CURSO BÁSICO DE PROGRAMAÇÃO EM TURBO C

DESCRIÇÃO DO PROGRAMA

Introdução e conceitos básicos

a) Comandos e manipulação do editor ;
b) Estrutura básica de um programa em C ;
c) Declaração de constantes e variaveis ;
d) Funções de entrada e saida de dados ;

Operadores

a) Operador de atribuição = ;
b) Operadores relacionais e lógicos ;
c) Funções getch( ) , getchar( ) e getche( ) ;

Laços

a) Laço for;
b) Laço while;
c) Laço do-while;

Comandos de decisão

a) Comando if-else;
b) Comandos break , continue e goto;
c) Comando switch-case;

Funções

a) Chamando uma função;
b) Comando return;
c) funções recursivas;

Matrizes

a) Declaração de matrizes;
b) inicializando matrizes;
c) Matrizes como argumentos de funções;
d) Matrizes e strings;
e) Funções gets( ) , puts( ),strlen( ),strcat( ),strcpy( ),strcmp( );

Estruturas

a) Definindo estruturas;
b) Acessando membros da estrutura;
c) Passando estruturas para funções;

DEFINIÇÕES BÁSICAS

Programa - Conjunto de intruções distribuídas de maneira lógica , com a finalidade de executar satisfatoriamente determinada tarefa .

Linguagem de Programação - Conjunto de instruções possíveis utilizadas pelo homem para se comunicar com a máquina .

Endereço de memória - Número de uma posição de memória .

Compilador - Programa que traduz programas em linguagem de alto nível para linguagem de máquina.

Erro de compilação - Erro no programa em linguagem de alto nível que é detectado pelo compilador.

Erro de execução - Erro cometido em um programa que não é detectado até que o programa seja executado .

Variável - Simbolo que representa uma posição de memória .

Ponteriros - Tipos de variavéis que nos permite manipular endereços de outras variavéis .

INTRODUÇÃO E COMANDOS BáSICOS

A linguagem C assim como Pascal é uma linguagem estruturada .

Uma linguagem é chamada estruturada quando é formada por blocos chamados de funções . Um programa em C é formado por uma coleção de funções . Em um programa bem escrito cada função executa apenas uma tarefa . Cada função tem um nome e uma lista de argumentos que a mesma receberá .

A execução do programa escrito em C sempre começa pela função principal **main( )** .

EXEMPLO - 1

```
void  main( )
{
        int   idade ;
        clrscr( );
        printf (" Digite  sua  idade ");
        scanf (" %d",&idade );
        printf ( "Sua  idade  é  %d" ,idade );
        getch( );
}
```

Estudando cada linha :

void main ( ) → especifica o nome e o tipo da função (nesse caso void)
{ → inicio da função main

int idade → declara uma variável de nome idade e tipo inteiro
clrscr( ) → função predefinida para limpar a tela
printf (“ Digite sua idade “) → imprime a mensagem entre aspas na tela
scanf (“ %d”,&idade) → lê via teclado um valor que e colocado em idade
getch ( ) → função predefinida , espera uma tecla ser pressionada
} → fim da função main

VARIÁVEIS

Uma variável é um espaço de memória reservado para armazenar um certo tipo de dado e tendo um nome para referênciar o seu conteúdo .

Duas variáveis globais não podem ter o mesmo nome , uma variável local pode ter o mesmo nome de uma variável local de outra função .

DECLARANDO VARIAVEIS

tipo lista_variaveis;

TABELA CONTENDO OS TIPOS E TAMANHOS DE VARIAVEIS VÁLIDAS EM C

TIPO	EXTENSAO DO BIT	ESCALA
char	8	-128 a 127
int	16	-32768 a 32767
unsigned int	16	0 a 65535
signed int	16	-32768 a 32767
long int	32	-2147483648 a 147483648
float	32	3.4E-38 a 3.4E+38
double	64	1.7E-308 a 1.7E+308

INICIALIZANDO VARIAVEIS

Inicializar uma variavel significa atribuir um primeiro valor a essa variavel . Variáveis globais são sempre inicializadas com zero . Exemplo :

```
int k = 5 ;
char op = ‘f’;
float num = 21.5;
char nome[20] = “Fernanda”;
```

FUNÇAO *printf( )*

- Funcao predefinida no arquivo STDIO.H , e serve para imprimirmos um determinado dado na tela.
- Sintaxe
  printf(“string de controle”,variavel);
- Exemplo :
  int k=25;
  printf(“%i”,k);

FUNCAO *scanf( )*

- Função predefinida no arquivo STDIO.H , e serve pra se ler um determinado dado via teclado .
- Sintaxe
  scanf(“string de controle”,&variavel);
- Exemplo :
  char op;
  scanf(“%c”,&op);

EXERCÍCIO - 1

Faça um programa que leia dois números e em seguida mostre ,o produto a soma e a subtração entre eles .

TABELA CONTENDO OS TIPOS E TAMANHOS DE VARIAVEIS VÁLIDAS EM C

TIPO	EXTENSAO DO BIT	ESCALA
char	8	-128 a 127
int	16	-32768 a 32767
unsigned int	16	0 a 65535
signed int	16	-32768 a 32767
long int	32	-2147483648 a 2147483648
float	32	3.4E-38 a 3.4E+38
double	64	1.7E-308 a 1.7E+308

ALGUNS TERMOS COMUNS

Tempo de compilação : Os eventos ocorrem enquanto seu programa esta sendo compilado .
Tempo de execução : Os eventos ocorrem quando seu programa esta sendo executado .

Biblioteca : É um arquivo contendo as funções padrão que seu programa poderá usar .
Código fonte : É o texto de um programa que o usuário pode ler , comumente chamado de programa .

OPERADORES

C é uma linguagem rica em operadores . Alguns são mais usados do que outros como é o caso do operador de atribuição e dos operadores aritméticos , a seguir mostramos tabelas contendo os operadores aritméticos , relacionais e lógicos .
OPERADORES MATEMÁTICOS

OPERADOR	ACAO
-	subtração
+	adição
*	multiplicação
/	divisão
%	resto da divisão
--	decremento
++	incremento

OPERADORES LOGICOS

OPERADOR	ACAO
&&	and
| |	or
!	not

OPERADORES RELACIONAIS

OPERADOR	ACAO
>	maior que

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

FUNCOES

Um programa em C é formado por um conjunto de funções .

- Declarando uma função :

```
tipo identificador(lista de parametros)
{
        declaração de variáveis locais;
        comando ou bloco de comandos;
}
```

Exemplo :

```
void quadrado(int p)
{
        int k;
        k = p*p;
        printf("%i",p);
}
void main( )
{
        int k=25;
        quadrado(k);
        getch( );
}
```

COMANDO *return*

Serve para retornarmos um valor calculado dentro de uma função quando chamada de alguma parte do programa .

Exemplo :

```
float calc_sin(float arg)
{
        float val;
        val = sin(arg);
        return(val);
}
void main( )
```

```
{
        float valor;
        valor = calc_sin(50);
        printf("%f",valor);
}
```

COMANDOS DE DECISAO

Os comandos de decisão permitem determinar qual é a ação a ser tomada com base no resultado de uma expressão condicional .

COMANDO *IF-ELSE*

O comando if instrui o computador a tomar uma decisão simples .
Forma geral :

```
if ( condição ) comando ;
else comando ;
```

Exemplo :

```
/* programa do numero magico */
#include <stdio.h>
#include <conio.h>
void main( )
{
        int magico , entrada;
        magico = random(20);      //gera um numero entre 0 e 20
        clrscr( );
printf( "Adivinhe o numero : ");
scanf("%d",&entrada);
if (entrada == magico) printf (" == Você acertou ==");
else printf (" Você não acertou pressione qualquer tecla );
getch( );
}
```

EXERCÍCIO - 2

Escreva um programa onde o usuário entra com um número qualquer e o programa responda se o número e par ou impar .

Se for par emite a mensagem " O número é par " ou caso contrário "O número é impar ".

COMANDO *SWITCH CASE*

O comando switch pode ser usado no caso de alternativas múltiplas.
Forma geral

```
switch( variável )
{
    case constante1 : seqüência de comandos ; break;
    case constante2 : seqüência de comandos ; break;
    case constante3 : seqüência de comandos ; break;
        .
        .
     default : seqüência de comandos ;
}
```

O comando switch ao avaliar a expressão entre parênteses , desviamos para o rótulo case cujo valor coincida com o valor da expressão . O comando break serve para sairmos do bloco mais interno ao qual o break aparece . O comando break garante a execução de apenas uma chamada dentro do switch .

Exemplo :

```
#include <stdio.h>
void main ( )
{
        char opção;
        clrscr( );
        printf("A - imprime a letra f");
        printf("B - imprime a letra g");
        printf("C - imprime a letra h");
        opção = getch( ) ;
        switch(opção)
        {
                case 'a' : printf("f");break;
                case 'b': printf("g");break;
                case 'c' : printf("h");break;
        }
}
```

EXERCÍCIO - 3

Faça um programa contendo um menu com as seguintes opções :

S - soma
P - produto
U - subtração
D - divisão
Q - sair

O programa deve conter uma função para executar cada tarefa pedida : soma , subtração etc . Quando o usuário teclar ESC o programa deve terminar .

LAÇOS

Laços são comandos da linguagem C úteis sempre que uma ou mais instruções devam ser repetidas enquanto uma certa condição estiver sendo satisfeita .

LAÇO *FOR*

O laço for é geralmente usado quando queremos repetir algo um número fixo de vezes . Isto significa que utilizamos um laço for quando sabemos de antemão o número de vezes a repetir .
Forma geral :

for (inicialização ; condição ; incremento) comando ;

Exemplo :

```
/* programa que imprime os números de 1 a 100 */
#include <stdio.h>
#include <conio.h>
void main ( )
{
        int x;
        for ( x=1;x<=100;x++)
        printf ( "%d",x);
        getch( ) ;
}
```

Exemplo2 :

```
void main( )
{
        float num;
        for (num=1;num<20;num++)   raiz(num);
}
void raiz( float n)
{
        printf("\nn = %f   raiz quadrada = %f",n,sqrt(n));
}
```

LAÇO WHILE

Um laço while é apropriado em situações em que o laço pode ser terminado inesperadamente , por condições desenvolvidas dentro do laço .
Forma geral

while (expressão de teste**)** comando ;

Exemplo :

```
void  imprime(char op)
{
        int k=0;
        while ( k != 50 )
        {
                if (op==’p’)
                   if (k%2==0)  printf(“%i”,k);
                if (op==’i’)
                    if (k%2!=0)  printf(“%ik);
                k++;
        }
}
```

LAÇO DO WHILE

Este laço é bastante similar ao laço while e é utilizado em situações em que é necessario executar o corpo do laço pelo menos uma vez e depois avaliar a expressão de teste .

Forma geral

```
do
{
        comando ou bloco de comandos;

}
while(expressão de teste);
```

Exemplo :

```
void main( )
{
        char op;
        int sair = 0;
        do
        {
                op = getche( );
                switch(op)
                {
                        case ‘s’ : somatorio( );break;
                        case ‘f’  : fibonacci( );break;
                        case ‘q’ : sair = 1;
                }
        }while(sair!=1);
}
```

EXERCICIO - 4

Suponha um número N qualquer
se N é par então N agora é N / 2
se N é ímpar N agora é 3*N + 1
Assim para N = 3 calculamos a seguinte tabela :

3 → 10	4 → 2
10 → 5	2 → 1
5 → 16	1 → 4
16 → 8	4 → 2
8 → 4	2 → 1

Observe que apartir de sete iterações a sequencia 4 2 1 começa a se repetir . Faça um programa que calcule para um dado N o número de iterações até se chegar ao primeiro 1 .

EXERCÍCIO - 5

Faça um programa que imprima os elementos de uma PA e o somatório da mesma dados :
primeiro termo , numero de termos e razão

EXERCÍCIO - 6

Faça um programa que imprima um elemento da seqüência de Fibonacci , dado o numero do elemento .

EXERCÍCIO - 7

Faça um programa onde o usuário entra com um número decimal e o mesmo calcula e imprime o número no sistema binário .

EXERCÍCIO - 8

Escreva um programa onde o usuário digita um caracter do teclado e o mesmo imprime o caracter e seu código ASCII , até que o usuário tecle ESC .

OBS :. Código Ascii da tecla ESC = 27;

EXERCÍCIO - 9

Faça um programa onde o usuário entra com dois números A e B o programa devolve como resultado A elevado a B .

EXERCICIO - 10

Escreva um programa que solicite ao úsuario três números inteiros a,b,c onde a é maior que 1 . Seu programa deve somar todos os inteiros entre b e c divisieis por a .

FUNÇÕES RECURSIVAS

Uma função é dita recursiva quando se é definida dentro dela mesma. Isto é, uma função e´ recursiva quando dentro dela está presente uma instrução de chamada a ela própria.

```
// imprime uma frase invertida . Usa recursao
#include <stdio.h>
#include <conio.h>
void inverte( )
void main( )
{
        clrscr( );
        scanf( "%c",'\n');
        inverte( );
        getch();
}
void inverte ( )
{
        char ch ;
        if ((ch=getche( )) != '\r' ) inverte( );
        scanf("%c",ch)
}
```

MATRIZES E STRINGS

Uma matriz é uma coleção de variáveis do mesmo tipo que são referenciadas pelo mesmo nome .

A forma geral de se declarar uma uma matriz unidimensional é :

tipo nome_var[*tamanho*] ;

Strings : são matrizes unidimensionais de caracteres sempre terminada em zero '\0'. Ex :. char str[11];

Funções pré-definidas para se trabalhar com strings :

gets (str) : lê a string str via teclado ;
puts (str) : imprime string str ;
strcpy (s1 , s2) : copia o conteudo de s2 para s1 ;
strcat (s1, s2) : anexa s2 ao final de s1 ;
strcmp (s1,s2) : retorna 0 se as duas strings forem iguais ;
strlen (str): calcula e retorna o comprimento de str ;

EXERCICIO - 11

Faça um programa que leia uma senha digitada . Se a senha for correta imprime uma mensagem e se for falsa imprime outra mensagem .

EXERCÍCIO - 12

Escreva um programa que leia uma string e imprime a mesma de trás para frente .

EXERCÍCIO - 13

Construa um programa em que voce entra com um determinado numero de nomes de alunos . Depois da entrada dos nomes você digita o número do aluno e o programa mostra o nome do mesmo .

INICIALIZAÇÃO DE MATRIZES

Em C você pode inicializar matrizes globais . Você não pode inicializar matrizes locais .

Exemplo de inicializaçãao de matrizes :

```
int  mat[8]={5,2,1,5,4,5,4,7};
int  sqr[2][3]={
          2,3,6,
          4,5,5,
           };
char str[80] = “Linguagem  C”;
```

EXERCÍCIO - 14

Faça um programa onde o usuário entra com a ordem da matriz , o programa deve ler a matriz do tipo inteiro e imprimir a mesma .

EXERCÍCIO - 15

Faça uma programa que calcule a matriz transposta e oposta de uma matriz digitada pelo usuário .

EXERCÍCIO - 16

Escreva um programa que verifique a idêntidade de duas matrizes de mesma ordem .

TABELA CONTENDO FUNÇOES DE LEITURA VIA TECLADO

FUNCAO	OPERACAO
getchar ( )	lê um caractere ; espera por <enter>
getche ( )	lê um caractere com eco ; não espera por <enter>
getch ( )	lê um caractere sem eco ; não espera por <enter>
putchar ( )	imprime um caractere na tela
gets ( )	lê uma string via teclado
puts ( )	imprime uma string na tela

TABELA CONTENDO CODIGOS DE FORMATAÇÃO PARA *PRINTF* E *SCANF*

CODIGO	PRINTF	SCANF
%d	imprime um inteiro decimal	le um inteiro decimal
%f	ponto decimal flutuante	le um numero com ponto flutuante
%s	string de caracteres	le uma string de caracteres
%c	um único caractere	le um único caractere
%i	decimal	le um interio decimal
%p	imprime um ponteiro	le um ponteiro
%e	notacao cientifica	le um numero com ponto flutuante

TIPOS DE DADOS DEFINIDOS PELO USUARIO

ESTRUTURAS

Estrutura é uma coleção de variáveis referênciadas por um nome
As variáveis que formam a estrutura são chamados de elementos da estrutura .

EXEMPLO - 1

```
#include <stdio.h>
#include <conio.h>
struct endereco {
                    char nome[30];
                    char rua[40];
                    char cidade[20];
                    unsigned long int  cep;
                  };
struct endereco ficha ;
void main( )
{
        gets(ficha.nome);
        ficha.cep = 12345;
        printf("%u",ficha.cep);
        register int i;
        for ( i= 0 ; ficha.nome[i];i++)   putchar( ficha.nome[i]);
}
```

MATRIZES E ESTRUTURAS

Por exemplo , para se criar 100 conjuntos de variáveis do tipo struct ficha , você deve escrever :

```
struct endereco ficha[100] ;
```

Para acessar um elemento da estrutura 3 , você deve escrever :

```
printf( "%u",ficha[2].cep);
```

EXERCICIO - 2

Faça uma agenda com o nome de 5 clientes . Cada cliente é cadastrado , usando se a opção cadastrar , a agenda deve conter também as opções consultar e sair . Cada cadastro deve ter nome , endereço , cidade e telefone .

PASSANDO ELEMENTOS DA ESTRUTURA PARA FUNÇÕES

EXEMPLO - 2

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

```
#include <string.h>
struct lista {
                        char x ;
                        int y;
                        float ;
                };
struct lista exemplo;
void imprime (char ch);
void main ( )
{
        getchar( exemplo.x);
        imprime(exemplo.x);  //passa o valor do caracter em x
}
void imprime(char ch)
{
        printf("%c",ch);
        getch( );
}
```

PASSANDO ESTRUTURAS INTEIRAS PARA FUNÇOES

EXEMPLO -3

```
void funcao(struct lista par);
void main( )
{
        struct lista exe2;
        exe2.k = 5;
        funcao(exe2);
}
void funcao(struct lista par)
{
        printf("%i",par.k);
        getch( );
}
```

PONTEIROS

Ponteiro é uma variável que contém um endereço de memória de outra variável .

DECLARAÇÃO DE UMA VARIÁVEL PONTEIRO

*tipo *nome_da_variável ;*

OPERADORES DE PONTEIROS

& : operador que retorna o endereço de memória de seu operando
* : operador que retorna o valor da variável que ele aponta

EXEMPLO - 1

```
#include <stdio.h>
void main( )
{
        int *ender , cont , val ;
        cont = 100;
        ender = &cont;  //pega o endereço de cont
        val = *ender;      // pega o valor que esta no endereço ender
        printf ( " %d ",val )   // exibe o valor de val
        getch( );
}
```

EXEMPLO - 2

```
#include <stdio.h>
void main( )
{
        int x,*p1,*p2;
        x = 101;
        p1 = %x;
        p2 = p1;
        printf ( " %p    %d " ,p2,*p2);
        getch( );
}
```

PONTEIROS E MATRIZES

Na linguagem C há uma estreita relação entre ponteiros e matrizes . Observe o exemplo abaixo :

```
// versão usando matriz
#include <stdio.h>
#include <ctype.h>
#include <string.h>
void main( )
{
        char str[80];
        int i;
```

```
        printf( “digite uma string em letras  maiúsculas” );
        gets(str);
        for ( i= 0;str[i] ;i++ ) printf ( “%c”,tolower(str[i]));
        getch( );
}

// versão  usando  ponteiro
#include <stdio.h>
#include <ctype.h>
#include <string.h>
void main( )
{
        char str[80] , *p ;
        printf( “digite uma string em letras  maiúsculas” );
        gets(str);
        p = str;
        while ( *p )   printf ( “%c”,tolower(*p++) );
        getch( );
}
```

A versão deste programa usando se ponteiro é mais rápida que a outra usando se matriz . O compilador demora mais tempo para indexar uma matriz do que para usar o operador * .

INDEXANDO UM PONTEIRO

Em C é possível indexar um ponteiro como se ele fosse uma matriz .

EXEMPLO - 4

```
#include <stdio.h>
void main( )
{
        int i[5]={1,2,3,4,5};
        int *p , t;
        p = i;
        for ( t=0 ; t<5 ; t++)   printf (“%d”,p[t]);
        getch( );
}
```

EXEMPLO - 5 (Ponteiros e strings)

```
#include <stdio.h>
#include <conio.h>
int conta(char *s);
void main( )
{
        char  mat[20] = ”tamanho da frase”;
        int tam;
        tam = conta(mat);
        printf( “%i”,tam);
        getch( );
}

int conta(char *s)
{
        int i = 0;
        while(*s)  // repete até o final  da  string
        {
                i++;
                s++;
        }
        return i ;
}
```

EXEMPLO - 6 (Obtendo o endereço de um elemento da matriz)

```
/* este programa exibe a  string à direita  depois  que  o primeiro espaço é
encontrado */
#include <stdio.h>
void main( )
{
        char  s[80] , *p ;
        int i;
        printf( “digite uma string ”);
        gets(s);
        for ( i = 0 ; s[i]  && s[i] != ‘ ‘ ;i++ )
                p = &s[i];
        printf (p);
}
```

EXERCíCIO - 1

Escreva um programa cuja execução se segue abaixo :

```
c:\> digite uma frase :
c:\> carlos  <enter>
c:\> digite uma letra dessa  frase :
c:\> r  <enter>
c:\> rlos
```

CHAMADA POR REFERENCIA

A linguagem C usa a chamada por valor para passar argumentos para funções . Esse método copia o valor do argumento para o parâmetro . Assim não se altera o valor das variáveis usadas para chamar a função . Você pode alterar os valores dessas variáveis fazendo uma chamada por referência usando ponteiros . Agora o endereço da variável é passada para a função e não o seu valor .

EXEMPLO - 7

```
#include <stdio.h>
#include <string.h>
void leia( char  *s );
void main( )
{
        char str[80];
        leia (str);
        printf ("%s",str);
        getch( );
}
void leia( char  *s )
{
        char ch;
        for (int i= 0 ; i< 80 ; i++)
        {
                ch = getchar( );
                switch (ch);
                {
                        case '\n' : return;
                        case '\b' : if (t> 0) t-- ;break ;
                        default : s[t] = ch ;
                }
        s[80] = '\0';
}
```

EXERCíCIO - 2

Monte a função : **void troca (int * x , int * y)** , cuja execução seja assim :

```
c:\> digite dois numero  x e y respectivamente
c:\> 4   9  <enter>
c:\> x = 9  e  y = 4
```

EXERCíCIO - 3

Escreva um programa que leia e imprima uma número . Obs. não use a função padrão **scanf** , você deve definir uma , por exemplo :

void leia (int * x)

MATRIZES DE PONTEIROS

Podem existir matrizes de ponteiros , como acontece com qualquer outro tipo de dado .

EXEMPLO - 8

```
#include <stdio.h>
#include <conio.h>
char *erro[ ] =  {   "não  posso abrir  arquivo",
                     "erro de leitura ",
                     "erro de gravação " };
void mensagem(int i) ;
void main( )
{
        clrscr( );
        mensagem(2);
        getch( );
}
void mensagem( int i)
{
        printf( "%s",erro[i] );
}
```

RETORNANDO PONTEIROS

EXEMPLO - 9

```
#include <stdio.h>
#include <conio.h>
char *verifica( char c , char *s );
void main( )
{
        char s[80] , *p , ch;
        printf("digite uma frase e uma caracter \n ");
        gets(s);
        ch=getche( );
        p=verifica( ch,s);
        if (p) printf( "%s",p);
           else printf( "nenhuma correspondência foi encontrada")
}
```

```
char  *verifica(char c , char *s)
{
        int k = 0;
        while ( c !=  s[k] && s[k]  != '\0' ) k++;
        if ( s[k] ) return ( &s[k] );
        else  return  '\0' ;
}
```

EXERCíCIO - 4

Faça um programa que use a função (definida por você) :
char *concat(char *s1 , char * s2) . A função retorna um ponteiro que é a concatenaçao de s2 com s1 . Exemplo de chamada :

```
char  mat1[80]="casa " ;
char  mat2[80]="grande";
char  *p;
p  = concat(mat1,mat2);
printf( "%s",p);
```

MANIPULAÇÃO DE ARQUIVOS

ARQUIVOS BINÁRIOS E ARQUIVOS ASCII EM C

A linguagem C trata os arquivos de duas maneiras diferentes : como arquivos ASCII ou arquivos binarios.

Binarios : A linguagem C não interpreta o sentido do arquivo quando da leitura ou escrita de dados;

Ascii : Possui duas diferenças principais - quando encontra um Z^(ASCII 26) no arquivo que estiver sendo lido,C interpreta o ^Z como um caracter de fim de arquivo, e presume que o fim do arquivo foi alcançado;

- O caracter de nova linha '\n' e armazenado em disco como uma sequência ASCII 13 10.

OPCOES PARA ABERTURA DE ARQUIVOS

r	Abrir arquivo existente para leitura
w	Abrir (criar se necessário) para gravação
a	Abrir (criar se necessário) arquivo para acréscimo
r +	Abrir arquivo para leitura e escrita
w +	Criar e abrir arquivo para leitura e escrita
a +	Abrir arquivo para leitura acréscimo
rb	Abrir arquivo binario para leitura

wb	Abrir arquivo binario para escrita
ab	Abrir arquivo binario para acrescimo
rt	Abrir arquivo texto para leitura
wt	Criar arquivo texto para escrita
at	Abrir arquivo texto para acrescimo
r+b	Abrir arquivo binario para leitura e escrita
w+b	Criar arquivo binario para escrita
a+b	Abrir arquivo binario para acrescimo

ABERTURA DE ARQUIVOS

Dados podem ser gravados em arquivos de tres maneiras diferentes : como dados formatados , como conjunto de caracteres ou como estruturas .

No primeiro caso , poderemos utilizar a função *fprintf( )* , que é uma derivação da função *printf( )* e que possui a seguinte sintaxe :

fprintf(nome_do_ponteiro_de_arquivo,” tipo_de_dados”,dados)

Para iniciar a abertura do arquivo utilizamos a função *fopen( )* cuja sintaxe é a seguinte :

ponteiro_de_arquivo = fopen(“nome_do_arquivo”,”codigo”)

```
#include<stdio.h>
#include <conio.h>
void main( )
{
        FILE *fp;
        char texto[ ] = {“menina bonita”};
        fp = fopen(“alo.txt”,”w”);
        fprintf(fp,”%s”,texto);
        fclose(fp);
        getch();
}
```

Neste programa nos declaramos uma string constante *texto* que recebeu o tipo de formatacao “%s” na função fprintf , sendo criado no diretório atual um arquivo ASCII de nome “alo.txt” com a string “menina bonita”;

A função *fclose* fecha o arquivo , que não passa a receber mais nem uma ação de gravação ou leitura. Caso não seja utilizada esta função o próprio sistema se tratara de fechar o arquivo.

No segundo caso comumente utilizamos a função *putc* no lugar de *fprintf* sendo a sua sintaxe a seguinte :

puts(caracter,ponteiro_para_o_arquivo);

No terceiro caso e comum substituirmos estas funções pela função *fwrite* que e a função mais utilizada por programadores em C . Sua sintaxe apesar de um pouco mais elaborada não possui muitas dificuldades :

fwrite(estrutura,tamanho(estrutura),indice,ponteiro_de_arquivo);

Eis um exemplo de sua utilização :

```
#include <stdio.h>
#include <conio.h>
typedef struct entrada
{
        char nome[20];
        float deve;
};

void main()
{
        FILE *fp;
        entrada meus_dados[15];
        strcpy(meus_dados[0].nome,"Hugo Antonio");
        meus_dados[0].deve = 0.001;
        strcpy( meus_dados[1].nome,"Ricardo Tavares");
        meus_dados[1].deve = 13.89;
        fp = fopen("escritor.dat","w");
        fwrite(meus_dados,sizeof(meus_dados),1,fp);
        fclose(fp);
        getche( );
}
```

EXERCICIO - 1

Utilizando - se da função fprintf , criar um programa que grava em um arquivo em disco uma de dados que contenha o nome da pessoa e o seu numero de identidade; O programa deve possuir um menu com as opções "*entrada de dados*" e "*sair do programa*".

EXERCICIO - 2

Faça uma agenda com o dados de n clientes , sendo o valor de n lido no programa.Os dados de cada cliente devem estar numa estrutura denominada CLIENTE ,que possua os itens "*nome,numero de identidade,telefone,estado civil (s) olteiro ou (c) asado,cidade de origem* ".Tais dados devem ser gradados em disco.

LEITURA DE ARQUIVOS

Da mesma forma para a abertura de arquivos dados podem ser lidos no carater formatado , no carater de um conjunto de caracteres ou lendo-se uma estrutura inteira.No primeiro caso utilizamos a função *fscanf,* que possui função contraria a função *fprintf,* e cuja a sintaxe e :

fscanf(ponteiro de arquivo,"tipo das variaveis",variaveis);

No segundo caso utilizamos a função *getc* para o terceiro caso a função *fread* de sintaxe :

fread(estrutura,tamanho(estrutura),indice,ponteiro de arquivo);

```
#include <stdio.h>
#include <conio.h>
typedef struct entrada
{
        char nome[20];
        int deve;
};
void main()
{
        FILE *fp;
        entrada meus_dados[15];
        fp = fopen("escritor.dat","r");
        fread(&meus_dados,sizeof(meus_dados),1,fp);
        printf("%s  %d",meus_dados[0].nome,meus_dados[0].deve);
        printf("%s  %d",meus_dados[1].nome,meus_dados[1].deve);
        getch();
        fclose(fp);
}
```

FUNCAO *Fseek( )*

Posiciona o ponteiro de arquivo em determinada posição da estrutura , para um melhor aproveitamento de memória disponível.

```
#include <stdio.h>
#include <conio.h>
        typedef struct entrada
        {
                char nome[20];
                int deve;
        };
        void main()
        {
                FILE *fp;
                int d;
                entrada meus_dados[15];
                fp = fopen("escritor.dat","r");
                puts("Digite o numero do registro a ser recuperado ");
                scanf("%d",&d);
                fseek(fp,(long)(d*sizeof(meus_dados)),0);
                fread(&meus_dados,sizeof(meus_dados),1,fp);
                printf("%s  %d",meus_dados[0].nome,meus_dados[0].deve);
                printf("%s  %d",meus_dados[1].nome,meus_dados[1].deve);
                getch();
                fclose(fp);
        }
```

EXERCICIO - 3

Complementando o exercício 2 criar duas funções com os nomes *criar_arquivo , ler_arquivo ,* sendo cada uma destas opções chamadas de um menu que contém os seguintes itens :

1- CONSULTAR
2- CADASTRAR CLIENTE
3- SAIR