 1.5. A expressão 2 será verdadeira se **w** for diferente de zero, pois este será o valor retornado por **k=w**. Pense bem antes de usar a expressão dois, pois ela pode gerar erros de interpretação. Você *não* está comparando **k** e **w**. Você está atribuindo o valor de **w** a **k** e usando este

Operadores Relacionais e Lógicos

Os operadores relacionais do C realizam ***comparações*** entre variáveis.

São eles:

Operador	Ação
>	Maior do que
>=	Maior ou igual a
<	Menor do que
<=	Menor ou igual a
==	Igual a
!=	Diferente de

Os operadores relacionais retornam verdadeiro (1) ou falso (0). Para verificar o funcionamento dos operadores relacionais, execute o programa abaixo:

```
/* Este programa ilustra o funcionamento dos operadores
relacionais. */
#include <stdio.h> #include <conio.h>
int main()
{
    int i, j;
    printf("\nEntre com dois números inteiros: ");
    scanf("%d%d", &i, &j);
    printf("\n%d == %d é %d\n", i, j, i==j);
    printf("\n%d != %d é %d\n", i, j, i!=j);
    printf("\n%d <= %d é %d\n", i, j, i<=j);
    printf("\n%d >= %d é %d\n", i, j, i>=j);
    printf("\n%d < %d é %d\n", i, j, i<j);
```

```
    printf("\n%d > %d é %d\n", i, j, i>j);
    getch(); return(0);
}
```

Você pode notar que o resultado dos operadores relacionais é sempre igual a 0 (falso) ou 1 (verdadeiro).

Para fazer ***operações com valores lógicos*** (verdadeiro e falso) temos ***os operadores lógicos***:

Operador	Ação
&&	AND (E)
||	OR (OU)
!	NOT (NÃO)

Usando os operadores relacionais e lógicos podemos realizar uma grande gama de testes. A tabela-verdade destes operadores é dada a seguir:

p	q	p AND q	p OR q
falso	falso	falso	falso
falso	verdadeiro	falso	verdadeiro
verdadeiro	falso	falso	verdadeiro
verdadeiro	verdadeiro	verdadeiro	verdadeiro

O programa a seguir ilustra o funcionamento dos operadores lógicos. Compile-o e faça testes com vários valores para i e j:

```
#include <stdio.h> #include <conio.h>
int main()
{
    int i, j;
    printf("informe dois números(cada um sendo 0 ou 1): ");
    scanf("%d%d", &i, &j);
    printf("%d AND %d é %d\n", i, j, i && j);
    printf("%d OR %d é %d\n", i, j, i || j);
    printf("NOT %d é %d\n", i, !i);
}
```

Exemplo: No trecho de programa abaixo a operação j++ será executada, pois o resultado da expressão lógica é verdadeiro:

```
int i = 5, j =7;
if ( (i > 3) && ( j <= 7)  && ( i != j) )  j++;
        V    AND    V     AND   V  = V
```

Mais um exemplo. O programa abaixo, imprime na tela somente os números pares entre 1 e 100, apesar da variação de i ocorrer de 1 em 1:

```
/* Imprime os números pares entre 1 e 100. */
```

```
#include <stdio.h> #include <conio.h>
int main()
{
    int i;
    for(i=1; i<=100; i++)
        if(!(i%2)) printf("%d ",i);     /* o operador de
resto dará falso (zero) */
}                                       /* quando usada c/
número par. Esse resultado*/
                                        /* é invertido pelo
! */
```

Expressões

Expressões são combinações de variáveis, constantes e operadores. Quando montamos expressões temos que levar em consideração a ordem com que os operadores são executados, conforme a tabela de precedências da linguagem C.

Exemplos de expressões:

```
Anos=Dias/365.25;
i = i+3;
c= a*b + d/e;
c= a*(b+d)/e;
```

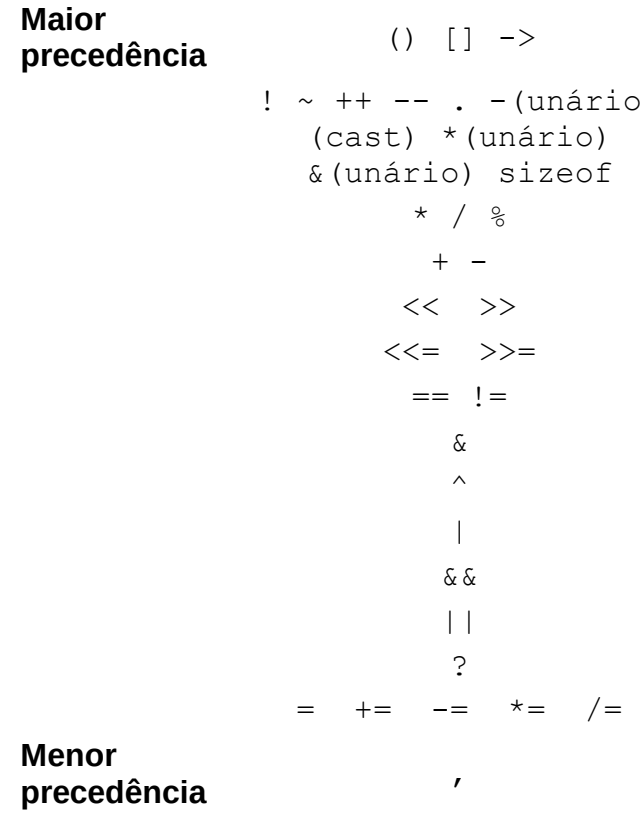
- Conversão de tipos em expressões

Quando o C avalia expressões onde temos variáveis de tipos diferentes o compilador verifica se as conversões são possíveis. Se não são, ele não compilará o programa, dando uma mensagem de erro. Se as conversões forem possíveis ele as faz, seguindo as regras abaixo:

1. Todos os **char**s e **short int**s são convertidos para **int**s. Todos os **float**s são convertidos para **double**s.
2. Para pares de operandos de tipos diferentes: se um deles é **long double** o outro é convertido para **long double**; se um deles é **double** o outro é convertido para **double**; se um é **long** o outro é convertido para **long**; se um é **unsigned** o outro é convertido para **unsigned**.

- Tabela de Precedências do C

Esta é a tabela de precedência dos operadores em C. Alguns (poucos) operadores ainda não foram estudados, e serão apresentados em Tópicos posteriores.



Uma dica aos iniciantes: Você não precisa saber toda a tabela de precedências de cor. É útil que você conheça as principais relações, mas é aconselhável que ao escrever o seu código, você tente isolar as expressões com parênteses, para tornar o seu programa mais legível.

Tópico 4 - ESTRUTURAS DE CONTROLE DE FLUXO

As estruturas de controle de fluxo são fundamentais para qualquer linguagem de programação. Sem elas só haveria uma maneira do programa ser executado: de cima para baixo comando por comando. Não haveria condições, repetições ou saltos. A linguagem C possui diversos comandos de controle de fluxo. É possível resolver todos os problemas sem utilizar todas elas, mas devemos nos lembrar que a elegância e facilidade de entendimento de um programa dependem do uso correto das estruturas no local certo.

O Comando if

Já introduzimos o comando **if**. Sua forma geral é:

if (condição) declaração;

A expressão, na condição, será avaliada. Se ela for zero, a declaração não será executada. Se a condição for diferente de zero a declaração será executada. Aqui reapresentamos o exemplo de um uso do comando **if** :

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

```
int main ()
{
       int num;
       printf ("Digite um numero: ");
       scanf ("%d",&num);
       if (num>10)
              printf ("\n\nO numero e maior que 10");
       if (num==10)
         {
              printf ("\n\nVoce acertou!\n");
              printf ("O numero e igual a 10.");
         }
       if (num<10)
              printf ("\n\nO numero e menor que 10");
       getch(); return(0);
}
```

- O else

Podemos pensar no comando **else** como sendo um complemento do comando **if**. O comando **if** completo tem a seguinte forma geral:

if (condição) declaração_1;
else declaração_2;

A expressão da condição será avaliada. Se ela for diferente de zero a declaração 1 será executada. Se for zero a declaração 2 será executada. É importante nunca esquecer que, quando usamos a estrutura **if-else**, estamos garantindo que uma das duas declarações será executada. Nunca serão executadas as duas ou nenhuma delas. Abaixo está um exemplo do uso do **if-else** que deve funcionar como o programa da seção anterior.

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

```
int main ()
{
       int num;
       printf ("Digite um numero: ");
       scanf ("%d",&num);
       if (num==10)
       {
              printf ("\n\nVoce acertou!\n");
              printf ("O numero e igual a 10.\n");
       }
       else
       {
              printf ("\n\nVoce errou!\n");
              printf ("O numero e diferente de 10.\n");
       }
       getch(); return(0);
}
```

- O if-else-if

A estrutura **if-else-if** é apenas uma extensão da estrutura **if-else**. Sua forma geral pode ser escrita como sendo:

if (condição_1) declaração_1;
else if (condição_2) declaração_2;
else if (condição_3) declaração_3;
.
.
.
else if (condição_n) declaração_n;
else declaração_default;

A estrutura acima funciona da seguinte maneira: o programa começa a testar as condições começando pela 1 e continua a testar até que ele ache uma expressão cujo resultado dê diferente de zero. Neste caso ele executa a declaração correspondente. Só uma declaração será executada, ou seja, só será executada a declaração equivalente à *primeira* condição que der diferente de zero. A última declaração (default) é a que será executada no caso de todas as condições darem zero e é opcional.

Um exemplo da estrutura acima:

```
#include <stdio.h> #include <conio.h>
int main ()
{
       int num;
       printf ("Digite um numero: ");
       scanf ("%d",&num);
       if (num>10)
             printf ("\n\nO numero e maior que 10");
       else if (num==10)
         {
             printf ("\n\nVoce acertou!\n");
             printf ("O numero e igual a 10.");
         }
       else if (num<10)
              printf ("\n\nO numero e menor que 10");
       getch(); return(0);
}
```

- A expressão condicional

Quando o compilador avalia uma condição, ele quer um valor de retorno para poder tomar a decisão. Mas esta expressão não necessita ser uma expressão no sentido convencional. Uma variável sozinha pode ser uma "expressão" e esta retorna o seu próprio valor. Isto quer dizer que teremos as seguintes expressões:

```
int num;
   if (num!=0) ....
   if (num==0) ....
for (i = 0; string[i] != '\0'; i++)
```

equivalem a

```
   int num;
   if (num) ....
   if (!num) ....
for (i = 0; string[i]; i++)
```

Isto quer dizer que podemos simplificar algumas expressões simples.

- ifs aninhados

O **if** aninhado é simplesmente um **if** dentro da declaração de um outro **if** externo. O único cuidado que devemos ter é o de saber exatamente a qual **if** um determinado **else** está ligado.

Vejamos um exemplo:

```
#include <stdio.h> #include <conio.h>
int main ()
{
       int num;
       printf ("Digite um numero: ");
       scanf ("%d",&num);
       if (num==10)
         {
              printf ("\n\nVoce acertou!\n");
              printf ("O numero e igual a 10.\n");
         }
       else
         {
              if (num>10)
                 {
                    printf ("O numero e maior que 10.");
                 }
              else
                 {
                    printf ("O numero e menor que 10.");
                 }
         }
       getch(); return(0);
}
```

O Comando switch

O comando **if-else** e o comando **switch** são os dois comandos de tomada de decisão. Sem dúvida alguma o mais importante dos dois é o **if**, mas o comando **switch** tem aplicações valiosas. Mais uma vez vale lembrar que devemos usar o comando certo no local certo. Isto assegura um código limpo e de fácil entendimento. O comando **switch** é próprio para se testar uma variável em relação a diversos valores pré-estabelecidos. Sua forma geral é:

```
switch (variável)

    {
    case constante_1:
    declaração_1;
    break;
    case constante_2:
    declaração_2;
    break;
    .
    .
    .
    case constante_n:
    declaração_n;
    break;
    default
    declaração_default;
    }
```

Podemos fazer uma analogia entre o **switch** e a estrutura **if-else-if** apresentada anteriormente. A diferença fundamental é que a estrutura **switch** *não* aceita expressões. Aceita apenas constantes. O **switch** testa a variável e executa a declaração cujo **case** corresponda ao valor atual da variável. A declaração **default** é opcional e será executada apenas se a variável, que está sendo testada, não for igual a nenhuma das constantes.

O comando **break**, faz com que o **switch** seja interrompido assim que uma das declarações seja executada. Mas ele não é essencial ao comando **switch**. Se após a execução da declaração não houver um **break**, o programa continuará executando. Isto pode ser útil em algumas situações, mas eu recomendo cuidado. Veremos agora um exemplo do comando **switch**:

```
#include <stdio.h> #include <conio.h>
int main ()
{
       int num;
       printf ("Digite um numero: ");
       scanf ("%d",&num);
       switch (num)
         {
              case 9:
                    printf ("\n\nO numero e igual a 9.\n");
              break;
              case 10:
                    printf ("\n\nO numero e igual a
10.\n");
              break;
              case 11:
                    printf ("\n\nO numero e igual a
11.\n");
              break;
              default:
                    printf ("\n\nO numero nao e nem 9 nem
10 nem 11.\n");
         }
       getch(); return(0);
}
```

O Comando for

for é a primeira de uma série de três estruturas para se trabalhar com loops de repetição. As outras são **while** e **do**. As três compõem a segunda família de comandos de controle de fluxo. Podemos pensar nesta família como sendo a das estruturas de repetição controlada.

Como já foi dito, o loop **for** é usado para repetir um comando, ou bloco de comandos, diversas vezes, de maneira que se possa ter um bom controle sobre o loop. Sua forma geral é:

for (inicialização;condição;incremento) declaração;

O melhor modo de se entender o loop **for** é ver como ele funciona "por dentro". O loop **for** é equivalente a se fazer o seguinte:

inicialização;
if (condição)
{

declaração;
incremento;
"Volte para o comando if"
}

Podemos ver, então, que o **for** executa a inicialização incondicionalmente e testa a condição. Se a condição for falsa ele não faz mais nada. Se a condição for verdadeira ele executa a declaração, faz o incremento e volta a testar a condição. Ele fica repetindo estas operações até que a condição seja falsa. Um ponto importante é que podemos omitir qualquer um dos elementos do **for**, isto é, se não quisermos uma inicialização poderemos omiti-la. Abaixo vemos um programa que coloca os primeiros 100 números inteiros na tela:

```
#include <stdio.h> #include <conio.h>
int main ()
{
    int count;
    for (count=1; count<=100; count++) printf ("%d
",count);
    getch(); return(0);
}
```

Note que, no exemplo acima, há uma diferença em relação ao exemplo anterior. O incremento da variável **count** é feito usando o operador de incremento que nós agora já conhecemos. Esta é a forma usual de se fazer o incremento (ou decremento) em um loop **for**.

O **for** na linguagem C é bastante flexível. Temos acesso à inicialização, à condição e ao incremento. Qualquer uma destas partes do for pode ser uma expressão qualquer do C, desde que ela seja válida. Isto nos permite fazer o que quisermos com o comando. As três formas do for abaixo são válidas:

```
for ( count = 1; count < 100 ; count++) { ... }
for (count = 1; count < NUMERO_DE_ELEMENTOS ; count++) { ... }
for (count = 1; count < BusqueNumeroDeElementos() ; count+=2) {
... }
etc ...
```

Preste atenção ao último exemplo: o incremento está sendo feito de dois em dois. Além disto, no teste está sendo utilizada uma função (BusqueNumeroDeElementos()) que retorna um valor que está sendo comparado com count.

- O loop infinito

O loop infinito tem a forma

for (inicialização; ;incremento) declaração;

Este loop chama-se loop infinito porque será executado para sempre (não existindo a condição, ela será sempre considerada verdadeira), a não ser que ele seja interrompido. Para interromper um loop como este usamos o comando **break**. O comando **break** vai quebrar o loop infinito e o programa continuará sua execução normalmente.

Como exemplo vamos ver um programa que faz a leitura de uma tecla e sua impressão na tela, até que o usuario aperte uma tecla sinalizadora de final (um FLAG). O nosso FLAG será a letra 'X'. Repare que tivemos que usar dois scanf() dentro do for. Um busca o caractere que foi digitado e o outro busca o outro caracter digitado na seqüência, que é o caractere correspondente ao <ENTER>.

```
#include <stdio.h> #include <conio.h>
int main ()
{
   int Count;
   char ch;
   printf(" Digite uma letra - <X para sair> ");
   for (Count=1;;Count++)
   {
     scanf("%c", &ch);
     if (ch == 'X') break;
     printf("\nLetra: %c \n",ch);
     scanf("%c", &ch);
   }
   getch(); return(0);
}
```

- O loop sem conteúdo

Loop sem conteúdo é aquele no qual se omite a declaração. Sua forma geral é (atenção ao ponto e vírgula!):

for (inicialização;condição;incremento);

Uma das aplicações desta estrutura é gerar tempos de espera.

O programa

```
#include <stdio.h> #include <conio.h>
int main ()
{
      long int i;
      printf("\a");                  /* Imprime o caracter
de alerta (um beep) */
      for (i=0; i<10000000; i++);  /* Espera 10.000.000 de
iteracoes */
```

```
      printf("\a");                  /* Imprime outro
caracter de alerta */
      getch(); return(0);
}
```

faz isto.

O Comando while

O comando **while** tem a seguinte forma geral:

while (condição) declaração;

Assim como fizemos para o comando **for**, vamos tentar mostrar como o **while** funciona fazendo uma analogia. Então o **while** seria equivalente a:

if (condição)

{

declaração;
"Volte para o comando if"
}

Podemos ver que a estrutura **while** testa uma condição. Se esta for verdadeira a declaração é executada e faz-se o teste novamente, e assim por diante. Assim como no caso do **for**, podemos fazer um loop infinito. Para tanto basta colocar uma expressão eternamente verdadeira na condição. Pode-se também omitir a declaração e fazer um loop sem conteúdo. Vamos ver um exemplo do uso do **while**. O programa abaixo é executado enquanto i for menor que 100. Veja que ele seria implementado mais naturalmente com um for ...

```
#include <stdio.h> #include <conio.h>
int main ()
{
      int i = 0;
      while ( i < 100)
      {
            printf(" %d", i);
            i++;
            }
      getch(); return(0);
}
```

O programa abaixo espera o usuário digitar a tecla 'q' e só depois finaliza:

```
#include <stdio.h> #include <conio.h>
int main ()
{
       char Ch;
       Ch='\0';
       while (Ch!='q')
        {
              scanf("%c", &Ch);
        }
       getch(); return(0);
}
```

O Comando do-while

A terceira estrutura de repetição que veremos é o **do-while** de forma geral:

do
{
declaração;
} while (condição);

Mesmo que a declaração seja apenas um comando é uma boa prática deixar as chaves. O ponto-e- vírgula final é obrigatório. Vamos, como anteriormente, ver o funcionamento da estrutura **do-while** "por dentro":

declaração;
if (condição) "Volta para a declaração"

Vemos pela análise do bloco acima que a estrutura **do-while** executa a declaração, testa a condição e, se esta for verdadeira, volta para a declaração. A grande novidade no comando **do-while** é que ele, ao contrário do **for** e do **while**, garante que a declaração será executada pelo menos uma vez.

Um dos usos da extrutura **do-while** é em menus, nos quais você quer garantir que o valor digitado pelo usuário seja válido, conforme apresentado abaixo:

```
#include <stdio.h> #include <conio.h>
int main ()
{
       int  i;
       do
         {
              printf ("\n\nEscolha a fruta pelo
numero:\n\n");
              printf ("\t(1)...Mamao\n");
              printf ("\t(2)...Abacaxi\n");
```

```
            printf ("\t(3)...Laranja\n\n");
            scanf("%d", &i);
        } while ((i<1)||(i>3));

      switch (i)
        {
            case 1:
                  printf ("\t\tVoce escolheu Mamao.\n");
            break;
            case 2:
                  printf ("\t\tVoce escolheu
Abacaxi.\n");
            break;
            case 3:
                  printf ("\t\tVoce escolheu
Laranja.\n");
            break;
        }
      getch(); return(0);
}
```

O Comando break

Nós já vimos dois usos para o comando **break:** interrompendo os comandos **switch** e **for**. Na verdade, estes são os dois usos do comando **break**: ele pode quebrar a execução de um comando (como no caso do **switch**) ou interromper a execução de *qualquer* loop (como no caso do **for**, do **while** ou do **do while**). O **break** faz com que a execução do programa continue na primeira linha seguinte ao loop ou bloco que está sendo interrompido.

Observe que um break causará uma saída somente do laço mais interno. Por exemplo:

```
for(t=0; t<100; ++t)
{
      count=1;
      for(;;)
      {
            printf("%d", count);
            count++;
            if(count==10) break;
      }
}
```

O código acima imprimirá os números de 1 a 10 cem vezes na tela. Toda vez que o break é encontrado, o controle é devolvido para o laço for externo.

Outra observaçao é o fato que um break usado dentro de uma declaraçao switch afetará somente os dados relacionados com o switch e nao qualquer outro laço em que o

switch estiver.

O Comando continue

O comando **continue** pode ser visto como sendo o oposto do **break**. Ele só funciona dentro de um loop. Quando o comando **continue** é encontrado, o loop pula para a próxima iteração, sem o abandono do loop, ao contrário do que acontecia no comando **break**. O programa abaixo exemplifica o uso do *continue*:

```
#include <stdio.h> #include <conio.h>
int main()
{
        int opcao;
        while (opcao != 5)
        {
                printf("\n\n Escolha uma opcao entre 1 e 5: ");
                scanf("%d", &opcao);
                if ((opcao > 5)||(opcao <1)) continue;  /* Opcao
invalida: volta ao inicio do loop */
                switch (opcao)
                {
                        case 1:
                                printf("\n --> Primeira opcao..");
                        break;
                        case 2:
                                printf("\n --> Segunda opcao..");
                        break;
                        case 3:
                                printf("\n --> Terceira opcao..");
                        break;
                        case 4:
                                printf("\n --> Quarta opcao..");
                        break;
                        case 5:
                                printf("\n --> Abandonando..");
                        break;
                }
       }
getch(); return(0);
}
```

O programa acima ilustra uma aplicação simples para o *continue*. Ele recebe uma opção do usuario. Se esta opção for inválida, o *continue* faz com que o fluxo seja desviado de volta ao início do loop. Caso a opção escolhida seja válida o programa segue normalmente.

TÓPICO 5 - MATRIZES E STRINGS

Vetores

Vetores nada mais são que matrizes unidimensionais. Vetores são uma estrutura de dados muito utilizada. É importante notar que vetores, matrizes bidimensionais e matrizes de qualquer dimensão são caracterizadas por terem todos os elementos pertencentes ao mesmo tipo de dado. Para se declarar um vetor podemos utilizar a seguinte forma geral:

tipo_da_variável nome_da_variável [tamanho];

Quando o C vê uma declaração como esta ele reserva um espaço na memória suficientemente grande para armazenar o número de células especificadas em tamanho. Por exemplo, se declararmos:

```
float exemplo [20];
```

o C irá reservar 4x20=80 bytes. Estes bytes são reservados de maneira contígua.

Na linguagem C a numeração começa sempre em zero. Isto significa que, no exemplo acima, os dados serão indexados de 0 a 19. Para acessá-los vamos escrever:

```
exemplo[0]
exemplo[1]
 .
 .
 .
exemplo[19]
```

Mas ninguém o impede de escrever:

```
exemplo[30]
exemplo[103]
```

Por quê? Porque o C não verifica se o índice que você usou está dentro dos limites válidos. Este é um cuidado que *você* deve tomar. Se o programador não tiver atenção com os limites de validade para os índices ele corre o risco de ter variáveis sobreescritas ou de ver o computador travar. Bugs terríveis podem surgir. Vamos ver agora um exemplo de utilização de vetores:

```
#include <stdio.h> #include <conio.h>
int main ()
{
        int num[100];  /* Declara um vetor de inteiros de 100 posicoes
*/
        int count=0;
        int totalnums;
        do
        {
                printf ("\nEntre com um numero (-999 p/ terminar): ");
                scanf ("%d",&num[count]);
                count++;
        } while (num[count-1]!=-999);
        totalnums=count-1;
        printf ("\n\n\n\t Os números que você digitou foram:\n\n");
        for (count=0;count<totalnums;count++)
                printf (" %d",num[count]);
        getch(); return(0);
}
```

No exemplo acima, o inteiro *count* é inicializado em 0. O programa pede pela entrada de números até que o usuário entre com o Flag -999. Os números são armazenados no vetor **num**. A cada número armazenado, o contador do vetor é incrementado para na próxima iteração escrever na próxima posição do vetor. Quando o usuário digita o flag, o programa abandona o primeiro loop e armazena o total de números gravados. Por fim, todos os números são impressos. É bom lembrar aqui que nenhuma restrição é feita quanto a quantidade de números digitados. Se o usuário digitar mais de 100 números, o programa tentará ler normalmente, mas o programa os escreverá em uma parte não alocada de memória, pois o espaço alocado foi para somente 100 inteiros. Isto pode resultar nos mais variados erros no instante da execução do programa.

Strings

Strings são vetores de **chars**. Nada mais e nada menos. As strings são o uso mais comum para os vetores. Devemos apenas ficar atentos para o fato de que as strings têm o seu último elemento como um '\0'. A declaração geral para uma string é:

char nome_da_string [tamanho];

Devemos lembrar que o tamanho da string deve incluir o '\0' final. A biblioteca padrão do C possui diversas funções que manipulam strings. Estas funções são úteis pois não se pode, por exemplo, igualar duas strings:

```
string1=string2;          /* NAO faca isto */
```

Fazer isto é um desastre. Quando você terminar de ler a seção que trata de ponteiros você entenderá porquê. As strings devem ser igualadas elemento a elemento.

Quando vamos fazer programas que tratam de string muitas vezes podemos fazer bom proveito do fato de que uma string termina com '\0' (isto é, o número inteiro 0). Veja, por exemplo, o programa abaixo que serve para igualar duas strings (isto é, copia os caracteres de uma string para o vetor da outra) :

```
#include <stdio.h> #include <conio.h>
int main ()
{
       int count;
       char str1[100],str2[100];
       ....            /* Aqui o programa le str1 que sera
copiada para str2 */
       for (count=0;str1[count];count++)
              str2[count]=str1[count];
         str2[count]='\0';
....            /* Aqui o programa continua */
}
```

A condição no loop **for** acima é baseada no fato de que a string que está sendo copiada termina em '\0'. Quando o elemento encontrado em **str1[count]** é o '\0', o valor retornado para o teste condicional é falso (nulo). Desta forma a expressão que vinha sendo verdadeira (não zero) continuamente, torna-se falsa.

Vamos ver agora algumas funções básicas para manipulação de strings.

- gets

A função **gets()** lê uma string do teclado. Sua forma geral é:

gets (nome_da_string);

O programa abaixo demonstra o funcionamento da função **gets()**:

```
#include <stdio.h> #include <conio.h>
int main ()
{
       char string[100];
       printf ("Digite o seu nome: ");
       gets (string);
       printf ("\n\n Ola %s",string);
       getch(); return(0);
}
```

Repare que é válido passar para a função **printf()** o nome da string. Você verá mais adiante porque isto é válido. Como o primeiro argumento da função **printf()** é uma string também é válido fazer:

```
printf (string);
```

isto simplesmente imprimirá a string.

- strcpy

Sua forma geral é:

strcpy (string_destino,string_origem);

A função **strcpy()** copia a string-origem para a string- destino. Seu funcionamento é semelhante ao da rotina apresentada na seção anterior. As funções apresentadas nestas seções estão no arquivo cabeçalho **string.h**. A seguir apresentamos um exemplo de uso da função **strcpy()**:

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

```
#include <string.h>
int main ()
{
        char str1[100],str2[100],str3[100];
        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(); return(0);
}
```

- strcat

A função **strcat()** tem a seguinte forma geral:

strcat (string_destino,string_origem);

A string de origem permanecerá inalterada e será anexada ao fim da string de destino. Um exemplo:

```
#include <stdio.h>
#include <string.h>
int main ()
{
      char str1[100],str2[100];
      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(); return(0);
}
```

- strlen

Sua forma geral é:

strlen (string);

A função **strlen()** retorna o comprimento da string fornecida. O terminador nulo não é contado. Isto quer dizer que, de fato, o comprimento do vetor da string deve ser um a mais que o inteiro retornado por **strlen()**.

Um exemplo do seu uso:

```
#include <stdio.h> #include <conio.h>
#include <string.h>
int main ()
{
       int size;
       char str[100];
       printf ("Entre com uma string: ");
       gets (str);
       size=strlen (str);
       printf ("\n\nA string que voce digitou tem tamanho
%d",size);
       getch(); return(0);
}
```

- strcmp

Sua forma geral é:

strcmp (string1,string2);

A função **strcmp()** compara a string 1 com a string 2. Se as duas forem idênticas a função retorna zero. Se elas forem diferentes a função retorna não-zero. Um exemplo da sua utilização:

```
#include <stdio.h> #include <conio.h>
#include <string.h>
int main ()
{
       char str1[100],str2[100];
       printf ("Entre com uma string: ");
       gets (str1);
       printf ("\n\nEntre com outra string: ");
       gets (str2);
       if (strcmp(str1,str2))
              printf ("\n\nAs duas strings são
diferentes.");
       else printf ("\n\nAs duas strings são iguais.");
       getch(); return(0);
}
```

Matrizes

- Matrizes bidimensionais

Já vimos como declarar matrizes unidimensionais (vetores). Vamos tratar agora de matrizes bidimensionais. A forma geral da declaração de uma matriz bidimensional é muito parecida com a declaração de um vetor:

tipo_da_variável nome_da_variável [altura][largura];

É muito importante ressaltar que, nesta estrutura, o índice da esquerda indexa as linhas e o da direita indexa as colunas. Quando vamos preencher ou ler uma matriz no C o índice mais à direita varia mais rapidamente que o índice à esquerda. Mais uma vez é bom lembrar que, na linguagem C, os índices variam de zero ao valor declarado, menos um; mas o C não vai verificar isto para o usuário. Manter os índices na faixa permitida é tarefa do programador. Abaixo damos um exemplo do uso de uma matriz:

```
#include <stdio.h> #include <conio.h>
int main ()
{
int mtrx [20][10];
int i,j,count;
count=1;
for (i=0;i<20;i++)
       for (j=0;j<10;j++)
       {
              mtrx[i][j]=count;
                 count++;
         }
getch(); return(0);
}
```

No exemplo acima, a matriz **mtrx** é preenchida, sequencialmente por linhas, com os números de 1 a 200. Você deve entender o funcionamento do programa acima antes de prosseguir.

- Matrizes de strings

Matrizes de strings são matrizes bidimensionais. Imagine uma string. Ela é um vetor. Se fizermos um vetor de strings estaremos fazendo uma lista de vetores. Esta estrutura é uma matriz bidimensional de **chars**. Podemos ver a forma geral de uma matriz de strings como sendo:

char nome_da_variável [num_de_strings][compr_das_strings];

Aí surge a pergunta: como acessar uma string individual? Fácil. É só usar apenas o primeiro índice. Então, para acessar uma determinada string faça:

nome_da_variável [índice]

Aqui está um exemplo de um programa que lê 5 strings e as exibe na tela:

```
#include <stdio.h> #include <conio.h>
int main ()
{
       char strings [5][100];
       int count;
       for (count=0;count<5;count++)
         {
              printf ("\n\nDigite uma string: ");
              gets (strings[count]);
         }
       printf ("\n\n\nAs strings que voce digitou
foram:\n\n");
       for (count=0;count<5;count++)
                 printf ("%s\n",strings[count]);
         getch(); return(0);
}
```

- Matrizes multidimensionais

O uso de matrizes multidimensionais na linguagem C é simples. Sua forma geral é:

tipo_da_variável nome_da_variável [tam1][tam2] ... [tamN];

Uma matriz N-dimensional funciona basicamente como outros tipos de matrizes. Basta lembrar que o índice que varia mais rapidamente é o índice mais à direita.

- Inicialização

Podemos inicializar matrizes, assim como podemos inicializar variáveis. A forma geral de uma matriz como inicialização é:

tipo_da_variável nome_da_variável [tam1][tam2] ... [tamN] = {lista_de_valores};

A lista de valores é composta por valores (do mesmo tipo da variável) separados por vírgula. Os valores devem ser dados na ordem em que serão colocados na matriz. Abaixo vemos alguns exemplos de inicializações de matrizes:

```
float vect [6] = { 1.3, 4.5, 2.7, 4.1, 0.0, 100.1 };
int matrx [3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
char str [10] = { 'J', 'o', 'a', 'o', '\0' };
char str [10] = "Joao";
char str_vect [3][10] = { "Joao", "Maria", "Jose" };
```

O primeiro demonstra inicialização de vetores. O segundo exemplo demonstra a inicialização de matrizes multidimensionais, onde **matrx** está sendo inicializada com 1, 2, 3 e 4 em sua primeira linha, 5, 6, 7 e 8 na segunda linha e 9, 10, 11 e 12 na última linha. No terceiro exemplo vemos como inicializar uma string e, no quarto exemplo, um modo mais compacto de inicializar uma string. O quinto exemplo combina as duas técnicas para inicializar um vetor de strings. Repare que devemos incluir o ; no final da inicialização.

- Inicialização sem especificação de tamanho

Podemos, em alguns casos, inicializar matrizes das quais não sabemos o tamanho *a priori*. O compilador C vai, neste caso verificar o tamanho do que você declarou e considerar como sendo o tamanho da matriz. Isto ocorre na hora da compilação e não poderá mais ser mudado durante o programa, sendo muito útil, por exemplo, quando vamos inicializar uma string e não queremos contar quantos caracteres serão necessários. Alguns exemplos:

```
        char mess [] = "Linguagem C: flexibilidade e
poder.";
        int matrx [][2] = { 1,2,2,4,3,6,4,8,5,10 };
```

No primeiro exemplo, a string mess terá tamanho 36. Repare que o artifício para realizar a inicialização sem especificação de tamanho é não especificar o tamanho! No segundo exemplo o valor não especificado será 5.

TÓPICO 6 – ARQUIVOS

Abrindo e Fechando um Arquivo

No sistema de entrada e saída ANSI é definido o tipo "ponteiro de arquivo". Este não é um tipo propriamente dito, mas uma definição usando o comando typedef. Esta definição está no arquivo cabeçalho **stdio.h** ou **stdlib.h** dependendo do seu compilador. Podemos declarar um ponteiro de arquivo da seguinte maneira:

```
FILE *p;
```

onde **p** será então um ponteiro para um arquivo. É usando este tipo de ponteiro que vamos poder manipular arquivos no C.

fopen

Esta é a função de abertura de arquivos. Seu protótipo é:

```
FILE *fopen (char *nome_do_arquivo,char *modo);
```

O nome_do_arquivo determina qual arquivo deverá ser aberto. Este nome deve ser válido no sistema operacional que estiver sendo utilizado. O modo de abertura diz à função **fopen()** que tipo de uso você vai fazer do arquivo. A tabela abaixo mostra os valores de modo válidos:

Modo	Significado
"r"	Abre um arquivo para leitura
"w"	Cria um arquivo para escrita
"a"	Acrescenta dados no fim do arquivo ("append")
"rb"	Abre um arquivo binário para leitura
"wb"	Cria um arquivo binário para escrita
"ab"	Acrescenta dados binários no fim do arquivo
"r+"	Abre um arquivo para leitura e escrita
"w+"	Cria um arquivo para leitura e escrita
"a+"	Acrescenta dados ou cria uma arquivo para leitura e escrita
"r+b"	Abre um arquivo binário para leitura e escrita
"w+b"	Cria um arquivo binário para leitura e escrita
"a+b"	Acrescenta dados ou cria uma arquivo binário para leitura e escrita
"rt"	Abre um arquivo texto para leitura
"wt"	Cria um arquivo texto para escrita
"at"	Acrescenta dados no fim do arquivo texto
"r+t"	Abre um arquivo texto para leitura e escrita
"w+t"	Cria um arquivo texto para leitura e escrita
"a+t"	Acrescenta dados ou cria uma arquivo texto para leitura e escrita

Poderíamos então, para abrir um arquivo binário, escrever:

```
FILE *fp;
fp=fopen ("exemplo.bin","wb");
if (!fp)
        printf ("Erro na abertura do arquivo.");
```

A condição **!fp** testa se o arquivo foi aberto com sucesso porque no caso de um erro a função **fopen()** retorna um ponteiro nullo (**NULL**). Veja o exemplo a seguir:

```
#include <stdio.h
main (void)
{
FILE *fp;
...
fp=fopen ("exemplo.bin","wb");
if (!fp)
        {
        printf ("Erro na abertura do arquivo. Fim de programa.");
        exit (1);
        }
...
return 0;
}
```

exit

Aqui abrimos um parênteses para explicar a função **exit()** cujo protótipo é:

```
void exit (int codigo_de_retorno);
```

Esta função aborta a execução do programa. Pode ser chamada de qualquer ponto no programa e faz com que o programa termine e retorne, para o sistema operacional, o código_de_retorno. A convenção mais usada é que um programa retorne zero no caso de um término normal e retorne um número não nulo no caso de ter ocorrido um problema. A função **exit()** se torna importante em casos como alocação dinâmica e abertura de arquivos pois pode ser essencial que uma determinada memória seja alocada ou que um arquivo seja aberto.

fclose

Quando abrimos um arquivo devemos fechá-lo. Para tanto devemos usar a função **fclose()**:

```
int fclose (FILE *fp);
```

É importante que se perceba que se deve tomar o maior cuidado para não se "perder" o ponteiro do arquivo. "Perder" neste caso seria se atribuir um valor de um outro ponteiro qualquer ao ponteiro de arquivo (perdendo assim o valor original). É utilizando este ponteiro que vamos poder trabalhar com o arquivo.

Se perdermos o ponteiro de um determinado arquivo não poderemos nem fechá-lo. O ponteiro **fp** passado à função **fclose()** determina o arquivo a ser fechado. A função retorna zero no caso de sucesso.

Lendo e Escrevendo Caracteres em Arquivos

fputc (Escreve um caracter no arquivo)
Toda vez que estamos trabalhando com arquivos, há uma espécie de posição atual no arquivo. Esta posição, gerenciada pelo compilador, é a posição de onde será lido ou escrito o próximo caracter. Normalmente, num acesso seqüencial a um arquivo, não temos que mexer nesta posição pois quando lemos um caractere a posição no arquivo é automaticamente atualizada. Num acesso randômico teremos que mexer nesta posição (ver fseek()). Protótipo:

```
int fputc (int ch,FILE *fp);
```

fgetc

Retorna um caracter lido do arquivo. Protótipo:

```
int fgetc (FILE *fp);
```

feof

EOF ("End of file") indica o fim de um arquivo. Às vezes, é necessário verificar se um arquivo chegou ao fim. Para isto podemos usar a função **feof()**. Ela retorna não-zero se o arquivo chegou ao EOF, caso contrário retorna zero. Seu protótipo é:

```
int feof (FILE *fp);
```

Outros Comandos de Acesso a Arquivos

ferror

Protótipo:

```
int ferror (FILE *fp);
```

A função **ferror()** se torna muito útil quando queremos verificar se cada acesso a um arquivo teve sucesso. Cada vez que uma função de arquivo é executada, a própria função registra numa variável especial se houve sucesso na operação ou não. Com **ferror()** podemos ter acesso ao conteúdo desta variável: ela retorna não zero se houve algum erro na última função de acesso ao arquivo.

rewind

A função **rewind()** de protótipo

```
void rewind (FILE *fp);
```

retorna a posição corrente do arquivo para o início.

getw() lê no arquivo um inteiro. Seu protótipo é:

```
int getw (FILE *fp);
```

putw() escreve um inteiro num arquivo. Protótipo:

```
int putw (int i,FILE *fp);
```

fgets

Para se ler uma string num arquivo podemos usar **fgets()** cujo protótipo é:

```
char *fgets (char *str,int tamanho,FILE *fp);
```

A string str lida deve ter seu tamanho determinado pelo programador.

fputs

Protótipo:

```
char *fputs (char *str,FILE *fp);
```

Escreve uma string num arquivo.

fread

Podemos escrever e ler blocos de dados. Para tanto temos as funções **fread()** e **fwrite()**. O protótipo de **fread()** é:

```
unsigned fread (void *buffer,int numero_de_bytes,int count,FILE *fp);
```

O buffer é a região de memória na qual serão armazenados os dados lidos. O número de bytes é o tamanho da unidade a ser lida. O contador indica quantas unidades devem ser lidas. Isto significa que o número total de bytes lidos é:

*numero_de_bytes*count*

A função retorna o número de unidades efetivamente lidas. Este número pode ser menor que o valor requisitado quando o fim do arquivo for encontrado.

fwrite

A função **fwrite()** funciona como a sua companheira **fread()**. Seu protótipo é:

```
unsigned fwrite(void *buffer,int numero_de_bytes,int count,FILE *fp);
```

fseek

Para se fazer procuras e acessos randômico usa-se a função **fseek()**. Esta move o cursor no arquivo de um valor especificado, a partir de um ponto especificado. Seu protótipo é:

```
int fseek (FILE *fp,long numbytes,int origem);
```

O parâmetro de origem determina a partir de onde é que os bytes de movimentação serão contados. Os valores possíveis são definidos por macros no arquivo cabeçalho e são:

Nome	Valor	Significado
SEEK_SET	0	Início do arquivo
SEEK_CUR	1	Ponto corrente no arquivo
SEEK_END	2	Fim do arquivo

Tendo-se definido a partir de onde irá se contar numbytes determina quantos bytes de deslocamento será dado na posição atual.

remove

Protótipo:

```
int remove (char *nome_do_arquivo);
```

Desmancha um arquivo especificado.

Fluxos Padrão

Os fluxos padrão em arquivos permitem ao programador ler e escrever em arquivos da maneira padrão com a qual o programador já lia e escrevia na tela.

fprintf

A função **fprintf()** funciona como a função **printf()**. A diferença é que a saída de **fprintf()** é um arquivo e não a tela do computador.
Protótipo:

```
int fprintf (FILE *fp,char *str,...);
```

Como já poderíamos esperar, a única diferença do protótipo de **fprintf()** para o de **printf()** é a especificação do arquivo destino através do ponteiro de arquivo.

fscanf

A função **fscanf()** funciona como a função **scanf()**. A diferença é que **fscanf()** lê de um arquivo e não da tela do computador.
Protótipo:

```
int fscanf (FILE *fp,char *str,...);
```

Como já poderíamos esperar, a única diferença do protótipo de **fscanf()** para o de **scanf()** é a especificação do arquivo destino através do ponteiro de arquivo.

TÓPICO 7 – EXERCÍCIOS RESOLVIDOS

1)Receber um nome e imprimir as 4 primeiras letras do nome.

```
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <stdio.h>
main()
{
	char nome[30];
	int B;
	printf ("informe um nome:");
	gets(nome);
	for(B=0;B<=3;B++)
	printf("\%c",nome[B]);
	printf ("\n\n");
	system("PAUSE");
	return 0;
}
```

2)Receber um nome e imprimir as letras na posição impar

```
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <stdio.h>
main()
{
	int pp=1,tam,x;
	char APELIDO[100];
	printf ("Informe um apelido:");
	gets(APELIDO);
	tam=strlen(APELIDO);
	printf("\nAs letras na posicao impar sao: ",x);
	while(pp<=tam-1)
	{
		printf(" %c ",APELIDO[pp]);
		pp=pp+2;
	}
	printf ("\n");
	printf("\n\tNome digitado: %s\t",APELIDO);
	printf ("\n\n");
	system("PAUSE");
	return 0;
}
```

3)Receber um nome e imprimir as letras na posição impar

```
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <stdio.h>
main()
{
	int pp,tam,x;
	char APELIDO[100];
	printf ("[%c]Informe um apelido: ");
	gets(APELIDO);
```

```
	tam=strlen(APELIDO);
	printf("\nAs letras na posicao impar sao:");
	for(pp=1;
	pp<=tam-1;pp=pp+2)
	printf(" %c ",APELIDO[pp]);
	printf("\n\nNome digitado: %s\t",APELIDO);
	printf ("\n\n");
	system("PAUSE");
	return 0;
}
```

4)Ler nome, endereço, telefone e imprimir

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
int main()
{
	char nome[30], endereco[30], telefone[15];
	printf("Informe seu nome: ");
	gets(nome);
	printf("Informe seu endereco: ");
	gets(endereco);
	printf("Informe seu telefone: ");
	gets(telefone);
	printf("\n\nNome: %s\n\n",nome);
	printf("Endereco: %s\n\n",endereco);
	printf("Telefone: %s\n\n",telefone);
	system("PAUSE");
	return 0;
}
```

5) Ler nome, sexo e idade. Se sexo for feminino e idade menor que 25. Imprimir o nome da pessoa e a palavra ACEITA. Caso contrario imprimir NAO ACEITA.

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
main()
{
	char nome[30], sexo;
	int idade;
	printf("Informe seu nome: ");
	gets(nome);
	printf("Informe seu sexo: ");
	scanf("%c",&sexo);
	printf("Informe sua idade: ");
	scanf("%d",&idade);
	if (sexo == 'f' || sexo == 'F' && idade < 25)
	printf("\n%s. ACEITA.\n\n", nome);
	else
	printf("\nNAO ACEITA.\n\n");
	system("PAUSE");
	return 0;
}
```

6) Digite um nome e mostre quantas letras tem.

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

```
#include <string.h>
main()
{
	int x,tam;
	char nome[30];
	for (x=1; x <= 4; x++)
	{
	printf("Digite um nome: ");
	gets(nome);
	// na variavel tam ficará guardado quantas letras tem o nome
	tam = strlen(nome);
	printf("\nEsse nome tem %d\ letras.\n\n",tam);
	}
	printf("\n\n");
	system("pause");
	return 0;
}
```

7) Receber do teclado um nome e imprimir tantas vezes quantos forem seus caracteres.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <string.h>
main()
{
	int x,tam;
	char nome[30];
	printf("Digite um nome: ");
	gets(nome);
	tam = strlen(nome);
	for (x=1; x <= tam; x++)
	printf("\n%s",nome);
	printf("\n\n");
	system("pause");
	return 0;
}
```

8) Receber do teclado uma mensagem e imprimir quantas letras A, E, I, O, U tem esta mensagem. Considerar minúscula e maiúscula.

Exemplo:

curso = "curso de redes"

Imprimir strelem(curso,3) ==> irá imprimir a letra s, pois a posição da primeira letra da palavra curso é 0. a segunda é 1, a terceira é 2 e assim sucessivamente.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <string.h>
main()
{
	int x,tam;
	int ca,ce,ci,co,cu;
	char nome[30];
	// inicializei todas as variaveis com zero porque sao contadores
	ca = ce = ci = co = cu = 0;
	printf("Digite uma frase: ");
	gets(nome);
	tam = strlen(nome);
	for (x=1; x <= tam-1; x++)
```

```
        {
        if (nome[x] == 'a' || nome[x] == 'A')
        ca++;
        else if (nome[x] == 'e' || nome[x] == 'E')
        ce++;
        else if (nome[x] == 'i' || nome[x] == 'I')
        ci++;
        else if (nome[x] == 'o' || nome[x] == 'O')
        co++;
        else if (nome[x] == 'u' || nome[x] == 'U')
        cu++;
        }
        printf("\n\nA frase tem:\n");
        printf("\n%d letra a",ca);
        printf("\n%d letra e",ce);
        printf("\n%d letra i",ci);
        printf("\n%d letra o",co);
        printf("\n%d letra u",cu);
        printf("\n\n");
        system("pause");
        return 0;
}
```

9) Criar um algoritmo que entre com uma palavra e imprima conforme exemplo a seguir:
Exemplo: SONHO
Como a palavra SONHO tem 5 letras a impressão ficaria assim:
SONHO
SONHO SONHO
SONHO SONHO SONHO
SONHO SONHO SONHO SONHO
SONHO SONHO SONHO SONHO SONHO
Repare que foram impressos 5 vezes na horizontal e 5 na vertical.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <string.h>
main()
{
        int x,y,tam;
        char nome[30];
        printf("Digite uma palavra: ");
        gets(nome);
        tam = strlen(nome);
        for (x=1; x <= tam; x++)
        {
        // o limite superior da repetição é o valor de x do primeiro for
        // y <= x
        for (y=1;y<=x;y++)
        printf("%s\t",nome);
        printf("\n");
        }
        system("pause");
        return 0;
}
```

10) Receber um nome do teclado e imprimí-lo de trás pra frente.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <string.h>
main()
{
	int x,y,tam;
	char nome[30];
	printf("Digite uma palavra: ");
	gets(nome);
	tam = strlen(nome);
	printf("\n A palavra de tras pra frente e: ");
	for (x=tam-1; x >= 0; x--)
	printf("%c",nome[x]);
	printf("\n\n");
	system("pause");
	return 0;
}
```

11) Receber do teclado a sigla do estado de uma pessoa e imprimir uma das seguintes mensagens:
Carioca
Paulista
Mineiro
Outros estados

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <string.h>
// neste programa tem que usar o arquivo de include string.h por causa da função
// strcmp, cuja função é comparar duas strings.
main()
{
	char estado[3];
	printf("Informe a sigla de um estado do Brasil: ");
	// gets(sigla);
	scanf("%s", estado);
	if(!strcmp(estado,"MG") || !strcmp(estado,"mg"))
	printf("Mineiro\n");
	else if(!strcmp(estado,"RJ") || !strcmp(estado,"rj"))
	printf("Carioca\n");
	else if(!strcmp(estado,"SP") || !strcmp(estado,"sp"))
	printf("Paulista\n");
	else printf("Outros estados\n");
	printf("\n");
	system("pause");
	return 0;
}
```

12)Entrar com um nome e imprimir o nome somente se a primeira letra do nome for “a” (maiúscula ou minúscula).

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
main()
	{
```

```
	char NOME[30];
	printf("\nINFORME UM NOME: ");
	gets(NOME);
	if(NOME[0]=='A'||NOME[0]=='a')
	printf("\n%s",NOME);
	printf("\n\n");
	system("pause");
	return(0);
}
```

13) Escrever um programa que receba um nome -Que conte o número de vogais existentes nele. – O programa deverá imprimir o numero total de caracteres do nome -Quantas vogais - E a respectiva porcentagem das vogais em relação ao total de caracteres.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main()
{
	char nome[20],M;
	int x,t,soma=0;
	float percent, t2=0;
	printf("Digite um nome: ");
	gets(nome);
	t=strlen(nome);
	for(x=0;x<=t;x++)
	if(nome[x]=='a' || nome[x]=='A' || nome[x]=='e' || nome[x]=='E' ||
	nome[x]=='i' || nome[x]=='I' || nome[x]=='o' || nome[x]=='O' ||
	nome[x]=='u' || nome[x]=='U')
	soma++;
	printf("\n O nome tem %d caracteres: ",t);
	printf("\n O nome tem %d vogais: ",soma);
	//Este for vai calcular o percentual de vogais no nome.
	for(x=0;x<=t2;x++)
	{
	t2=strlen(nome);
	percent=soma*100/t2;
	}
	printf("\n %f porcento do nome %s sao vogais: ",percent, nome);
	printf("\n\n");
	system("pause");
	return(0);
}
```

14)Receber um nome e imprimir as 4 primeiras letras do nome.

```
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <stdio.h>
main()
{
	char nome[30];
	int B;
	printf ("informe um nome:");
	gets(nome);
	for(B=0;B<=3;B++)
	printf("\%c",nome[B]);
	printf ("\n\n");
	system("PAUSE");
	return (0);
}
```

15) Fazer um programa que tenha a seguinte saida, conforme o tamanho da palavra escrita.

Saida na tela.

//xxx

//xxx xxx

//xxx xxx xxx

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<conio.h>
#include<string.h>
main()
{
	int x,y,tam;
	char palavra[30];
	printf("\nDigite uma palavra: ");
	gets(palavra);
	tam = strlen(palavra);
	for(x=1; x<=tam; x++)
	{
	for(y=1;y<=x;y++)
	printf("%s\t",palavra);
	printf("\n");
	}
	system("pause");
	return(0);
}
```

16) Digitar um nome e solicitar que seja mostrado em maiúsculo na tela.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
main()
{
	int x,tam;
	float M;
	char nome[30];
	printf("\nDigite um nome: ");
	gets(nome);
	tam = strlen(nome);
	for(x=0; x<=tam-1; x++)
	{
		printf("%c\a",toupper(nome[x]));
	}
	printf("\n");
	system("pause");
	return(0);
}
```

17) Solicitar dois nomes e escrevê-los, mostrar a posição de cada letra.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main()
{
	char nome[30], nome2[30]; int x,t,t2;
	printf("\n\n DIGITE UM NOME: ");
	gets(nome);
	t=strlen(nome);
	printf("\n\n DIGITE UM NOME: ");
	gets(nome2);
	t2=strlen(nome2);
	printf("\n\n");
	//Este for vai mostrar a posição de cada letra do primeiro nome.
	for(x=0;x<=t-1;x++)
	{
	printf("\t[%d]",x=x+0);
	}
	printf("\n\n");
	//Este for vai mostrar cada letra abaixo da sua posição no primeiro nome.
	for(x=0;x<=t-1;x++)
	{
	printf("\t %c",nome[x]);
	}
	printf("\n\n");
	//Este for vai mostrar a posição de cada letra do segundo nome.
	for(x=0;x<=t2-1;x++)
	{
	printf("\t[%d]",x=x+0);
	}
	printf("\n\n");
	//Este for vai mostrar cada letra abaixo da sua posição no segundo nome.
	for(x=0;x<=t2-1;x++)
	{
	printf("\t %c",nome2[x]);
	}
	printf("\n\n");
	system("pause");
	return(0);
}
```

Usando Números, calculando médias, etc...

1) **Fazer um programa que imprima a média aritmética dos números 8,9 e 7. A media dos numeros 4, 5 e 6. A soma das duas médias. A media das medias.**

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>
#include<string.h>
main()
{
	float n1=8, n2=9, n3=7, n4=4, n5=5,n6=6, somam, media3;
	printf("\n\n A media dos numeros 8, 9 e 7 e = %2.2f\n\n",float((n1+n2+n3))/3 );
	printf("\n\n A media dos numeros 4, 5 e 6 e = %2.2f\n\n",float((n4+n5+n6))/3 );
	somam=((n1+n2+n3)/3)+((n4+n5+n6)/3);
	printf("\n\n A soma das duas medias e = %2.2f\n\n",somam );
	media3=(((n1+n2+n3)/3)+((n4+n5+n6)/3))/2;
	printf("\n\n A media das medias e = %2.2f\n\n",media3);
```

```
	printf("\n\n");
	system("pause");
	return (0);
}
```

2) Ler um número inteiro e imprimir seu sucessor e seu antecessor.

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>
#include<string.h>
main()
{
int x, n1, n2;
printf("\n\n Digite um numero: ");
scanf("%d",&x);
n1=x+1;
n2=x-1;
printf("\n\nSeu sucessor e : %d",n1);
printf("\n\nSeu antecessor e : %d",n2);
printf("\n\n");
system("pause");
return (0);
}
```

3) Receber um valor qualquer do teclado e imprimir esse valor com reajuste de 10%..

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>
#include<string.h>
main()
{
	float va;
	printf("\n\tDigite um numero: ");
	scanf("%f",&va);
	printf("\n\tValor reajustado em 10%% e: %2.2f\n",va*110/100);
	printf("\n\n");
	system("pause");
	return 0;
}
```

4) Informar tres numeros inteiros e imprimir a média

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
int main()
{
	int a,b,c;
	printf("Informe um numero inteiro: ");
	scanf("%d",&a);
	printf("Informe um numero inteiro: ");
	scanf("%d",&b);
	printf("Informe um numero inteiro: ");
	scanf("%d",&c);
	printf("A media dos tres numeros informados e: %4.2f\n\n",float((a+b+c))/3);
	system("PAUSE");
	return 0;
}
```

5) Ler um número inteiro e imprimir seu quadrado.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
int main()
{
	float a;
	printf("Informe um numero inteiro: ");
	scanf("%f",&a);
	printf("O quadrado do numero informado e: %3.0f\n\n",pow(a,2));
	// para usar a potencia, usa-se pow(numero, potencia)
	system("PAUSE");
	return 0;
}
```

6) Informar um saldo e imprimir o saldo com reajuste de 1%

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
int main()
{
	float saldo;
	printf("Informe o valor do saldo: ");
	scanf("%f",&saldo);
	printf("O saldo corrigido e %4.2f\n\n",saldo * 1.01);
	system("PAUSE");
	return 0;
}
```

7) Calcule e imprima o valor em reais de cada kw o valor em reais a ser pago o novo valor a ser pago por essa residencia com um desconto de 10%. Dado: 100 kilowatts custa 1/7 do salario minimo. quantidade de kw gasto por residencia

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
int main()
{
	float SM, kwgasto, umkw;
	printf("Informe o valor do salario minimo: ");
	scanf("%f",&SM);
	printf("\n\nInforme total Kw gasto na residencia: ");
	scanf("%f",&kwgasto);
	umkw = SM/7/100;
	printf("\n\nO valor de 1 Kw e: %3.2f\n\n",umkw);
	printf("\nO valor a ser pago pela residencia e: %4.2f",kwgasto * umkw);
	printf("\n\nNovo valor a ser pago com desconto de 10%% e: %3.2f\n\n",(kwgasto * umkw) *
	0.90);
	system("PAUSE");
	return 0;
}
```

8) Informar um preço de um produto e calcular novo preço com desconto de 9%

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
int main()
{
	float precoatual;
	printf("Informe o preco atual do produto: ");
	scanf("%f",&precoatual);
	printf("\n\nNovo preco com desconto de 9%% e: %3.2f\n\n",precoatual * 0.91);
	system("PAUSE");
	return 0;
}
```

9) Cálculo de um salario líquido de um professor . Serão fornecidos valor da hora aula, numero de aulas dadas e o % de desconto do INSS.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
int main()
{
	float vha,nad,inss, salario, desconto;
	printf("Informe o valor da hora aula: ");
	scanf("%f",&vha);
	printf("Informe o numero de aulas dadas: ");
	scanf("%f",&nad);
	printf("Informe o valor do percentual de desconto do INSS: ");
	scanf("%f",&inss);
	salario = vha * nad;
	desconto = salario * inss /100;
	printf("\n\nSalario liquido e: %3.2f\n\n",salario - desconto);
	system("PAUSE");
	return 0;
}
```

10) Ler uma temperatura em graus Celsius e transformá-la em graus Fahrenheit.

Formula: F = (9*C+160)/5

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
main()
{
	int tgc;
	printf("Informe temperatura em graus Celsius: ");
	scanf("%d",&tgc);
	printf("\n%d graus Celsius corresponde a %3.2f graus Farenheit: \n\n", tgc, float(9*tgc+160)/5);
	system("PAUSE");
	return 0;
}
```

11) Ler um numero e se for maior que 20 imprimir a metade desse numero.

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
int main()
{
	float numero;
```

```
	printf("Informe um numero: ");
	scanf("%f",&numero);
	if (numero > 20)
	printf("A metade desse numero e %3.2f", numero/2);
	system("PAUSE");
	return 0;
}
```

12) Ler 2 numeros inteiros e soma-los. Se a soma for maior que 10, mostrar o resultado da soma.

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
int main()
{
	float numero1, numero2;
	printf("Informe o primeiro numero: ");
	scanf("%f",&numero1);
	printf("Informe o segundo numero: ");
	scanf("%f",&numero2);
	if ((numero1 + numero2) > 10)
	printf("\nA soma dos numeros informados e %3.2f\n\n", numero1 + numero2);
	system("PAUSE");
	return 0;
}
```

13) Ler 1 número. Se positivo, imprimir raiz quadrada senao o quadrado.

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
int main()
{
	float numero1;
	printf("Informe um numero: ");
	scanf("%f",&numero1);
	if (numero1 > 0)
	printf("\nA raiz quadrado do numero e %3.2f\n\n", sqrt(numero1));
	else
	printf("\nO quadrado do numero e %3.2f\n\n", pow(numero1,2));
	system("PAUSE");
	return 0;
}
```

14) Solicitar salario, prestaçao. Se prestação for maior que 20% do salario, imprimir : Empréstimo não pode ser concedido. Senão imprimir Empréstimo pode ser concedido.

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
int main()
{
	float salbruto, prestacao, porcent;
	printf("Informe o salario bruto: ");
	scanf("%f",&salbruto);
	printf("Informe o valor da prestacao: ");
	scanf("%f",&prestacao);
	porcent = salbruto * 0.20;
	if (prestacao > porcent)
	printf("\nEmprestimo não pode ser concedito.n\n");
```

```
        else
        printf("\nEmprestimo pode ser concedido");
        system("PAUSE");
        return 0;
}
```

15) Ler um numero e imprimir: maior que 20, igual a 20 ou menor que 20.

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
int main()
{
        float numero;
        printf("Informe um numero: ");
        scanf("%f",&numero);
        if (numero > 20)
        printf("\nNumero informado e maior a 20.\n");
        else
        if (numero = 20)
        printf("\nNumero informado e igual a 20.\n");
        else
        printf("\nNumero informado e menor que 20.\n\n");
        system("PAUSE");
        return 0;
}
```

16) Ler um ano de nascimento e ano atual. Imprimir a idade da pessoa.

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
int main()
{
        char nome[30], sexo;
        int idade;
        printf("Informe seu nome: ");
        gets(nome);
        printf("Informe seu sexo: ");
        scanf("%c",&sexo);
        printf("Informe sua idade: ");
        scanf("%d",&idade);
        if (sexo == 'f' || sexo == 'F' && idade < 25)
        printf("\n%s. ACEITA.\n\n", nome);
        else
        printf("\nNAO ACEITA.\n\n");
        system("PAUSE");
        return 0;
}
```

17) Criar um algoritmo que leia os limites inferior e superior de um intervalo e imprimir todos os números pares no intervalo aberto e seu somatório. Suponha que os dados digitados são para um intervalo crescente.

Exemplo:

Limite inferior: 3 Saída: 4 6 8 10

Limite superior: 12 Soma: 28

Repare que os valores iniciais e finais (3 e 12) não entram no cálculo e não são mostrados na saída

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <string.h>
main()
{
	int liminferior, limsuperior,x;
	float soma = 0;
	printf("Informe o valor do limite inferior: ");
	scanf("%d", &liminferior);
	printf("\nInforme o valor do limite superior: ");
	scanf("%d", &limsuperior);
	// verifica se o limite inferior digitado é par
	if (liminferior % 2 == 0)
	// se for par, soma 2 para começar com o próximo numero par
	liminferior = liminferior + 2;
	else
	// se nao for par, aumenta 1 para ele ficar par
	liminferior = liminferior + 1;
	printf("\n\n");
	// no limite superior subtrair 1 para ficar sempre com um numero
	// menor que o limite superior digitado.
	for (x=liminferior; x <= limsuperior -1; x=x+2)
	{
	printf("%d\t",x);
	soma = soma + x;
	}
	printf("\n\n");
	printf("Somatorio: %3.0f",soma);
	printf("\n\n");
	system("pause");
	return 0;
}
```

18)Apresentar o total da soma obtida dos cem primeiros números inteiros

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
main()
{
	int x,soma=0;
	for(x=1; x<=100;x++)
	soma = soma + x;
	printf("A soma dos 100 primeiros numeros inteiros eh: %d\n",soma);
	/* Pode ser feito assim tambem
	x=1;
	while (x <=100)
	{
	soma=soma + x;
	x=x+1;
```

```
        }
        printf("A soma dos 100 primeiros numeros inteiros eh: %d\n",soma);
        */
        system("pause");
        return 0;
}
```

19) Apresentar todos os números divisíveis por 4 que sejam menores que 200.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
main()
{
        int x;
        for(x=1; x<200;x++)
        if (x % 4 == 0)
        printf("%d\n",x);
        /* Pode ser feito assim tambem
        x=1;
        while (x <200)
        {
        if (x % 4 == 0)
        printf("%d\n",x);
        x=x+1;
        }
        */
        system("pause");
        return 0;
}
```

20) Elaborar um programa que efetue a leitura sucessiva de valores numéricos e apresente no final o total do somatório, a média e o total de valores lidos. O programa deve fazer as leituras dos valores enquanto o usuário estiver fornecendo valores positivos. Ou seja, o programa deve parar quando o usuário fornecer um valor negativo.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
main()
{
        int x, media=0, numero=0, conta=0;
        float soma = 0;
        // a variavel soma tem que ser float porque se não for a media
        // só da como resultado um número inteiro.
        while (numero >= 0)
        {
                printf("Informe um valor positivo: ");
                scanf("%d", &numero);
                if (numero > 0)
                {
                        soma=soma+numero;
                        conta = conta + 1;
                }
        }
        printf("A soma eh %3.0f e a media eh %5.2f\n",soma, (soma/conta));
        // %3.0f formata o numero float com 3 inteiros e zero decimal
        system("pause");
        return 0;
}
```

21) Elaborar um programa que efetue a leitura de valores positivos inteiros até que um valor negativo seja informado. Ao final devem ser apresentados o maior e menor valores informados pelo usuário.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
main()
{
	int x,numero=0, maior=0, menor=0;
	// Tem que inicializar as variaveis maior e menor com zero
	// Inicializa a variavel numero com zero para entrar no loop while
	while (numero >= 0)
	{
	printf("Informe um valor positivo: ");
	scanf("%d", &numero);
	if (numero > 0)
	// se o numero for positivo
	if (numero > maior)
	// se o numero informado for maior que o conteudo atual
	// da variavel maior, esta variavel recebe o numero informado
	maior = numero;
	else
	// senão a variavel menor recebe o numero informado
	menor = numero;
	}
	printf("O maior eh %d e o menor eh %d\n", maior, menor);
	system("pause");
	return 0;
}
```

22) Receber um número do teclado e informar se ele é divisível por 10, por 5, por 2 ou se não é divisível por nenhum destes.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
main()
{
	int numero;
	printf("Informe um valor positivo: ");
	scanf("%d", &numero);
	if (numero % 10 == 0 && numero%5 == 0 && numero % 2 == 0)
	printf("O numero eh divisivel por 10, 5 e 2\n");
	else
	printf("O numero nao eh divisivel por 10, 5 e 2\n");
	system("pause");
	return 0;
}
```

23) Um comerciante comprou um produto e quer vendê-lo com lucro de 45% se o valor da compra for menor que 20,00; caso contrário, o lucro será de 30%. Entrar com o valor do produto e imprimir o valor da venda.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
main()
{
	float valor_produto;
```

```
	printf("Informe o valor do produto: ");
	// gets(sigla);
	scanf("%f", &valor_produto);
	if (valor_produto < 20)
	printf("O valor da venda eh %3.2f\n", valor_produto * 1.45);
	else
	printf("O valor da venda eh %3.2f\n", valor_produto * 1.30);
	system("pause");
	return 0;
}
```

24) Ler a idade de uma pessoa e informar a sua classe eleitoral.

a. Não-eleitor (abaixo de 16 anos)
b. Eleitor obrigatório (entre 18 e 65 anos)
c. Eleitor facultativo (entre 16 e 18 e maior de 65 anos)

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
main()
{
	int idade;
	printf("Informe sua idade: ");
	scanf("%d", &idade);
	if (idade < 16)
	printf("Nao eleitor.\n");
	else
	if (idade >= 18 && idade <=65)
	printf("Eleitor obrigatorio.\n");
	else
	if ((idade >= 16 && idade < 18) || (idade > 65))
	printf("Eleitor faculdativo.\n");
	system("pause");
	return 0;
}
```

25) Receber do teclado, vários números e verificar se eles são ou não quadrados perfeitos. O programa termina quando o usuário digitar um número menor ou igual a zero.

(UM NÚMERO É QUADRADO PERFEITO QUANDO TEM UM NÚMERO INTEIRO COMO RAIZ-QUADRADA.)

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
main()
{
	int numero,c,p;
	printf("Informe um numero positivo: ");
	scanf("%d", &numero);
	while (numero > 0)
	{
	c = 1;
	p = c * c;
	while (p < numero)
	{
	c++;
	p=c * c;
	}
	if (p == numero)
	printf("\nO numero informado eh quadrado perfeito.\n");
```

```
        else
        printf("\nO numero informado nao eh quadrado perfeito.\n");
        printf("Informe um numero positivo: ");
        scanf("%d", &numero);
        }
        printf("\n");
        system("pause");
        return 0;
}
```

26) Receber um número e verificar se ele é triangular.
(UM NÚMERO É TRIANGULAR QUANDO É RESULTADO DO PRODUTO DE 3 NÚMEROS CONSECUTIVOS. EXEMPLO: 24 = 2 * 3 * 4)

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
main()
{
        int num,num1,p;
        num1 = 1;
        printf("Digite um numero: ");
        scanf("%d", &num);
        p = num1 * (num1 + 1) * (num1 + 2);
        while (p < num)
        {
        num1++;
        p = num1 * (num1 + 1) * (num1 + 2);
        }
        if (p == num)
        printf("\nEh triangular.");
        else
        printf("\nNao eh triangular.");
        printf("\n");
        system("pause");
        return 0;
}
```

27) Ler 3 números e imprimir se eles podem ou não ser lados de um triângulo.
A condição para isto é que A<B+C e B<A+C e C<A+B .

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<conio.h>
#include<string.h>
main()
{
        int A,B,C;
        printf("\n\t INFORME UM NUMERO : ");
        scanf("%d",&A);
        printf("\n\t INFORME UM SEGUNDO NUMERO : ");
        scanf("%d",&B);
        printf("\n\t INFORME UM TERCEIRO NUMERO : ");
        scanf("%d",&C);
        if(A<B+C != B<A+C != C<A+B)
        printf("\n\n ESTES NUMEROS SAO LADOS DE UM TRANGULO");
        else
        printf("\n\n ESTES NUMEROS NAO SAO LADOS DE UM TRIANGULO");
        printf("\n\n ");
        system("pause");
```

```
	return(0);
}
```

28) Ler 2 valores e somar os dois. Caso a soma seja maior que 10, mostrar a soma.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
main()
{
	int N1,N2;
	printf("\nINFORME PRIMEIRO NUMERO: ");
	scanf("%d",&N1);
	printf("\nINFORME SEGUNDO NUMERO: ");
	scanf("%d",&N2);
	if ((N1+N2)>10)
	printf("\nA SOMA E: %d",N1+N2);
	printf("\n\n");
	system("pause");
	return(0);
}
```

29) Entrar com um número e imprimir a raiz quadrada do número. Caso ele seja positivo. E o quadrado dele caso seja negativo.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
main()
{
	int NUM;
	printf("\nINFORME NUMERO: ");
	scanf("%d",&NUM);
	if (NUM>0)
	printf("\nA RAIZ QUADRADA DO NUMERO E:%f2.2",sqrt(NUM));
	else
	printf("\nO QUADRADO DO NUMERO E:%2.2f",pow(NUM,2));
	printf("\n\n");
	system("pause");
	return(0);
}
```

30) Ler um número do teclado e imprimir todos os números de 1 até o número lido. Imprimir o produto dos números.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<conio.h>
main()
{
	// x são os numeros menores e iguais ao numero digitado e tem que ser >1.
	// max é o número digitado.
	// p é o produto dos numeros gerados e começa com 1.
	int x, max=0, p=1;
	printf("\n\t INFORME UM NUMERO : ");
	scanf("%d",&max);
	for(x=1;x<=max;x++)
	{
	//Valor menor ou igual ao numero informado.
	printf("\n\t Numero gerados: %d ",x);
```

```
        p=p*x;
        }
        printf("\n\n O produto dos numeros gerados e: %d",p);
        printf("\n\n");
        system("pause");
        return(0);
}
```

31) Informe o tipo de carro (A, B e C). Informe o percurso rodado em km e calcule o consumo estimado, conforme o tipo, sendo (A=8, B=9 e C=12) km/litro

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
main()
{
        float percurso;
        char tipo;
        printf("Informe o tipo do carro: ");
        scanf("%c", &tipo);
        printf("\nInforme o percurso do carro: ");
        scanf("%f",&percurso);
        if (tipo == 'a' || tipo == 'A')
        printf("O consumo estimado do carro A sera %3.2f\ litros.\n",percurso/8);
        else
        if (tipo == 'b' || tipo == 'B')
        printf("O consumo estimado do carro A sera %3.2f litros.\n",percurso/9);
        else
        if (tipo == 'c' || tipo == 'C')
        printf("O consumo estimado do carro A sera %3.2f litros.\n",percurso/12);
        system("pause");
        return 0;
}
```

32)Escrever um programa que leia, valores inteiros, até ser lido o valor-99. Quando isso acontecer o programa deverá escrever a soma e a média dos valores lidos.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main()
{
        int num, soma=0;
        float media=0, cont=0;
        printf("\n DIGITE UM NUMERO INTEIRO: ");
        scanf("%d",&num);
        while(num!=-99)
        {
        soma=soma+num;
        cont++;
        printf("\n DIGITE UM NUMERO INTEIRO: ");
        scanf("%d",&num);
        }
        media=soma/cont;
        printf("\n\n A soma dos numeros e: %d ",soma);
        printf("\n\n A media dos numeros e: %3.2f ",media);
        printf("\n\n");
        system("pause");
        return(0);
}
```

33) Escrever um programa que receba vários números inteiros no teclado. E no final imprimir a média dos números multiplos de 3. Para sair digitar 0(zero).

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main()
{
	int num, soma=0;
	float media=0, cont=0;
	printf("\n DIGITE UM NUMERO INTEIRO: ");
	scanf("%d",&num);
	if(num %3==0 && num!=0)
	{
	soma=soma+num;
	cont++;
	}
	while(num!=0)
	{
	printf("\n DIGITE UM NUMERO INTEIRO: ");
	scanf("%d",&num);
	if(num %3==0 && num!=0)
	{
	soma=soma+num;
	cont++;
	}}
	media=soma/cont;
	printf("\n\n A media dos numeros e: %3.2f ",media);
	printf("\n\n");
	system("pause");
	return(0);
}
```

34) Receber dois numeros e imprimi-los em ordem crescente.

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>
#include<string.h>
main()
{
	int num1, num2;
	printf("\n\n Digite um numero: ");
	scanf("%d",&num1);
	printf("\n\n Digite um numero: ");
	scanf("%d",&num2);
	if(num1<num2)
	printf("\n\tOs numeros digitados na ordem crescente e: %d e %d\t",num1,num2);
	else
	printf("\n\tOs numeros digitados na ordem crescente e: %d e %d\t",num2,num1);
	printf("\n\n");
	system("pause");
	return 0;
}
```

35) Preencher um vetor com números inteiros(8 unidades); solicitar um número do teclado. Pesquisar se esse número existe no vetor. Se existir,imprimir em qual posição do vetor e qual a ordem foi digitado. Se não existir, imprimir MSG que não existe.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
main()
{
	int x, vet[8], num, achei=0;
	for(int x=0;x<8;x++)
	{
	printf("\n[%d] Digite um numero: ",x);
	scanf("%d",&vet[x]);
	}
	printf("\n\n");
	printf("Digite um valor a ser pesquisado: ");
	scanf("%d",&num);
	for(int x=0;x<8;x++)
	if(vet[x]==num)
	{
	printf("\n O numero %d esta na posicao %d: ",num,x);
	printf("\n O numero %d foi o numero %d a ser digitado: ",num,(x+1));
	achei=1;
	}
	if(achei!=1)
	printf("\n Este numero nao existe");
	printf("\n\n");
	system("pause");
	return(0);
}
```

WHILE E FOR

1) Solicitar a idade de várias pessoas e imprimir: Total de pessoas com menos de 21 anos. Total de pessoas com mais de 50 anos. O programa termina quando idade for =-99.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main()
{
	int idade,contador21=0,contador50=0;
	printf("\n DIGITE A IDADE: ");
	scanf("%d",&idade);
	while(idade!=-99)
	{
		if(idade<21)
		contador21++;
		if(idade>50)
		contador50++;
		printf("\n DIGITE A IDADE(PARA ENCERRAR DIGITE -99) : ");
		scanf("%d",&idade);
	}
	printf("\n O TOTAL DE PESSOAS COM MENOS DE 21 ANOS E:%d",contador21);
	printf("\n O TOTAL DE PESSOAS COM MAIS DE 50 ANOS E:%d",contador50);
	printf("\n\n");
	system("pause");
	return(0);
	}
```

2) Solicitar um número entre 1 e 4. Se a pessoa digitar um número diferente, mostrar a mensagem "entrada inválida" e solicitar o número novamente. Se digitar correto mostrar o número digitado.

```
#include<stdio.h>
#include<stdlib.h>
main()
{
	int num=-1;
	while(num<1||num>4)
	{
	printf("\n\n INFORME UM NUMERO ENTRE 1 e 4: ");
	scanf("%d",&num);
	if(num<1 || num>4)
	printf("\n VOCE NAO DIGITOU UM NUMERO ENTRE 1 e 4. ENTRADA INVALIDA. ");
	}
	printf("\n\n NUMERO DIGITADO:% d",num);
	printf("\n\n");
	system("pause");
	return(0);
}
```

3) Fazer um programa que gere a saída.

```
* * * * * * * * * * * * *
O,2,4,6,8,10,12,14
* * * * * * * * * * * * *
#include<stdio.h>
#include<stdlib.h>
main()
{
	int x;
	for(x=1;x<=19;x++)
	printf("%c",'*');
	printf("\n");
	for(x=0;x<=14;x=x+2)
	printf("%d ",x);
	printf("\n");
	for(x=1;x<=19;x++)
	printf("%c",'*');
	printf("\n");
	system("pause");
	return(0);
}
```

4) Solicitar um nome e escrevê-lo de trás pra frente.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main()
{
	char nome[30]; int x,t;
	printf("\n\n DIGITE UM NOME: ");
	gets(nome);
	t=strlen(nome);
	for(x=t+1;x>=0;x++)
	printf("%c",nome[x]);
	printf("\n\n");
	system("pause");
	return(0);
}
```

5) Fazer um programa que receba um valor n no teclado e determine o maior. A condição de término do programa é quando o usuário digitar zero.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main()
{
	int NUM,maior=NUM;
	printf("\n DIGITE UM NUMERO INTEIRO: ");
	scanf("%d",&NUM);
	while(NUM!=0)
	{
	if(NUM>maior)
	maior=NUM;
	printf("\n DIGITE OUTRO NUMERO(PARA ENCERRAR DIGITE 0) : ");
	scanf("%d",&NUM);
	}
	printf("\n O MAIOR NUMERO E: %d",maior);
	printf("\n\n");
	system("pause");
	return(0);
}
```

6) Apresentar o total da soma obtida dos cem primeiros números inteiros.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main()
{
	float soma=0;int x;
	for (x=1;x<=100;x++)
	soma=soma+x;
	printf("\n O TOTAL DA SOMA E: %4.3f",soma);
	printf("\n\n");
	system("pause");
	return(0);
}
```

7)Receber um numero e verificar se está entre 100 e 200. Se estiver na faixa,imprimir: "Voce digitou um numero entre 100 e 200", Senão estiver na faixa,imprimir:"Voce digitou um numero fora da faixa entre100 e 200".

```
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <stdio.h>
main()
{
	int num;
	printf ("informe um numero:");
	scanf("%d",&num);
	if(num>=100 && num<=200)
	printf("Voce digitou um numero entre 100 e 200");
	else
	printf("Voce digitou um numero fora da faixa 100 e 200");
	printf ("\n\n");
	system("PAUSE");
	return 0;
}
```

VETORES

1)Preencher um vetor com números inteiros(8 unidades); solicitar um número do teclado. Pesquisar se esse número existe no vetor. Se existir,imprimir em qual posição do vetor. Se não existir,imprimir MSG que não existe.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
main()
{
	int x, vet[8], num, achei=0;
	for(int x=0;x<8;x++)
	{
		printf("\n[%d] Digite um numero: ",x);
		scanf("%d",&vet[x]);
	}
	printf("\n\n");
	printf("Digite um valor a ser pesquisado: ");
	scanf("%d",&num);
	for(int x=0;x<8;x++)
		if(vet[x]==num)
		{
			printf("\n O numero %d esta na posicao %d: ",num,x);
			achei=1;
		}
	if(achei!=1)
		printf("\n Este numero nao existe");
	printf("\n\n");
	system("pause");
	return(0);
}
```

2) um vetor com os numeros pares do número 2 a 20.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
main()
{
	int vet[10],x, y=0;
	// preechimento do vetor vet
	for(x=0;x<=9;x++)
	{
		vet[x]=y+2;
		y=y+2;
	}
	//exibindo o vetor vet, por isso repete o for.
	for(x=0;x<=9;x++)
	//exibindo os valores pares 2,4,6,8,10,12,14,16,18,20.
		printf(" %d ",vet[x]);
	printf("\n\n");
	system("pause");
	return(0);
}
```

2) Preencher um vetor com os numeros pares do número 2 a 20. Preencher um vetor com os numeros de 10 a 19. Somar os vetores acima.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
main()
{
	int x, y=0, vet[10],vet1[10];
	// preechimento do vetor vet
	for(x=0;x<=9;x++)
	{
	vet[x]=y+2;
	y=y+2;
	}
	//exibindo o vetor vet
	for(x=0;x<=9;x++)
	//exibindo os valores pares 2,4,6,8,10,12,14,16,18,20.
	printf("\t %d ",vet[x]);
	// preechimento do vetor vet1
	for(x=0;x<=9;x++)
	{
	vet1[x]=x+10;
	}
	//exibindo o vetor vet1
	for(x=0;x<=9;x++)
	//exibindo os valores pares 10,11,12,13,14,15,16,17,18,19,20.
	printf("\t %d ",vet1[x]);
	//preechimento da soma dos vetores vet[x] + vet1[x]
	for(x=0;x<=9;x++)
	//exibindo a soma dos valores 2+10,4+11,6+12,8+13,10+14,12+15,14+16,16+17,18+18,20+19.
	printf("\t %d ",vet[x]+vet1[x]);
	printf("\n\n");
	system("pause");
	return(0);
}
```

3) Preencher um vetor de 8 elementos inteiros. Mostrar o vetor e informar quantos números são maior que 30, Somar estes números. Somar todos os números.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
main()
{
	int vet1[8], x, cont=0, soma=0, soma2=0;
	for(x=0;x<=7;x++)
	{
	printf("\nDigite um valor: ");
	scanf("%d",&vet1[x]);
	if(vet1[x]>30)
	{
	cont++;
	soma=soma+vet1[x];
	}
	}
	for(x=0;x<=7;x++)
	printf("\t%d",vet1[x]);
	printf("\n\n %d Numeros sao maiores que 30",cont);
	printf("\n\n A Soma dos numeros maiores que 30 e = %d",soma);
	for(x=0;x<=7;x++)
	soma2=soma2+vet1[x];
	printf("\n\n A Soma dos numeros digitados e = %d",soma2);
```

```
	printf("\n\n");
	system("pause");
	return(0);
}
```

4) Preencher um vetor com 3 nomes com 20 letras no máximo cada. Imprimir os Nomes.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include <string.h>
main()
{
	char nome[3][20],;
	int x;
	for(x=0;x<=2;x++)
	{
	printf("\n[%d] Digite o nome : %d ",x,(x+1));
	gets(nome[x]);
	}
	for(x=0;x<=2;x++)
	{
	printf("\n %s",nome[x],x);
	printf("\tO NOME %s tem %d letras",nome[x],strlen(nome[x]));
	}
	printf("\n\n");
	system("pause");
	return(0);
}
```

5) Neste exércicio temos dois vetores com 5 posições (0 a 4). Em cada vetor entraremos com cinco números. Mostrar os números e depois somar números que perteçam a mesma posição ou seja:
[0]+[0],[1]+[1],...

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
main()
{
	int vet1[5], vet2[5], num, cont=0, x;
	printf("\n");
	printf("\nVetor 1");
	for(x=0;x<=4;x++)
	{
	//este contador vai mostrar em qual posição o número digitado está.
	cont=0+x;
	printf("\t[%d] Digite um valor: ",cont);
	scanf("%d",&num);
	vet1[x]=num;
	}
	printf("\n\n");
	printf("\nVetor 2");
	for(x=0;x<=4;x++)
	{
	//este contador vai mostrar em qual posição o número digitado está.
	cont=0+x;
	printf("\t[%d] Digite um valor: ",cont);
	scanf("%d",&num);
	vet2[x]=num;
	}
	printf("\n\n");
	//Este for vai mostrar os valores de vet1.
```

```
        printf("\nVetor 1");
        for(x=0;x<=4;x++)
        printf("\t%d ",vet1[x]);
        printf("\n");
        //Este for vai mostrar os valores de vet2.
        printf("\nVetor 2");
        for(x=0;x<=4;x++)
        printf("\t%d ",vet2[x]);
        printf("\n\n");
        printf("\n\nSoma:");
        //Este for vai mostrar a Soma do vet1 + vet2.
        for(x=0;x<=4;x++)
        printf("\t%d ",vet1[x]+vet2[x]);
        printf("\n\n");
        system("pause");
        return(0);
}
```

6) Preencher um vetor de 8 elementos inteiros. Mostrar o vetor na horizontal com\t. Calcular a média do vetor. Mostrar quantos numeros são múltiplos de 5. Quantos números são maiores que 10 e menores que 30. Qual o maior número do vetor.

```
        #include<stdio.h>
        #include<stdlib.h>
        #include<math.h>
        main()
        {
        int vet1[8], x, cont=0, m5=0, NF=0, MN=0;
        float soma=0;
        for(x=0;x<=7;x++)
        {
        printf("Informe um numero %d: ", x+1);
        scanf("%d",&vet1[x]);
        printf("\n");
        }
        printf("\n\n");
        for(x=0;x<=7;x++)
        printf("\t%d",vet1[x]);
        printf("\n\n");
        for(x=0;x<=7;x++)
        {
        soma=soma+vet1[x];
        //multiplos de 5
        if(vet1[x]%5==0)
        m5++;
        if(vet1[x]>10 && vet1[x]<30)  //Maior que 10 e maior que 30
        NF++;
        //maior valor
        if(vet1[x]>MN)
        MN=vet1[x];
        }
        printf(" _____");
        printf("\n| A media do vetor e: %3.2f |",soma/8);
        printf("\n| Multiplos de 5: %d |",m5);
        printf("\n| Entre 10 e 30: %d |",NF);
        printf("\n| Maior numero: %d |",MN);
        printf("\n|_____|");
        printf("\n\n");
        system("pause");
        return(0);
}
```

7) Preencher um vetor com 3 nomes e mostrar quantas letras A e E tem nos 3 nome.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
main()
{
	int x, m;
	char nome[3][30],conta=0, conte=0, tam=0;
	for(x=0;x<3;x++)
	{
	printf("\tDigite um nome: ");
	gets(nome[x]);
	}
	for(x=0;x<3;x++)
	{
	tam=strlen(nome[x]);
	for(m=0;m<=tam-1;m++)
	{
	if(nome[x][m]=='A'||nome[x][m]=='a')
	conta++;
	if(nome[x][m]=='E' || nome[x][m]=='e')
	conte++;
	}
	}
	printf("Nos nomes digitados tem %d letras A",conta);
	printf("\nNos nomes digitados tem %d letras E",conte);
	printf("\n\n");
	system("pause");
	return(0);
}
```

8)Armazenar em Vetores, Nomes e Notas PR1 e PR2 de 6 alunos. Calcular a média de cada aluno e imprimir aprovado se a méida for maior que 5 e reprovado se média for menor ou igual a 5.

OBS.: 2 vetores para as notas tipo float. 1 vetor para os nomes. 1 vetor para a média. 1 vetor para situação.

```
#include<stdio.h>
#include<stdlib.h>
#include <string.h>
#include<math.h>
main()
{
	float PR1[3], PR2[3], media[3];
	char nome[3][30],nome1[3][30], situacao[3][50];
	int x, y=1;
	for(x=0;x<3;x++)
	{
	printf("Informe nome %d: ",x+1);
	gets(nome[x]);
	printf("Informe nota %d do aluno %s: ",y,nome[x]);
	scanf("%f",&PR1[x]);
	gets(nome1[x]);
	y++;
	printf("Informe nota %d do aluno %s: ",y,nome[x]);
	scanf("%f",&PR2[x]);
	gets(nome1[x]);
	y=1;
	}
	for(x=0;x<3;x++)
```

```
        {
        media[x]=(PR1[x]+PR2[x])/2;
        if(media[x]>5)
        strcpy(situacao[x],"Aprovado Parabens");
        else
        strcpy(situacao[x],"Reprovado - Vai estudar chimpanze");
        }
        printf("_____________________________________________________________________");
        printf("\n\nNome\tNota1\t\tNota2\tMedia\tSituacao");
        printf("\n_____________________________________________________________________");
        for(x=0;x<3;x++)
        {
        printf("\n%s\t%3.2f\t\t%3.2f\t%3.2f\t%s",nome[x],PR1[x],PR2[x],media[x],situacao[x]);
        }
        printf("\n_____________________________________________________________________");
        printf("\n\n");
        system("pause");
        return(0);
}
```

9) Preencher um vetor com 6 numeros e mostra-los na tela.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
main()
{
        int x, vetp[6];
        for(x=1; x<=6; x++)
        {
        printf ("\tDigite um numero: ");
        scanf("%d",&vetp[x]);
        printf ("\n");
        }
        for(x=1; x<=6; x++)
        printf ("%d\t",vetp[x]);
        printf ("\n");
        printf("\n");
        system("pause");
        return(0);
}
```

10) Preencher um vetor com 5 numeros e a medida que for digitado o numero, calcular o cubo e mostrar em outro vetor. Mostrar os dois vetores.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
main()
{
        float vetp[5], vetc[5];
        int x;
        printf ("\n");
        for(x=1; x<=5; x++)
        {
        printf ("\tDigite um numero: ");
        scanf("%f",&vetp[x]);
        vetc[x]=pow(vetp[x],3);
        printf (" %3.0f",vetc[x]);
        printf ("\n");
```

```
	}
	printf("\n");
	system("pause");
	return(0);
}
```

11) Preencher um vetor com 5 numeros e guardar o cubo dos numeros em outro vetor. Mostrar os dois vetores.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
main()
{
	float vetp[5], vetc[5];
	int x;
	printf ("\n");
	for(x=1; x<=5; x++)
	{
	printf ("\tDigite um numero: ");
	scanf("%f",&vetp[x]);
	vetc[x]=pow(vetp[x],3);
	printf ("\n");
	}
	for(x=1; x<=5; x++)
	printf (" %3.0f\t",vetc[x]);
	printf ("\n");
	printf("\n");
	system("pause");
	return(0);
}
```

12) Preencher um vetor com os numeros 10 a 20, e depois mostrar os elementos pares do vetor de trás prá frente. E também mostrar os números ímpares.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<conio.h>
#include<string.h>
main()
{
	system("color E"); // as cores vao de 0 a 15
	int z,x,vet[11], t=10;
	for(z=0; z<11; z++)
	{
	vet[z]= t;
	t++;
	}
	printf("\n\n");
	//este for vai contar os pares de tras pra frente.
	for(z=10; z>=0; z=z-2)
	printf("%d\t",vet[z]);
	printf("\n\n");
	for(x=1; x<11; x=x+2)
	printf("%d\t",vet[x]);
	printf("\n");
	system("pause");
	return(0);
}
```

13)Preencher um vetor com 5 numeros inteiros, solicitados no teclado e mostrar outro vetor com o cubo dos números do primeiro vetor.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
main()
{
system("color F5");
int z;
float vetcubo[5], vet[5];
for(z=0;z<5;z++)
{
printf("Digite um numero: ");
scanf("%f",&vet[z]);
printf("\n");
vetcubo[z]=pow(vet[z],3);
}
for(z=0;z<5;z++)
printf("%3.2f\t", vetcubo[z]);
printf("\n");
system("pause");
return(0);
}
```

14) Preencher um vetor com os numeros 10 a 20, e depois mostrar os elementos pares do vetor de trás prá frente.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<conio.h>
#include<string.h>
main()
{
	int z, vet[11], t=10;
	for(z=0; z<11; z++)
	{
	vet[z]= t;
	t++;
	}
	printf("\n\n");
	//este for vai contar os pares de tras pra frente.
	for(z=10; z>=0; z=z-2)
	printf("%d\t",vet[z]);
	printf("\n\n");
	system("pause");
	return(0);
}
```

FUNÇÕES

1)Função preencher Vetor, imprimir o Vetor, imprimir o quadrado, imprimir o primeiro e o ultimo numeros

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
void preenche(int vetp[])
{
	int x;
	for (x=0; x<6; x++)
	{
	// [%d] e o x vao mostrar a posição do número digitado
	printf ("\t[%d] Digite um numero: ",x);
	scanf("%d",&vetp[x]);
	printf ("\n");
	}
}
void imprimevet(int vetp[])
{
	int x;
	for (x=0; x<6; x++)
	printf (" [%d] %d\t",x,vetp[x]);
}
void quadrado(int vetp[])
{
	int x;
	for(x=0; x<6; x++)
	printf("%d\t",vetp[x]*vetp[x]);
}
void primultimo(int vetp[])
{
	printf ("%d\t %d\t",vetp[0], vetp[5]);
}
int vetp[5];
main()
{
	int x, resp;
	resp=1;
	while(resp!=0)
	{
	printf("\n 1 - Preenche o vetor: ");
	printf("\n");
	printf("\n 2 - Imprime o vetor: ");
	printf("\n");
	printf("\n 3 - Imprime o quadrado do vetor original: ");
	printf("\n");
	printf("\n 4 - Imprime o primeiro e ultimo numero: ");
	printf("\n");
	printf("\n 0 - Sair do programa: ");
	scanf("%d",&resp);
	printf("\n");
	if(resp==0)break;
	if(resp==1)
	preenche(vetp);
	if(resp==2)
	imprimevet(vetp);
	if(resp==3)
	quadrado(vetp);
	if(resp==4)
```

```
	primultimo(vetp);
	printf("\n");
	system("pause");
	system("cls");
	}
	return(0);
}
```

2) Criar uma função que retorna o seguinte: A função recebe 3 valores float e retornar o quadrado do 1º + a soma dos outros dois. Vai retornar o tipo inteiro.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
float calcula (float x, float y, float z)
{
	return (pow(x,2)+(y+z));
}
main()
{
	printf("Resultado: %3.2f\n", calcula(2,3,4));
	printf("\n\n");
	system("pause");
	return(0);
}
```

3)Criar uma função que retorna o seguinte: A função recebe 3 valores float (n1,N2,n3)e retornar o (x*x)+y+z ou seja : O quadrado do 1º + a soma dos outros dois. Vai retornar o tipo inteiro.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
float calcula (float x, float y, float z)
{
	return (pow(x,2)+(y+z));
}
main()
{
	int x;
	float vet[3];
	for(x=0;x<=2;x++)
	{
	printf("\n[%d] digite um numero: ",x);
	scanf("%f",&vet[x]);
	}
	printf("\n\nResultado: %3.2f\n", calcula(vet[0],vet[1],vet[2]));
	printf("\n\n");
	system("pause");
	return(0);
}
```

4) Criar uma função que receba um caractere como parâmetro e retorne 1 (um) caso seja uma vogal e zero caso não seja.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
int verificavogal (char M)
{
	if(M=='a' || M=='A' || M=='e' || M=='E'
	|| M=='i' || M=='I' || M=='o' || M=='O' || M=='u' || M=='U')
		return(1);
	else
		return(0);
}
main()
{
	char x;
	printf("Digite uma letra: ");
	scanf("%c",&x);
	if(verificavogal(x)==1)
	printf("\nA letra [ %c ] eh uma vogal: ",x);
	else
	printf("\nA letra [ %c ] eh uma constante: ",x);
	printf("\n\n");
	system("pause");
	return(0);
}
```

5) Criar um programa que receba um nome como parâmetro e retorne quais letras são vogais e quais são as constantes. Usar uma função que verifica se é uma vogal.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
int verificavogal (char M)
{
	if(M=='a' || M=='A' || M=='e' || M=='E'
	|| M=='i' || M=='I' || M=='o' || M=='O' || M=='u' || M=='U')
	return(1);
	else
	return(0);
}
main()
{
	char nome[30];
	int x,t;
	printf("\nDigite um nome: ");
	gets(nome);
	t=strlen(nome);
	for(x=0;x<=t-1;x++)
	if(verificavogal(nome[x])==1 )
	printf("\nA letra [ %c] eh uma VOGAL: ",nome[x]);
	else
	printf("\nA letra [ %c] eh uma CONSTANTE: ",nome[x]);
	printf("\n\n");
	system("pause");
	return(0);
}
```

6) Criar um programa que receba dois nomes e retorne quais letras são vogais e quais são as constantes. Usar uma função que verifica se é uma vogal.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
int verificavogal (char M)
{
	if(M=='a' || M=='A' || M=='e' || M=='E'
	|| M=='i' || M=='I' || M=='o' || M=='O' || M=='u' || M=='U')
		return(1);
	else
		return(0);
}
main()
{
	char nome1[30], nome2[30] ;
	int x,t, t2, cv=1, cc=1;
	printf("\nDigite um nome: ");
	gets(nome1);
	t=strlen(nome1);
	printf("\nDigite um nome: ");
	gets(nome2);
	t2=strlen(nome2);
	for(x=0;x<=t-1;x++)
	if(verificavogal(nome1[x])==1 )
	printf("\nA letra [ %c] eh uma VOGAL: ",nome1[x]);
	else
	printf("\nA letra [ %c] eh uma CONSTANTE: ",nome1[x]);
	printf("\n\n");
	for(x=0;x<=t-1;x++)
	if(verificavogal(nome2[x])==1 )
	printf("\nA letra [ %c] eh uma VOGAL: ",nome2[x]);
	else
	printf("\nA letra [ %c] eh uma CONSTANTE: ",nome2[x]);
	printf("\n\n");
	for(x=1;x<=t-1;x++)
	{
	if(verificavogal(nome1[x])==1 )
	cv++;
	else
	cc++;
	}
	printf("\n\n");
	system("pause");
	return(0);
}
```

MATRIZ

1)Ler um vetor vet de 10 elementos e obter um vetor quadrado cujos componentes deste vetor são o quadrado dos respectivos componentes de vet.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <string.h>
main()
{
	int x,y,tam;
	float vet[10], quad[10];
	for (x=0; x<=9; x++)
	{
	printf("Digite o numero %d: ",x+1);
	scanf("%f", &vet[x]);
	quad[x] = pow(vet[x],2);
	}
	printf("\n\nVetor VET: ");
	for (x=0; x<=9;x++)
	printf("%4.0f",vet[x]);
	printf("\n\n");
	printf("Vetor QUAD: ");
	for (x=0; x<=9;x++)
	printf("%4.0f",quad[x]);
	printf("\n\n");
	system("pause");
	return 0;
}
```

2) Criar um algoritmo que leia os elementos de uma matriz inteira de 4 x 4 e imprimir os elementos da diagonal principal.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <string.h>
main()
{
	int lin,col, tab;
	int mat[4][4];
	for (lin=0; lin<=3; lin++)
	{
	for (col=0; col<=3;col++)
	{
	printf("Digite ELEMENTO da linha %d, coluna %d da matriz: ",lin+1,col+1);
	// aqui no scanf preenchemos a matriz
	scanf("%d", &mat[lin][col]);
	}
	}
	//Imprimindo a matriz
	printf("Matriz\n");
	for (lin=0;lin<=3;lin++)
	{
	for (col=0;col<=3;col++)
	printf("%d\t",mat[lin][col]);
	printf("\n\n");
	}
	// Imprimindo a diagonal principal
```

```
	printf("\n\nDiagonal principal\n\n");
	for (lin=0; lin<=3;lin++)
	{
	printf("%d\n",mat[lin][lin]);
	for (tab=1;tab<=lin+1;tab++)
	printf("\t");
	}
	printf("\n\n");
	system("pause");
	return 0;
}
```

3)Criar um algoritmo que leia os elementos de uma matriz inteira de 3 x 3 e imprimir todos os elementos, exceto os elementos da diagonal principal.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <string.h>
main()
{
	int lin,col, tab;
	int mat[3][3];
	for (lin=0; lin<3; lin++)
	{
	for (col=0; col<3;col++)
	{
	printf("Digite ELEMENTO da linha %d, coluna %d da matriz: ",lin+1,col+1);
	// aqui no scanf preenchemos a matriz
	scanf("%d", &mat[lin][col]);
	}
	}
	//Imprimindo a matriz
	printf("Matriz\n");
	for (lin=0;lin<=2;lin++)
	{
	for (col=0;col<3;col++)
	printf("%d\t",mat[lin][col]);
	printf("\n\n");
	}
	// Imprimindo a matriz menos diagonal principal
	printf("\n\nMatriz menos a diagonal principal\n\n");
	for (lin=0; lin<3;lin++)
	{
	for (col=0;col<3;col++)
	{
	if (lin != col)
	printf("%d",mat[lin][col]);
	printf("\t");
	}
	printf("\n");
	}
	printf("\n\n");
	system("pause");
	return 0;
}
```

4) Criar um algoritmo que leia os elementos de uma matriz inteira de 3 x 3 e imprimir outra matriz multiplicando cada elemento da primeira matriz por 2.

Exemplo:
1 2 3 2 4 6
4 5 6 8 10 12
4 1 7 8 2 14

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <string.h>
main()
{
	int lin,col, tab;
	int mat[3][3], mat1[3][3];
	for (lin=0; lin<3; lin++)
	{
	for (col=0; col<3;col++)
	{
	printf("Digite ELEMENTO da linha %d, coluna %d da matriz: ",lin+1,col+1);
	// aqui no scanf preenchemos a matriz
	scanf("%d", &mat[lin][col]);
	}
	}
	//Imprimindo a matriz original
	printf("Matriz original\n");
	for (lin=0;lin<=2;lin++)
	{
	for (col=0;col<3;col++)
	printf("%d\t",mat[lin][col]);
	printf("\n\n");
	}
	// Preenche outra matriz (mat1) com os elementos multiplicados por 2
	for (lin=0;lin<=2;lin++)
	for (col=0;col<3;col++)
	mat1[lin][col] = (mat[lin][col])*2;
	// imprime a matriz mat1
	printf("\n\nMatriz com elementos multiplicados por 2\n\n");
	for (lin=0;lin<=2;lin++)
	{
	for (col=0;col<3;col++)
	printf("%d\t",mat1[lin][col]);
	printf("\n\n");
	}
	printf("\n\n");
	system("pause");
	return 0;
}
```

TÓPICO 8 – BIBLIOGRAFIA

Esta apostila trata-se de uma coletânea de textos e exercícios extraídos das fontes abaixo. Possui também colocações elaboradas pelos professores Flavio Costa e Claudio Passos. O maior intuito deste material é servir de apoio às aulas não substituindo os conteúdos apresentados nas mesmas.

I. www.unipacto.com.br

II. INTRODUÇÃO À LINGUAGEM C - ftp://ftp.ufv.br/dma/tutoriais/c%2B%2B/introd_ling_c.pdf

III. C Completo e Total 3a edição Herbert Schildt

IV. Apostilando – Coletânea de Exercícios em linguagem C, Autor: Rogério Rômulo de Oliveira

V. http://www.cprogramming.com