Universidade Católica de Pelotas
Escola de Informática
Ciência da Computação

C for Linux

por

Prof. Dr. Paulo Roberto Gomes Luzzardi
luzzardi@atlas.ucpel.tche.br
http://infovis.ucpel.tche.br/luzzardi

Referências Bibliográficas

KERNIGHAN, Briam W. C a Linguagem de Programação Padrão ANSI, Rio de Janeiro: Campus, 1990.

Junho, 2006

Sumário

1. Introdução

1.1 - Histórico

A linguagem de programação C foi desenvolvida nos anos 70 por Dennis Ritchie em um computador DEC PDP-11 que utilizava Sistema Operacional UNIX.

1.2 - Evolução

A partir de uma linguagem mais antiga chamada BCPL, desenvolvida por Martin Richards, surgiu uma linguagem chamada B, inventada por Ken Thompson que influenciou o desenvolvimento da linguagem de programação C.

1.3 - Características

- Linguagem de nível médio (combina recursos de alto e baixo nível);
- Bastante portável;
- Não é fortemente tipada;
- Permite a manipulação de bits, bytes e endereços;
- Permite escrever programas modulares e estruturados.

1.4 - Utilização

Permite o desenvolvimento e implementação de *software* básico, tais como:

- Sistemas Operacionais;
- Interpretadores;
- SGBD (**S**istema **G**erenciador de **B**anco de **D**ados);
- Editores (de texto e gráficos);
- Compiladores;
- Programas educacionais.

2. Ambiente de Programação Anjuta

2.1 Compilador gcc

O compilador **gcc** é o compilador C do projeto GNU, gratuito e de código aberto. O projeto GNU (Gnu's Not Unix) foi proposto por Richard Stallman em 1983. O projeto teve início em 1984, e a proposta era desenvolver um sistema operacional completo e livre, similar ao Unix.

2.2 Instalação do Anjuta

Um dos ambientes de programação que podem ser utilizados no Sistema Operacional Linux é o Anjuta. Para instalar o pacote execute os seguintes passos:

```
$ apt-get update <enter> ............... Atualiza a lista de pacotes
$ apt-get install gcc <enter> .......... Instala o Compilador C
$ apt-get install g++ <enter> .......... Instala o Compilador C++
$ apt-get install anjuta <enter> ....... Instala o ambiente anjuta
```

2.3 Interface do Anjuta

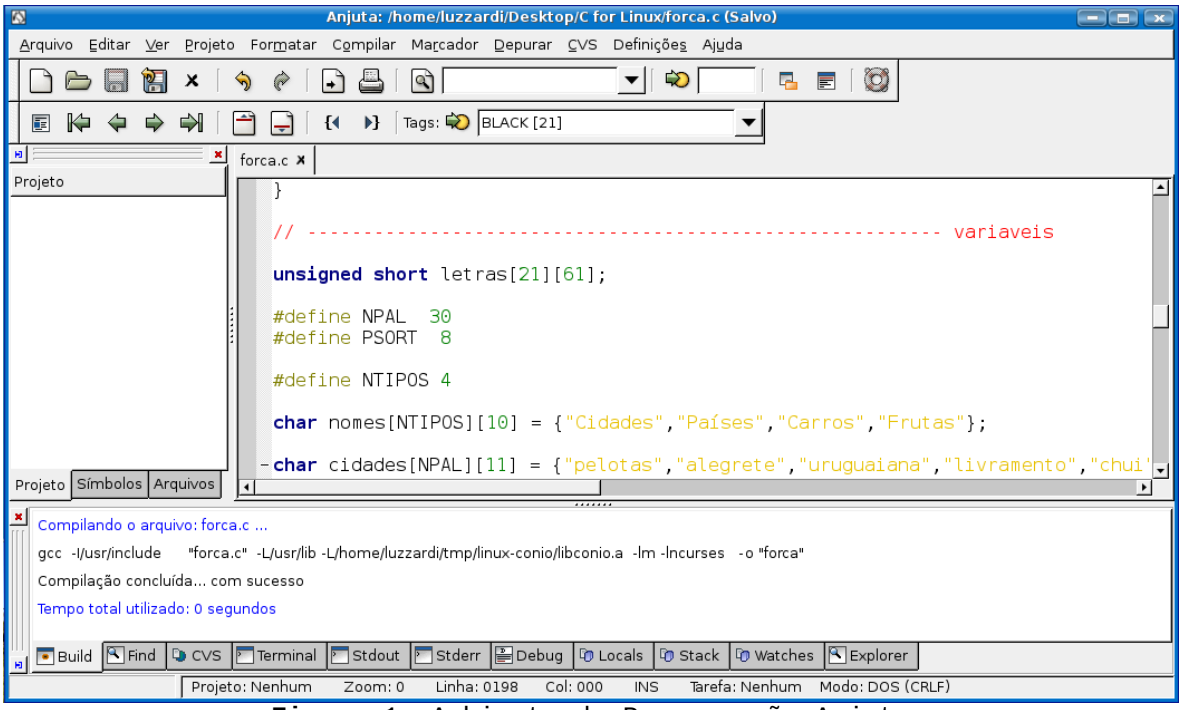


Figura 1: Ambiente de Programação Anjuta

3. Estrutura de um programa em C

3.1 - Identificadores

São os nomes criados pelo programador para fazer referência a **variáveis**, **contantes**, **funções** e **rótulos**.

Regras para a criação de identificadores:

- O primeiro caracter deve ser uma letra ou sublinha (_);
- Os caracteres seguintes devem ser letras, números ou sublinhas;
- Os primeiros 32 (*default*) caracteres são significativos;
- Não é permitido a utilização de caracteres em branco (caracter espaço).

Exemplos em variáveis:
```
int num_dentes = 32;
float inflação;
char a, _a;
```

Exemplos em constantes:
```
#define ESC    27
#define ENTER  10
```

Exemplos em funções:
```
x = raiz_quadrada(y);
printf("Valor: %.2f\n", inverso(n));
```

Observação: a função "inverso" é definida pelo programador (1/n).

3.2 – Comentários do programador

Os comentários do programador são linhas de código que não são compiladas pelo compilador, ou seja, servem apenas como anotações para serem lembradas mais tarde (por exemplo, quando forem feitas manutenções no programa). Em C os comentários podem ser feitos da seguinte maneira:

```
/* Isto é um comentário de várias linhas */

// Isto é um comentário em C++ de uma linha
```

Um comentário, pode ainda, utilizar várias linhas do programa. Veja o exemplo abaixo:

```
/* -----------------------------------------------------
    Função: STRING
    Parâmetros de entrada: x, y
    Parâmetros de saída: c, t
    Retorno: Sem Retorno
------------------------------------------------------ */
```

3.3 - Regras gerais para escrever um programa em C

Um programa em C é constituído de uma ou mais funções delimitadas por chaves { }, onde uma destas funções, obrigatoriamente é chamada **main**(). As principais regras são:

- Letras maiúsculas e minúsculas são tratadas como caracteres diferentes;

- O formato do texto é livre;
- A função **main**() especifica onde o programa começa e termina de ser executado;
- Todos os comandos são terminados por ponto e vírgula;
- Todas as variáveis devem ser declaradas;
- { função começa a ser executada;
- } função termina de ser executada.

Programa exemplo (1): Imprimir a data no seguinte formato: [Data: dd/mm/aaaa].

```
#include <stdio.h>

int main (void)
{
 int dia, mes, ano;      // variáveis locais utilizadas somente na função main

 dia = 12;
 mes = 6;
 ano = 2006;
 printf("Data: %02d/%02d/%04d\n", dia, mes, ano);
 return(0);
}
```

3.4 - Palavras reservadas

Definidas por K&R (Kernighan & Ritchie)				ANSI	
auto	double	if	static	const	asm
break	else	int	struct	enum	
case	entry	long	switch	signed	
char	extern	register	typedef	void	
continue	float	return	union	volatile	
default	for	sizeof	unsigned	near	
do	goto	short	while	_ds	

Observação: As palavras reservadas não podem ser utilizadas pelo programador como nome de variáveis, constantes ou funções, ou seja, não servem como identificadores.

3.5 - Declaração de variáveis

Sintaxe: tipo_dado_base lista_de_variáveis;

tipo_dado_base: deve ser um tipo de dado válido (***int, char, float, double ou void***)
lista_de_variáveis: um ou mais identificadores separados por vírgula.

Exemplo:

```
int main (void)
{
 int i, j ,k;
 float a, b;
 char ch;
```

3.5.1 - Onde as variáveis podem ser declaradas

- Definidas fora de todas as funções, incluindo a função **main**() são chamadas de **variáveis globais** e podem ser acessadas em qualquer parte do programa. Estas variáveis são alocadas estaticamente na memória RAM (***R**andom **A**ccess **M**emory* – Memória de acesso randômico).
- Definidas dentro de uma função são chamadas de **variáveis locais** e só podem ser acessadas dentro desta função. Estas variáveis são alocadas dinamicamente na memória RAM. Depois que uma função é executada, estas variáveis são desalocadas.
- Na declaração de parâmetros formais de uma função. Sendo estas **locais** e alocadas dinamicamente na memória RAM.

Observação: Memória ROM (***R**ead **O**nly **M**emory* – Memória somente de leitura).

Alocação de memória: Reserva de espaço de memória (RAM) para alocar uma variável.

- **Alocação estática de memória**: Tipo de alocação de memória em que uma variável é alocada (tem um espaço reservado) na memória RAM durante toda a execução do programa. Este espaço de memória é desalocado somente quando o programa acaba.
- **Alocação dinâmica de memória**: Tipo de alocação de memória em que uma variável é alocada (tem um espaço reservado) na memória RAM temporariamente. Este espaço de memória é desalocado quando o espaço não é mais necessário.

Programa exemplo (2): O programa realiza uma operação de potência X^y.

```
#include <stdio.h>      // o arquivo stdio.h é inserido dentro deste programa
#include <math.h>

float POT (float base, float expoente);   // protótipo da função POT
float resultado;                          // variável global

int main (void)
{
 float base, expoente;        // definição das variáveis locais

 printf("Base: ");            // função que permite imprimir dados na tela
 scanf("%f",&base);           // função que permite a entrada de dados via teclado
 printf("Expoente: ");
 scanf("%f",&expoente);
 resultado = POT(base, expoente);   // chamada da função POT
 printf("Resposta = %7.2f\n", resultado);
 return(0);
}

float POT (float x, float y)  // corpo da função POT definida pelo programador
{                             // os parâmetros x e y são variáveis locais
 float resp;                  // definição das variáveis locais

 resp = exp(log(x) * y);
 return(resp);                // retorno da função
}
```

Variáveis globais: resultado, POT
Variáveis locais: base, expoente, resp, x, y

Compilar por linha de comandos:

```
$ gcc prog2.c -o prog2 -lm <enter>
```

Onde: -o gera programa executável “prog2”
-lm linka a biblioteca “math.h”

3.5.2 - Inicialização de variáveis

Em C, é possível fornecer valores iniciais a maioria das variáveis ao mesmo tempo em que elas são declaradas, colocando um sinal de igual e uma constante após o nome da variável.

```
tipo_dado_base  nome_da_variável = constante;
```

Exemplos:

```
char ch = 'a';          // tipo_dado_base  nome_da_variável = constante_caracter
char s = “UCPel”;       // tipo_dado_base  nome_da_variável = constante_string
int n = 0;              // tipo_dado_base  nome_da_variável = constante_inteira
float y = 123.45;       // tipo_dado_base  nome_da_variável = constante_real
```

3.6 - Constantes

Valores fixos que o programa não pode alterar. As constantes podem ser de qualquer tipo básico.

Tipo	**Exemplos de constantes**
char	'a' 'n' '9'
int	1 123 2100 -234
float	123.23 4.34e-3
char *	"C for Linux" (forma de definir uma *string*)

3.6.1 - Constantes hexadecimais e octais

A linguaguem de programação C permite especificar constantes inteiras em hexadecimal ou octal. Uma constante **octal** começa com um **0** (zero), enquanto que uma **hexadecimal** começa por **0x**.

Exemplos:

```
int main (void)
{
 int hexadecimal = 0xFF;      // 255 em decimal
 int octal = 011;             // 9 em decimal
```

Observações:

- Qualquer número **octal** é formado por oito números (0 .. 7).
- Qualquer número **hexadecimal** é formado por dezesseis números (0 ..9, A, B, C, D, E, F).

3.6.2 - Constantes strings

Uma **string** é um conjunto de caracteres delimitados por aspas duplas. Em C não existe um tipo de dado especifico para definir uma **string**.

Uma **string** é definida como um vetor (unidimensional) de caracteres. Toda **string** deve ser finalizada pelo caracter especial **'\0'** (chamado de **NULL**).

Exemplo:

```
char s[6] = "UCPel";     ou    char s[6] = {'U', 'C', 'P', 'e', 'l', NULL};
```

'\0' é igual a NULL

0	1	2	3	4	5
'U'	'C'	'P'	'e'	'l'	NULL

Figura 2: Vetor de caracteres

Memória ocupada: 6 bytes

Observação: 'a' é diferente de "a"

'a' ocupa 1 byte na memória RAM (é do tipo **char**)
"a" ocupa 2 bytes na memória (toda **string** é terminada com o caracter '\0')

A palavra NULL (quer dizer nulo) está definida dentro do arquivo de **header** (cabeçalho) **stdio.h** (**st**an**d**ard **i**nput **o**utput).

```
#define NULL 0          // s[i] == (char) NULL; ou s[i] = '\0';
```

3.6.3 - Constantes especiais

As constantes especiais são utilizadas para representar caracteres que não podem ser inseridos pelo teclado. São elas:

Tabela 1: Constantes especiais

Constante	Significado
'\b'	Retrocesso
'\f'	Alimentação de formulário
'\n'	Nova linha
'\r'	Retorno de carro <CR>
'\t'	Tab horizontal
'\"'	Aspas duplas
'\''	Aspas simples
'\o'	Zero
'\\'	Barra invertida
'\a'	Alerta
'\o'	Constante octal
'\x'	Constante hexadecimal

3.7 - Comandos do pré-processador do C

3.7.1 - O comando #define

O comando #**define** é utilizado para definir uma macro-substituição. O compilador substitui o identificador pelo valor cada vez que aquele for encontrado no programa fonte antes da compilação do programa.

#define identificador valor

Exemplos:

```
#define TRUE      !0          // ou    #define TRUE 1
#define FALSE      0

#define ENTER 10
#define ESC   27
```

3.7.2 - O comando #include

O comando #**include** faz o compilador incluir um arquivo-fonte dentro de outro arquivo fonte.

```
#include <header .h>     // arquivo de header padrão do C
      ou
#include "header .h"     // arquivo de header escrito pelo programador
```

Observações: Normalmente os arquivos de ***header*** estão localizados em *"**/usr/include**"*. E os arquivos de ***header*** do programador estão localizados em seu diretório corrente *"**/home/luzzardi/fontes**", enquanto que as bibliotecas do **gcc** estão no diretório "**/usr/lib**".*

Exemplos:

```
#include <stdio.h>       // arquivo de header padrão do C (/usr/include)
#include "luzzardi.h"    // arquivo de header definido pelo programador
```

4. Tipos de dados

4.1 - Tipos básicos

A tabela abaixo exibe os cinco (5) tipos de dados básicos que podem ser utilizados pelo programador para definir suas variáveis. São exibidos os **tipos** básicos, a quantidade de **bits**, a **faixa de valores** válida e o número de **bytes** que cada tipo de dados ocupa na memória RAM (memória principal) ou em disco (quando armazenados na memória secundária).

Tabela 2: Tipos de dados

Tipo	Bits	Faixa de valores	Bytes
char	8	-128 à 127	1
int	32	-2147483648 à 21474483647	4
float	32	-3.4E-38 à 3.4E+38	4
double	64	-1.7E-308 à 1.7E+308	8
void	0	Sem valor	0

4.2 Modificadores de tipo

Os modificadores de tipo são utilizados para modificar os tipos de dados base, ou seja, se adaptando às necessidades do programador. Os modificadores modificam a quantidade de bits e bytes dos tipos-base, alterando, desta forma, a faixa de valores destes novos tipos de dados.

Tabela dos modificadores de tipos

Tabela 3: Modificadores de tipos

Modificador de tipo	Modificação	Descrição
signed	c/sinal	Números positivos e negativos
unsigned	s/sinal	Números positivos
long	longo	Aumenta o número de bytes do tipo
short	curto	Diminui o número de bytes do tipo

Tabela de tipos básicos modificados

Tabela 4: Tipos de dados básicos modificados

Tipo	Bits	Faixa de valores	Bytes
unsigned char	8	0 à 255	1
signed char	8	-128 à 127	1
unsigned int	32	0 à 4294967295	2
signed int	32	-2147483648 à 21474483647	2
short int	32	-2147483648 à 21474483647	2
long int	32	-2147483648 à 21474483647	4
unsigned short int	16	0 à 65535	2
signed short int	16	-32768 à 32767	2
unsigned long int	32	0 à 4294967295	4
signed long int	32	-2147483648 à 21474483647	4
long double	64	-1.7E-308 à 1.7E+308	8

5. Operadores

São símbolos especiais que obrigam o compilador a executar determinadas operações. Estas operações podem ser aritméticas, comparativas ou lógicas.

5.1 - Operadores aritméticos

São operadores que realizam uma operação matemática.

Tabela 5: Operadores aritméticos

Operador aritmético	Ação
-	Subtração
+	Adição
*	Multiplicação
/	Divisão
%	Resto inteiro da divisão
-- ++	Decremento/incremento

Precedência dos operadores aritméticos (Hierarquia nas Operações)

Tabela 6: Precedência dos operadores aritméticos

Hierarquia	Operação
1	Parênteses
2	Funções
3	++ --
4	- (menos unário)
5	* / %
6	+ -

Observação: Quando houver duas ou mais operações de mesma hierarquia, o compilador executa-as da **esquerda** para a **direita** (*veja tabela completa de precedência no item 21.2*).

5.2 - Operadores relacionais

São operadores que permitem comparar valores, ou seja, são utilizados principalmemte em comandos que possuem **condições**.

Tabela 7: Operadores relacionais

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

5.3 - Operadores lógicos

São operadores utilizados em comandos que tem mais de uma **condição**.

Exemplo: if (condição1 **&&** condição2 **||** condição3)

Tabela 8: Operadores lógicos

Operador lógica	Ação

&&	AND (e)
||	OR (ou)
!	NOT (não)

Precedência (Hierarquia dos operadores relacionais e lógicos)

Tabela 9: Precedência dos operadores relacionais e lógicos

Hierarquia	Operação
1	!
2	> >= < <=
3	== !=
4	&&
5	||

Observação: As expressões que utilizam operadores relacionais e lógicos retornam **0** (zero) para falso e **!0** (não zero) para verdadeiro, ou seja:

```
#define TRUE      !0
#define FALSE      0
```

5.4 - Incremento e decremento

São operadores aritméticos que permitem realizar operações de soma e subtração de forma simplificada.

- ++ adiciona (1) ao operando
- subtrai (1) ao operando

As seguintes operações são equivalentes:

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

Observação:

Os operadores (++ ou --) podem ser colocados antes ou depois do operando. Quando precede seu operando, C efetua a **operação** de incremento ou decremento antes de utilizar o valor do operando. Quando o operador vier depois do operando, C utiliza o **valor do operando** antes de incrementá-lo ou decrementá-lo.

Exemplo:

```
x = 10;          //  y será 11
y = ++x;         //  x será 11

x = 10;          //  y será 10
y = x++;         //  x será 11
```

5.5 - Operador de atribuição

O operador de atribuição é o sinal de igual =. A sintaxe do operador de atribuição pode ser escrito em uma das seguintes formas:

```
variável = constante;        x = 3;
variável = variável;         x = y;
variável = expressão;        x = a + b;
variável = função(x);        x = sqrt(y);
```

Programa exemplo (3): O programa calcula a idade de uma pessoa.

```
#include <stdio.h>

int main (void)
{
 int idade, ano_atual, ano_nasceu;

 printf("Ano ATUAL: ");
 scanf("%d",&ano_atual);                    // leitura do ano atual
 printf("Ano de NASCIMENTO: ");
 scanf("%d",&ano_nasceu);                   // leitura do ano de nascimento
 idade = ano_atual - ano_nasceu;            // atribuição – cálculo da idade
 printf("Sua IDADE eh %d\n",idade);
 return(0);
}
```

A linguagem de programação C permite utilizar o operador de atribuição em expressões, junto com operadores matemáticos, lógicos, relacionais, chamada de funções, e outros (como foi mencionado acima).

```
if ((produto = x * y) < 0)
```

Funcionamento: Primeiramente C atribui o valor x * y a variável **produto**, para depois avaliar a expressão, ou seja, comparar se o **produto** é menor (<) que zero.

5.6 - O operador sizeof

O operador **sizeof** (tamanho de) retorna o tamanho (em bytes) de uma variável ou de um tipo que está em seu operando.

Programa exemplo (4): O programa exibe a quantidade de bytes das variáveis e tipos.

```
#include <stdio.h>

int main (void)
{
 int x;
 char y;

 x = sizeof(short int);                          // x vale 2
 printf("x -> %d – y -> %d\n",x,sizeof(y));      // sizeof(y) é 1
 return(0);
}
```

5.7 - Casts

É possível forçar que o resultado de uma expressão seja de um determinado tipo. Para tanto deve ser utilizado uma construção chamada de **cast**, ou seja, pode ser utilizado para "**tipar**" uma variável com um tipo diferente do resultado da expressão.

variável = (tipo) expressão;

Programa exemplo (5): O programa imprime na tela o resultado de uma divisão.

```
#include <stdio.h>

int main (void)
{
 int x, y;
 float resp;

 printf("x = ");
 scanf("%d",&x);
 printf("y = ");
 scanf("%d",&y);
 resp = (float) x / y;        // é necessário um cast (float) pois a divisão de dois
 printf("Divisao = %.2f\n",resp);   // inteiros resulta em um inteiro
 return(0);
}
```

Observação: Em C, o tipo resultante de um inteiro dividido por outro inteiro é um inteiro, logo, deve-se utilizar um **cast** (float) para que o tipo resultante atribuído a variável **resp** seja **float**.

5.8 - Expressões

Uma expressão em C é qualquer combinação válida de **operadores** (aritméticos, relacionais ou lógicos), **constantes**, **funções** e **variáveis**.

Exemplo: c = sqrt (a) + b / 3.4;

5.8.1 - Conversão de tipos em expressões

Quando uma expressão é composta de tipos diferentes, C converte todos os operandos para o tipo do maior operando. Antes, os tipos abaixo são convertidos para:

char - convertido para **int**
float - convertido para **double**

Exemplo:

```
char ch;
int i;
float f;
double d;
float result;
```

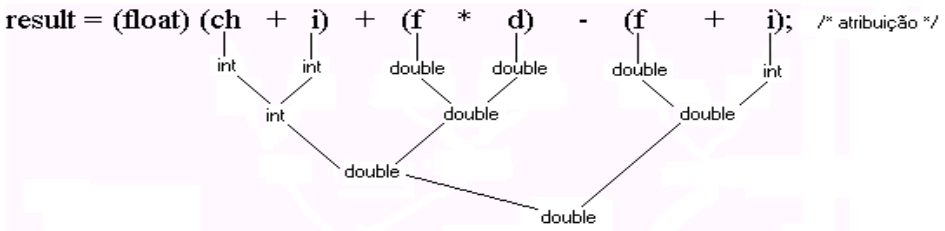


Figura 3: Conversão de tipos em expressões

6. Funções padrões

No linux é necessário incluir a biblioteca “math”. Por linha de comando fica assim:

```
$ gcc calc.c -o calc -lm <enter>
```

No ambiente de programação **anjuta** tem que incluir em “***Definições***” ... “***Definições de Compilação e Link...***” ... “***Bibliotecas***” ... “***m***” (*math routines*) ... “Adicionar”.

6.1 – abs

A função **abs** retorna o valor inteiro positivo – **absoluto**.

Sintaxe: int **abs**(int x);
Prototype: math.h

6.2 – fabs

A função **fabs** retorna o valor real positivo – **absoluto**.

Sintaxe: float **fabs** (float x);
Prototype: math.h

6.3 – asin

A função **asin** retorna o valor do **arco seno**. A variável **x** deve estar em radianos.

Sintaxe: double **asin** (double x);
Prototype: math.h
Faixa: -pi / 2 à pi / 2

6.4 – acos

A função **acos** retorna o valor do **arco cosseno**. A variável **x** deve estar em radianos.

Sintaxe: double **acos** (double x);
Prototype: math.h
Faixa: 0 à pi

6.5 – atan

A função **atan** retorna o valor do **arco tangente**. A variável **x** deve estar em radianos.

Sintaxe: double **atan** (double x);
Prototype: math.h
Faixa: -pi / 2 à pi / 2

6.6 – cos

A função **cos** retorna o valor do **cosseno**. A variável **x** deve estar em radianos.

Sintaxe: double **cos** (double x);
Prototype: math.h
Faixa: -1 à 1

6.7 – sin

A função **sin** retorna o valor do **seno**. A variável **x** deve estar em radianos.

Sintaxe: double **sin** (double x);
Prototype: math.h
Faixa: -1 à 1

6.8 – exp

A função **exp** retorna o valor do **expoente** (e^x).

Sintaxe: double **exp** (double x);
Prototype: math.h

6.9 – pow

A função **pow** (**pow**er) retorna o valor da **potência** (x^y).

Sintaxe: double **pow** (double x, double y);
Prototype: math.h

6.10 – sqrt

A função **sqrt** (**sq**uare **r**oo**t**) retorna o valor da **raiz quadrada**.

Sintaxe: double **sqrt** (double x);
Prototype: math.h

6.11 – log

A função **log** retorna o valor do **logaritmo natural**.

Sintaxe: double **log** (double x);
Prototype: math.h

6.12 – atof

A função **atof** converte **string** em **ponto flutuante**.

Sintaxe: double **atof** (const char *s);
Prototype: math.h

6.13 – atoi

A função **atoi** converte uma **string** em **inteiro**.

Sintaxe: int **atoi** (const char *s);
Prototype: math.h

6.14 – atol

A função **atol** converte uma **string** em **inteiro longo**.

Sintaxe: long int **atol** (const char *s);
Prototype: math.h

6.15 - log10

A função **log10** retorna o **logarítmo na base 10**.

Sintaxe: double **log10** (double x);
Prototype: math.h

6.16 – tan

A função **tan** retorna o valor da **tangente**. A variável **x** deve estar em radianos.
Sintaxe: double **tan** (double x);
Prototype: math.h

6.17 – rand

A função **rand** retorna um **número aleatário** entre 0 até 4294967295.

Sintaxe: int **rand** (void);
Prototype: stdlib.h
Faixa: 0 à 4294967295

```
int Random(int n)
{
 int x;

 x = rand() % n;  // retorna um número aleatório entre 0 e n-1
 return(x);
}
```

6.18 – srand

A função **srand** inicializa o gerador de **números aleatórios**. Não deve ser utilizado dentro de laços.

Sintaxe: void **srand** (unsigned int semente);
Prototype: stdlib.h
Forma de usar: srand(time(NULL));

```
void randomize(void)
{
 srand(time(NULL));
```

}

6.19 – system

A função **system** executa comandos e arquivos executáveis do sistema operacional linux.

Sintaxe: int **system** (const char *comando);
Prototype: stdlib.h
Retorna: 0 (ok) e -1 (erro)

Exemplos:

```
system ("ls -l");
system ("clear");
system ("ls");
```

Observação: **ls** e **clear** são comandos do sistema operacional Linux.

7. Comandos

7.1 - Tipos de Comandos

7.1.1 - Seqüência

São comandos, que no fluxo de controle do programa, são sempre executados passando a execução para a próxima instrução, ou seja, todos os comandos de seqüência são executados desde que eles não dependem de um comando de seleção.

Exemplo: (todas as instruções abaixo são de seqüência)

```
 system(“clear”);                    // limpa a tela
 printf("Digite uma letra: ");       // imprime na tela
 letra = getchar();                  // comando de atribuição
 printf("Digite um valor: ");
 scanf("%f",&valor);                 // entrada de dados via teclado
 resp = valor * 1.25;                // atribuição é um comando de seqüência
```

Observação: A função **getchar**() permite a entrada de um caracter via teclado. É necessário digitar <**enter**>.

7.1.2 - Seleção

São comandos, que no fluxo de controle do programa, permitem a seleção entre duas ou mais instruções, ou seja, este tipo de comando faz com que alguns comandos não sejam executados.

Exemplo:

```
if (numero % 2 == 0)          // testa se o número é par ou ímpar
    printf("Número: PAR\n");
else
    printf("Número: ÍMPAR\n");
```

ou

```
#include <stdio.h>
#include <math.h>       // por causa da função fabs

int main (void)
{
 float x, raiz;

 printf(“Digite um valor: “);
 scanf(“%f”, &x);
 if (x < 0)
    x = fabs(x);  // esta instrução só é executada se o valor de x for negativo
 raiz = sqrt(x);
 printf(“Raiz Quadrada: %.2f\n”, raiz);
 return(0);
}
```

7.1.3 - Repetição

São comandos, que no fluxo de controle do programa, permitem a repetição de uma ou mais instruções.

Programa Exemplo (6): O programa exibe na tela a tecla e o código (ASC II) da tecla digitada pelo usuário. O programa encerra quando o usuário digitar a tecla <**esc**> (**escape**).

```
#include <stdio.h>

#define ESC 27

int main (void)
{
 char tecla;

 do {
     printf("Digite uma tecla: ");
     tecla = getchar();
     printf("Tecla: %c - Código: %d\n", tecla, tecla);
    } while (tecla != ESC);
 return(0);
}
```

7.1.4 Atribuição

Veja item 5.5 (Operador de atribuição).

7.2 - Comando if

O comando **if** é um comando de seleção que permite selecionar um comando entre dois outros comandos (comandos simples) ou dois conjuntos de comandos (comandos compostos). Isto é feito através da avaliação de uma condição. O resultado desta avaliação (teste da condição) pode ser **verdadeiro** ou **falso**. Dependendo deste resultado um dos comandos é executado e o outro não.

Sintaxe:

```
if (condição)                              if (condição)
       comando 1;              ou              comando;
else
       comando 2;        // todos os comando são simples
```

Observação: O **else** é opcional.

Se a condição for avaliada como verdadeira (qualquer valor diferente de 0), o comando 1 será executado, caso contrário (condição falsa, valor igual a zero) o comando 2 será executado. Comando 1, comando 2 ou comando podem ser **simples** ou **compostos** (quando há mais de um comando ligado a outro, deve-se utilizar chaves ({ }). Veja exemplos abaixo

```
if (condição)           if (condição)
   {                       {
     comando 1;               comando 1;
     comando 2;               comando 2;
   }                          comando 3;
else                         }
      {
```

```
       comando 3;
       comando 4;              // todos os comando são compostos
      }
```

Programa exemplo (7): O usuário digita um número e o programa diz se este é **par** ou **ímpar**.

```
#include <stdio.h>

int main (void)
{
 int numero;

 printf("Digite um número: ");
 scanf("%d", &numero);
 if ((numero % 2) == 0)
     printf("Número é PAR\n");             // comando simples
 else
     printf("Número é IMPAR\n");           // comando simples
 return(0);
}
```

7.2.1 - if encadeados

Um **if** aninhado (ou encadeado) é um comando **if** dentro de outro comando **if** ou **if** ... **else**.

Programa exemplo (8): O usuário digita um número e o programa diz se este é **zero**, **positivo** ou **negativo**.

```
#include <stdio.h>

int main (void)
{
 int num;

 printf("Digite um número: ");
 scanf("%d", &num);
 if (num == 0)

       printf("Zero\n");                   // comando 1 do primeiro if

 else

    if (num > 0)
         printf("Positivo\n");
    else                                   // comando 2 do primeiro if
         printf("Negativo\n");

 return(0);
}
```

Observação: Note que no comando 2 (do primeiro **if**) existe outro **if**.

Exemplo:

```
if (x)                    // primeiro if
   if (y)                 // segundo if
        printf("1");
   else
```

```
        printf("2");
```

Primeira dúvida: O **else** pertence a qual **if**?

Funcionamento: Em **C** o **else** está ligado ao **if** mais próximo (mais interno), ou seja, neste exemplo o **else** pertence ao segundo **if**.

Segunda dúvida: Como fazer para que o **else** (do exemplo acima) pertença ao primeiro **if**?

Resposta: Deve-se utilizar chaves para anular a associação normal (veja exemplo abaixo).

Exemplo:

```
if (x)
   {
    if (y)
       printf("1");
   }
else
     printf("2");
```

7.3 - O comando switch

O comando **switch** é um comando de seleção que permite selecionar um comando entre vários outros comandos. Isto é feito através da comparação de uma variável a um conjunto de constantes. Cada um dos comandos está ligado a uma constante.

Sintaxe:

```
switch (variável)
{
 case constante_1 : seqüência de comandos;
                    break;
 case constante_2 : seqüência de comandos;
                    break;
          .
          .
          .
 default: seqüência de comandos;
}
```

O programa testa uma variável sucessivamente contra uma lista de constantes inteiras ou caracteres (**int** ou **char**). Depois de encontrar uma coincidência, o programa executa o comando ou bloco de comandos que estejam associados àquela constante. O comando **default** é executado se não houver nenhuma coincidência.

O comando **break** é utilizado para obrigar a saída do comando **switch**.
A opção **default** é opcional.

Observação: A variável não pode ser uma **string** (char *) e nem **real** (float).

Programa exemplo (9): O programa recebe um dígito de 0 à 9 e imprime na tela, este dígito, por extenso. Neste exemplo a variável dígito é inteira.

```
#include <stdio.h>

int main (void)
{
 int digito;

 printf("Dígito [0 .. 9]: ");
 scanf("%d", &digito);
 switch (digito)
 {
  case 0: printf("Zero\n");
          break;
  case 1: printf("Um\n");
          break;
  case 2: printf("Dois\n");
          break;
  case 3: printf("Três\n");
          break;
  case 4: printf("Quatro\n");
          break;
  case 5: printf("Cinco\n");
          break;
  case 6: printf("Seis\n");
          break;
  case 7: printf("Sete\n");
          break;
  case 8: printf("Oito\n");
          break;
  case 9: printf("Nove\n");
          break;
  default: printf("ERRO: Não é um digito\n");
 }
}
```

Programa exemplo (10): O programa recebe um dígito de 0 à 9 e imprime na tela, este dígito, por extenso. Neste exemplo a variável dígito é caracter, por causa disto as constantes estão entre apostrofes.

```
#include <stdio.h>

int main (void)
{
 char digito;

 printf("Dígito [0 .. 9]: ");
 scanf("%c", &digito);
 switch (digito)
 {
  case '0': printf("Zero\n");
            break;
  case '1': printf("Um\n");
            break;
  case '2': printf("Dois\n");
            break;
  case ‘3’: printf("Três\n");
```

```
            break;
  case ‘4’: printf("Quatro\n");
            break;
  case ‘5’: printf("Cinco\n");
            break;
  case ‘6’: printf("Seis\n");
            break;
  case ‘7’: printf("Sete\n");
            break;
  case ‘8’: printf("Oito\n");
            break;
  case '9': printf("Nove\n");
            break;
  default: printf("ERRO: Não é um digito\n");
 }
}
```

Programa exemplo (11): O programa constrói um menu com três funções: **inclusão**, **alteração** e **término**.

```
#include <stdio.h>

int main (void)
{
 char opcao;

 printf("[I]nclusão\n");
 printf("[A]lteração\n");
 printf("[T]érmino\n");
 printf("Qual a opção: ");
 opcao = getchar();
 switch (opcao)
 {
  case 'i':
  case 'I': inclusao();
           break;
  case 'a':
  case 'A': alteracao();
            break;
 case 't':
 case 'T': termino();
            break;
 default: printf("ERRO: Opção Inválida\n");
 }
}

void inclusao(void)
{
}

void alteracao(void)
{
}

void termino(void)
{
}
```

7.4 - Comando while

O comando **while** é um comando de repetição que permite executar um comando (simples) ou vários comandos (composto) diversas vezes. Isto é feito através da avaliação de uma condição. Enquanto a condição for verdadeira os comandos são repetidos. Quando a condição se tornar falsa o comando **while** é finalizado. O teste da condição é feita no início do comando, ou seja, antes que todos os comandos sejam executados.

Observação: Note que os comandos podem não ser executados nenhuma vez, basta a condição começar como falsa.

Sintaxe:

```
while (condição)        ou    while (condição)
      comando;                      {
                                     comando 1;
                                     comando 2;
// comando simples                  }

                                // comando composto
```

Condição: Qualquer expressão válida em C com resultado 0 (**false**) ou !0 (**true**). Na condição podem ser utilizados ainda **variáveis**, **constantes**, **funções**, **operadores** (aritméticos, relacionais e lógicos).

Funcionamento do comando: O loop (laço) é repetido enquanto a condição for verdadeira. Quando a condição se tornar falsa o controle do programa passa para a próxima instrução. O loop **while** verifica a condição no início do laço, por causa disto, normalmente, a variável de controle do laço deve ser inicializado.

Exemplo:

```
int i = 0;  // inicialização da variável de controle
while (i <= 10)          // condição i <= 10
     {
      printf("i = %d\n", i);
      i = i + 1;         // incremento
     }
```

Comando: Pode ser um comando vazio, simples ou composto que serão repetidos.

```
Comando vazio:  while (1);     // comando while não repete nenhum comando
      for (i = 0; i <= 1000; i++);  // comando for não repete nenhum comando
```

Verifique: Note que no final dos dois comandos (**while** e **for**) existe apenas um ponto-e-vírgula, isto é o sinal de comando vazio, ou seja, os comandos **while** e **for** que teriam outros comandos não os tem, caracterizando comandos vazios.

Problema freqüente de digitação: Muitas vezes o programador insere um ponto-e-vírgula no final de um comando **for** ou **while** por engano. Isto é um grave problema, pois este ponto-e-vírgula (inserido acidentalmente) faz com que os comandos que seriam repetidos, não são. Veja o exemplo abaixo:

Exemplo:

```
for (x = 1; x <= 10; x++);     // note o ponto-e-vírgula no final do comando for
```

```
    printf(“x = %d\n”, x);     // é impresso x = 11 na tela, porque?
```

Explicação: O comando **printf** não faz parte do comando **if** devido ao ponto-e-vírgula no comando **for**. O comando **for** termina quando a variável de controle **x** chega ao valor 11.

Comando correto: **for** (x = 1; x <= 10; x++) **printf**(“x = %d\n”, x);

Programa exemplo (12): O programa imprime caracteres de ‘A’ até ‘Z’ na tela.

```
#include <stdio.h>

int main (void)
{
 char letra = 'A';       // inicialização da variável de controle

 while (letra != 'Z')
       {
        printf ("Letra: %c\n", letra);
        letra++;                      // incremento
       }
}
```

7.5 - O comando for

O comando **for** é um comando de repetição que permite executar um comando (comando simples) ou vários comandos (comando composto) diversas vezes. Isto é feito através da avaliação de uma condição. Enquanto a condição for verdadeira os comandos são repetidos. Quando a condição se tornar falsa o comando **for** é finalizado.

Sintaxe:

```
for (inicialização; condição; incremento ou decremento)
        comando;
```

Inicialização: É um comando de atribuição (ou vários, separados por vírgula) que o compilador utiliza para inicializar a(s) variável(is) de controle do laço.

Condição: É uma expressão qualquer, que testa a variável de controle do laço contra algum valor para determinar quando o laço terminará.

Incremento ou decremento: Define a maneira como a(s) variável(is) de controle do laço serão alteradas após a repetição do laço.

- O laço (**for**) é repetido enquanto a condição é verdadeira.
- A condição é sempre testada no começo do laço.
- Qualquer uma das 3 partes do comando **for** (inicialização; condição; incremento) podem ser qualquer expressão válida em C.

Programa exemplo (13): O programa imprime de 1 até 10 na tela.

```
#include <stdio.h>

int main (void)
{
 int i;
```

```
 for (i = 1; i <= 10; i++)     // inicialização: i = 1
     printf("%d\n",i);         // condição: i <= 10
}                              // incremento: i++
```

Programa exemplo (14): O programa imprime na tela:

```
i = 1   j = 9
i = 2   j = 8
i = 3   j = 7
i = 4   j = 6
```

```
#include <stdio.h>

int main (void)
{
 int i, j;

 for (i = 1,,j = 9; i != j; i++,,j--)
      printf("i = %d   j = %d\n", i ,j);
}
```

O laço **for** é equivalente ao seguinte comando:

```
inicialização;
while (condição)
      {
       comando;
       incremento;            // ou decremento
      }
```

ou

```
inicialização;
do {
    comando;
    incremento;          // ou decremento
   } while (condição);
```

7.6 - O loop do { } while

O comando **do** ... **while** é um comando de repetição que permite executar um comando (comando simples) ou vários comandos (comando composto) diversas vezes. Isto é feito através do teste de uma condição. Enquanto a condição for verdadeira os comandos são repetidos. Quando a condição se tornar falsa o comando **do** ... **while** é finalizado. O teste da condição é feita no final do comando, ou seja, depois que os comandos são executados. (Note que os comandos são executados pelo menos uma vez).

Sintaxe:

```
do {
    comando;
   } while (condição);
```

- Repete o laço enquanto a condição for verdadeira.
- Testa a condição no final, fazendo com que o laço seja executado pelo menos uma vez.

Programa exemplo (15): Imprime na tela de 1 até 10.

```
#include <stdio.h>

int main(void)
```

```
{
 int i = 1;

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

7.7 - Interrupção de loops

7.7.1 - O comando break

Quando o programa encontra o comando **break** dentro de um laço, ele imediatamente encerra o laço, e o controle do programa segue para o próximo comando após o laço.

Programa exemplo (16): O programa imprime na tela a tecla digitada pelo usuário até que ele digite <***esc***>.

```
#include <stdio.h>

#define ESC 27

int main(void)
{
 char tecla;

 do {
      tecla = getchar();
      if (tecla == ESC)        // encerra o laço quando o usuário teclar ESC
         break;
      printf("Tecla: %c\n", tecla);
     } while (1);              // laço eterno
}
```

7.7.2 - O comando continue

O comando **continue** em vez de forçar o encerramento, força a próxima interação do laço e "pula", ou seja, não executa o código que estiver depois do comando **continue**.

Programa exemplo (17): O programa imprime na tela somente os números pares de 0 até 100.

```
#include <stdio.h>

int main(void)
{
 int i;

 for (i = 0; i < 100; i++)
     {
      if (i % 2)               // 0 é par, 1 é impar
         continue;
      printf("Par: %d\n", i);        // imprime somente números pares
     }
}
```

Nos laços **while** e **do {} while** um comando **continue** faz com que o controle do programa execute diretamente o teste da condição e depois continue o processo do laço.

No caso do comando **for**, o programa primeiro executa o incremento (ou decremento) do laço e, depois, executa o teste da condição antes de finalmente fazer o laço continuar.

7.8 - A função exit ()

A função **exit** aborta o programa em qualquer situação.

Modo de usar: **exit**(0); ou **exit**(!0);

8. Entrada e saída

Em C, as entradas e saídas são realizadas através da utilização das funções da biblioteca padrão do C, algumas são encontradas no arquivo **stdio.h**.

8.1 - Entrada e saída do console

As seguintes funções estão definidas em **stdio.h**.

Tabela 10: Funções de entrada e saída via console

Função	**Efeito** (Entrada)
getchar()	Lê um caracter do teclado, espera por <**enter**>
putchar()	Escreve um caracter na tela
scanf()	Lê qualquer tipo de dado, espera por <**enter**>
printf	Imprime na tela qualquer tipo de dado
fgets()	Lê uma *string* do teclado (aceita espaço)
puts()	Escreve uma *string* na tela

8.2 - Entrada e saída formatada

8.2.1 - Saída formatada (printf)

Sintaxe: **printf** ("string de controle", lista de variáveis);

String de controle: Formada pelos caracteres que a função imprime na tela, e pelos comandos de formatação (**%c**, **%s**, **%d**, **%f**) que definem a maneira como as variáveis serão impressas e caracteres especiais (**\n**, **\t**, ...).

Tabela 11: Comandos de formatação

Código	**Tipo**	**Formato**
%s	char *	String (vetor de caracteres)
%d	int	Inteiro decimal com sinal
%i	int	Inteiro decimal com sinal
%o	int	Inteiro octal sem sinal
%u	int	Inteiro decimal sem sinal
%x	int	Inteiro hexadecimal sem sinal (com a, b, c, d, e, f)
%X	int	Inteiro hexadecimal sem sinal (com A, B, C, D, E, F)
%f	float	Valor com sinal da forma [-]dddd.dddd
%e	float	Valor com sinal da forma [-]d.dddd e [+/-]ddd
%g	float	Valor com sinal na forma **e** ou **f** baseado na precisão do valor dado
%E	float	Mesmo que **e**, mas com E para expoente
%G	float	Mesmo que **g**, mas com E para expoente
%c	char	Um caracter
%%	nada	O caracter % é impresso
%n	int *	Armazena o número de caracteres escritos até o momento
%p	ponteiro	imprime como um ponteiro

Flags (Bandeiras):

- (-) Alinha o resultado à esquerda. Preenche o restante do campo com brancos. Se não é colocado, alinha o resultado à direita e preenche o restante à esquerda com zeros ou brancos.
- (+) O resultado sempre começa com o sinal + ou -
- (#) Especifica que o argumento será impresso usando uma das formas alternativas

Formas alternativas:

- 0 É colocado zeros (0) antes do argumento
- x ou X É colocado 0x (ou 0X) antes do argumento

Especificadores de largura do campo a ser impresso (exemplos):

Tabela 12: Especificadores de largura do campo

Prefixo	6d	6o	8x	10.2e	10.2f
%-+#0	+00555	01053	0x0022b	+5.50e+000	+000005.50
%-+#	+555	01053	0x22b	+5.50e+000	+5.50
%-+0	+00555	01053	000022b	+5.50e+000	+000005.50
%-+	+555	1053	22b	+5.50e+000	+5.50
%-#0	000555	001053	0x00022b	05.50e+000	0000005.50
%-#	555	01053	0x22b	5.50e+000	5.50
%-0	000555	01053	0000022b	05.50e+000	0000005.50
%-	555	1053	22b	5.50e+000	5.50
%+#0	+00555	01053	0x0022b	+5.50e+000	+000005.50
%+#	+555	01053	0x22b	+5.50e+000	+5.50
%+0	+00555	01053	000022b	+5.50e+000	+000005.50
%+	+555	1053	22b	+5.50e+000	+5.50
%#0	000555	001053	0x00022b	05.50e+000	0000005.50
%#	555	01053	0x22b	5.50e+000	5.50
%0	000555	001053	0000022b	05.50e+000	0000005.50
%	555	1053	22b	5.50e+000	5.50

8.2.2 - Entrada formatada (scanf)

Sintaxe: **scanf** ("string de controle", lista de variáveis);

String de controle: Local onde é definido o tipo de dado (**%d**, **%c**, **%s**, **&f**, ...) que será lido pelo teclado (não deve conter mais nenhum caracter).

Lista de variáveis: Nome da(s) variável(is) que será(ão) lida(s) pelo teclado.

Observação: Deve ser passado o endereço do argumento a ser lido.

```
int x;

scanf("%d",&x);
```

Programa exemplo (18): O programa permite a entrada (via teclado) do **nome**, **idade** e **salário** de uma pessoa.

```
#include <stdio.h>

int main(void)
{
 int idade;
 float salario;
 char nome[40];

 printf("Qual seu nome: ");
 scanf("%s", nome);            // ATENÇÃO: nome é igual ao &nome[0]
 printf("Idade: ");            // scanf não aceita espaços
 scanf("%d",&idade);           // &idade é o endereço da variável idade
 printf("Salário: ");
 scanf("%d",&salario);         // &salário é o endereço da variável salário
}
```

ScanSet: Funciona como uma entrada seletiva de caracteres na função "**scanf**". Veja o programa exemplo abaixo:

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

```
int main(void)
{
 char s[256] = "",str[256] = "";

 scanf("%[aeiou]", s);         // lê apenas as vogais “aeiou”
 printf("String (s): %s\n", s);
 scanf("%[^A-Z]", str);        // Não lê caracteres maiúsculos de A à Z
 printf("String (str): %s\n", str);
 return(0);
}
```

Modificadores de formato:

```
char s[31];

scanf(“%30s”,s);         // Copia apenas 30 caracteres para “s”
```

8.2.3 – Leitura de strings (fgets)

Sintaxe: **fgets** (char *string, int tam, FILE *fp);

String de controle: Local onde é definido o tipo de dado (**%d**, **%c**, **%s**, **&f**, ...) que será lido pelo teclado (não deve conter mais nenhum caracter).

Lista de variáveis: Nome da(s) variável(is) que será(ão) lida(s) pelo teclado.

```
#include <stdio.h>

int main()
{
 char str[31];

 printf("Digite seu nome: ");
 fgets(str, 30, stdin);        // stdin é a entrada padrão (teclado)
 puts(str);                    // puts imprime na tela uma string
 return(0);
}
```

Observação: ***stdin*** *(entrada)*, ***stdout*** (saída) e ***stderr*** (erro) são arquivos especiais que são abertos para cada programa que é executado.

8.3 - Saída na impressora (fprintf)

Sintaxe: **fprintf** (stdprn, "string de controle", lista de variáveis);

stdprn: Fila de impressão (**st**an**d**ard **pr**i**n**ter)

Programa exemplo (19): O programa imprime **“UCPel”** na impressora.

```
#include <stdio.h>

int main(void)
{
 fprintf(stdprn,"UCPel – Universidade Católica de Pelotas\n");
}
```

Observação:

```
// ------------------------- Salta uma página na impressora
 fprintf(stdprn,"%c\n",12);
// -------------------- Comprime os caracteres na impressora
 fprintf(stdprn,"%c\n",15);
```

9. Controle do vídeo e teclado

9.1 Biblioteca “ncurses” - modo texto

No linux para limpar a tela, posicionar o cursor, colorir texto, dentre outras funções, pode-se ser utilizada a biblioteca “ncurses”. O programa exemplo 78 mostra o uso desta biblioteca através da implementação de um jogo de caça-palavras.

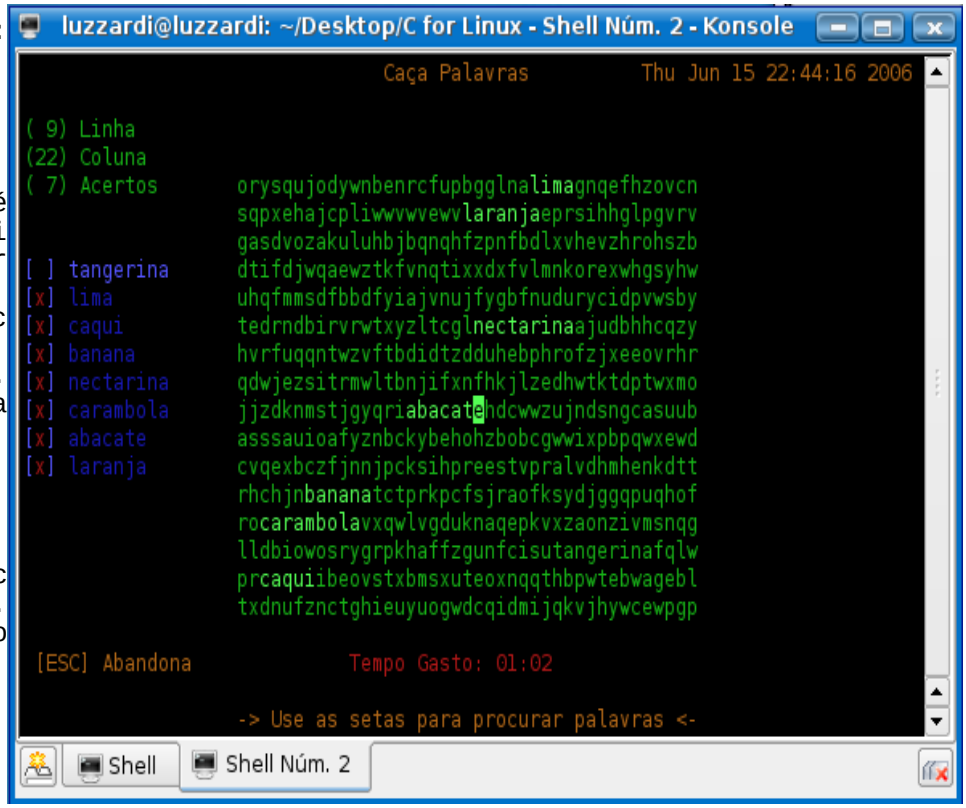


Figura 4: Jogo de Caça-Palavras

No linux é necessário incluir a biblioteca “curses”. Por linha de comando fica assim:

```
$ gcc palavras.c -o palavras -lcurses <enter>
```

No ambiente de programação **anjuta** tem que incluir em “***Definições***” ... “***Definições de Compilação e Link...***” ... “***Bibliotecas***” ... “***curses***” (*Screen handling and optimization routines*) ... “Adicionar”.

Tutorial da biblioteca ncurses:

http://www.del.ufrj.br/~pcfilho/ncurses/ncurses.htm

Veja mais detalhes da biblioteca **ncurses** no capítulo 20.

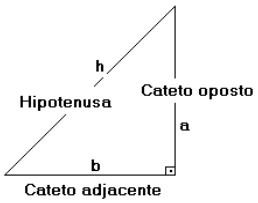
10. Lista de exercícios (comandos)

✓ Escreva um programa em C que recebe dois valores via teclado: **cateto adjacente** (b) e **cateto oposto** (a) e calcula o valor da **hipotenusa** dado pela seguinte fórmula:

Fórmula: $h^2 = a^2 + b^2$

Exemplo (Tela):

Cateto adjacente (b): **3** <**enter**>
Cateto oposto (a): **4** <**enter**>
Hipotenusa: **5**



✓ Escreva um programa em C que lê 4 notas via teclado: **n1**, **n2**, **n3** e **n4** obtidas por um aluno em 4 avaliações. Calcule a **média** utilizando a seguinte fórmula:

$$\text{Média} = \frac{n1 + n2 * 2 + n3 * 3 + n4}{7}$$

A seguir imprima na tela a **média** e o **conceito** do aluno baseado na seguinte tabela:

Média	**Conceito**
9,0 ou acima de 9,0	A
entre 7,5 (inclusive) e 9,0	B
entre 6,0 (inclusive) e 7,5	C
abaixo de 6,0	D

✓ Escreva um programa em C que recebe via teclado: **comprimento da circunferência**. O programa deve calcular e imprimir na tela o **diâmetro** e o **raio** da circunferência (veja exemplo abaixo).

Exemplo: Comprimento da circunferência: **36** <**enter**>
Diâmetro: **11.46**
Raio: **5.73**
Continua [S/N]? **N**

comprimento da circunferência = 2 . PI . raio
diâmetro = 2 . raio

Observação: O programa termina quando o usuário digitar 'N' ou 'n' na pergunta: **Continua [S/N]?**

✓ Desenvolva um programa em C que recebe via teclado: **peso** da carne que será vendida (em quilos) e **preço por quilo**. O programa deve calcular e imprimir na tela o **total a pagar**, o **valor pago ao ICMS** (17%) e o **lucro líquido** do açougue.

Exemplo (Tela):

Peso: **3.5** <**enter**>
Preço por Kg (R$): **4.90** <**enter**>
Total a pagar: **17.15**
ICMS (17%): **2.91**
Lucro líquido do açougue (R$): **14.24**
Sair [S/N]? **n**
Peso: **1.5** <**enter**>
Preço por Kg (R$): **9.00** <**enter**>
Total a pagar: **13.5**
ICMS (17%): **2.29**
Lucro líquido do açougue (R$): **11.21**
Sair [S/N]? **S**

- ✓ Escreva um programa em C que recebe via teclado: **tipo de animal** [1] Gado, [2] Eqüinos ou [3] Ovinos, **preço unitário do animal** e **quantidade de animais** comprados. O programa deve calcular e imprimir na tela: **preço total pago** e a **comissão do escritório de remate** (gado 5%, eqüinos 7% e ovinos 3%), conforme exemplo abaixo:

Exemplo (Tela):

Tipo de animal [1] Gado, [2] Eqüinos ou [3] Ovinos: **1** (SEM ENTER)
Preço unitário do animal (R$): **200** <**enter**>
Quantidade de animais: **10** <**enter**>
Preço total pago (R$): **2100.00**
Comissão a pagar (R$): **100.00**
Continua [S/N]? **s**
Tipo de animal [1] Gado, [2] Eqüinos ou [3] Ovinos: **2**
Preço unitário do animal (R$): **1000** <**enter**>
Quantidade de animais: **1** <**enter**>
Preço total pago (R$): **1070.00**
Comissão a pagar (R$): **70.00**
Continua [S/N]? **N**

- ✓ Reescreva o programa anterior recebendo via teclado uma **letra** para o **tipo de animal** [**G**]ado, [**E**]qüinos ou [**O**]vinos, **preço unitário do animal** e **quantidade de animais** comprado. O programa deve calcular e imprimir na tela: **preço total pago** e a **comissão do escritório de remate** (gado: 5%, eqüinos: 7% e ovinos: 3%), conforme exemplo abaixo:

Exemplo (Tela):

Tipo de animal [G]ado, [E]qüinos ou [O]vinos: **g** (SEM ENTER)
Preço unitário do animal (R$): **200** <**enter**>
Quantidade de animais: **10** <**enter**>
Preço total pago (R$): **2100.00**
Comissão a pagar (R$): **100.00**
Continua [S/N]? **s**

- ✓ Escreva um programa em C que recebe via teclado: a **data de hoje** da seguinte forma: **dia**, **mês**, **ano** e a sua **idade**, da seguinte forma: **anos**, **meses** e **dias** vividos. O programa deve calcular e imprimir a data de nascimento no seguinte formato: dd/mm/aaaa.

Exemplo (Tela):

Qual a data de hoje:
Dia: **16** <**enter**>
Mês: **6** <**enter**>
Ano: **2003** <**enter**>
Qual a sua idade:
Anos: **41** <**enter**>
Meses: **4** <**enter**>
Dias: **6** <**enter**>
Data de Nascimento: **10/02/1962**
Continuar [S/N]? **s**

- ✓ Escreva um programa em C que recebe via teclado um **número inteiro** de 0 à 99. O programa deve imprimir na tela este **número por extenso** (conforme exemplo abaixo). O programa termina quando o usuário digitar 0 (zero).

Exemplo: Número [0..99]: **23** <**enter**>
Vinte e três
Número [0..99]: **45** <**enter**>
Quarenta e cinco
Número [0..99]: **0** <**enter**>

- ✓ Escreva um programa em **C** que recebe via teclado: **quantidade de litros** vendidos, **tipo de combustível** ([**A**]lcool, [**G**]asolina ou [**D**]iesel) e o **tipo de pagamento** ([**P**]razo ou [**V**]ista). O programa deve calcular e imprimir na tela: **total à prazo, desconto** e o **total à vista**. O programa termina quando o usuário digitar 'N' ou 'n' na pergunta "Continua [S/N]?".

Tela de execução:

Quantidade de litros? **50** <**enter**>
Tipo de combustível [A]lcool, [G]asolina ou [D]iesel ? **g**
Tipo de pagamento [P]razo ou a [V]ista ? **v**
Total à prazo (R$) : **109.50**
Desconto (R$): **5.48**
Total à vista (R$): **104.02**
Continua [S/N]? **N**

Valores:

Álcool 1,23
Gasolina 2,19
Diesel 1,46

Desconto à vista: 5%

- ✓ Escreva um programa em C que recebe via teclado duas notas: **nota1 e nota2**. O programa deve imprimir na tela a **média**, o **conceito** do aluno (dado pela tabela abaixo) e a **situação** (aprovado, exame ou reprovado):

Conceito	Média	Situação
A	9,0 à 10,0	Aprovado
B	7,0 à 8,9	Aprovado
C	6,0 à 6,9	Exame
D	0,0 à 5,9	Reprovado

$$\text{Média} = \frac{\text{Nota1} + \text{Nota2}}{2}$$

Exemplo:

Nota1: **7** <**enter**>
Nota2: **8** <**enter**>
Média: **7.5**
Conceito: **B**
Situação: **Aprovado**
Sair [S/N]? **s**

Observação: O programa termina quando o usuário digitar ‘S’ ou ‘s’ na pergunta: **Sair [S/N]?**

- ✓ Escreva um programa em C que recebe via teclado uma **temperatura** e o **tipo de conversão** (converter para: [**C**]elsius ou [**F**]ahrenheit). Calcule e imprima na tela a temperatura correspondente a solicitação do usuário, conforme exemplos abaixo:

Exemplo:

```
Temperatura: 30 <enter>
Tipo de conversão (converte para: [C]elsius ou [F]ahrenheit): F
Temperatura em Fahrenheit: 86
Continua [S/N]? S
Temperatura: 86 <enter>
Tipo de conversão (converte para: [C]elsius ou [F]ahrenheit): C
Temperatura em Celsius: 30
Continua [S/N]? n
```

Fórmula:

$$9 \cdot \frac{C}{5} = F - 32$$

- ✓ Escreva um programa em C que recebe via teclado: **graus** (0 à 360), **minutos** (0 à 59) e **segundos** (0 à 59). O programa deve calcular e imprimir na tela o **ângulo em graus**, dado pela seguinte fórmula:

$$\text{ângulos em graus} = \text{graus} + \frac{\text{minutos}}{60} + \frac{\text{segundos}}{3600}$$

Exemplo:

Graus: **45** <**enter**>
Minutos: **45** <**enter**>
Segundos: **45** <**enter**>
Ângulo em Graus: **45.76**
Continua [S]im ou [N]ão? **s**
Graus: **45** <**enter**>
Minutos: **10** <**enter**>
Segundos: **15** <**enter**>
Ângulo em Graus: **45.17**
Continua [S]im ou [N]ão? **N**

OBSERVAÇÃO: Imprimir mensagens de erro se os valores de entrada estiverem fora da faixa:
ERRO: Graus fora da faixa, ERRO: Minutos fora da faixa ou ERRO: Segundos fora da faixa.

- ✓ Escreva um programa em **C** que recebe via teclado: **sexo** ([**M**]asculino ou [**F**]eminino), **altura** e **peso** da pessoa. O programa deve calcular e imprimir na tela: **peso ideal**, **diferença de peso** e **situação** (MAGRO, IDEAL ou GORDO) (conforme exemplo abaixo):

Exemplo: Sexo [M]asculino ou [F]eminino: **M** (**SEM enter**)

```
Altura: 1.65 <enter>
Peso: 92 <enter>
Peso Ideal: 62.0
Diferença de Peso: 30.0
Situação: GORDO
Sair [S/N]? s
```

PIM = 72,7 x altura – 58
PIF = 62,1 x altura – 44,7

MAGRO IDEAL GORDO

```
-------------------|-------------------|---------------------
        -5%                  pi                    5%
```

<u>Observação</u>: O programa termina quando o usuário digitar ‘S’ ou ‘s’ na pergunta: **Sair [S/N]?**

11. Vetores, matrizes e strings

Um vetor é uma coleção de variáveis de mesmo tipo (**agregados homogêneos**) que são referenciadas pelo mesmo nome, utilizando-se um índice para diferencia-los.

Um vetor consiste em locações contíguas de memória, ou seja, os elementos encontram-se em seqüência (contigüidade física). O menor endereço corresponde ao primeiro elemento, e o maior corresponde ao último elemento.

Uma vantagem na utilização de uma vetor é poder armazenar vários valores (elementos), na memória RAM, ao mesmo tempo, permitindo, por exemplo, compará-los e classificá-los.

Exemplo: Vetor unidimensional de inteiros (idades).

Tabela 16: Exemplo de um vetor unidimensional

Índice	Valor
0	24
1	12
2	36
3	41

11.1 - Vetores

Vetor (matriz de uma dimensão - 1D) é um tipo especial de matriz que possui apenas um índice, ou seja, permite armazenar variáveis unidimensionais (permite representar uma tabela).

tipo_dos_dados nome_do_vetor [número_de_elementos];

Onde:

tipo_dos_dados: tipo de dado de cada elemento (*char*, *int*, *float*, *double*).
nome_do_vetor: nome da variável que irá representar o vetor
número_de_elementos: número total de elementos do vetor

primeiro elemento: 0
último elemento: número_de_elementos – 1
número de bytes ocupados na memória RAM:
número_de_elementos x quantidade_de_bytes_de_um_elemento

Exemplo: int x[10]; // 10 elementos: x[0] à x[9]

primeiro elemento: x[0]
último elemento: x[número_de_elementos – 1], ou seja, x[9]
número de bytes: 10 x 2 = 20 bytes (um inteiro ocupa 2 bytes)

Observação: O programador deve verificar os limites do vetor, pois o compilador C não verifica estes limites, ou seja, o programador pode referenciar qualquer elemento do vetor, inclusive um que não existe. Isto pode causar um grave erro, "*travar*" o programa.

11.2 – Strings

Uma **string** (cadeia de caracteres) é um tipo de dado que consiste de um conjunto de caracteres. Uma *string* pode armazenar qualquer tipo de caracter válido da tabela ASCII.

ASCII: **A**mericam **S**tandard **C**ode for **I**nformation **I**nterchange.
url: http://www.asciitable.com

Em C, uma **string** consiste em um vetor de caracteres finalizado por um zero. Um zero é representado como **(char) NULL**.

Observação: Na inicialização de uma constante *string* "**teste**" não é necessário acrescentarmos o NULL, pois o compilador faz isso automaticamente.

char string[] = "**teste**";

Observação: Note que no exemplo acima, não foi necessário especificar o **número_de_elementos** da **string**. O compilador C interpreta que o tamanho da **string** é a quantidade de caracteres da inicialização (5) mais um (1) para o NULL, totalizandom neste caso, seis (6) elementos. Isto é um problema se quisermos (mais tarde) acrescentar mais caracteres na variável **string**, pois não foi reservado espaço préviamente para estes novos caracteres.

11.3 - Matrizes (Multidimensional)

tipo_dos_dados nome_variável [tamanho_1][tamanho_2]...[tamanho_n];

Exemplo: float y[5][5]; // matriz 2D

Para acessar o elemento 3, 4 da matriz y, deve-se escrever y[3][4]. Note que o primeiro elemento é y[0][0] e o último elemento é y[4][4]. O total de elementos é 25.

11.4 - Vetor de strings

Para criar um vetor de *strings*, deve-se utilizar uma matriz bidimensional de caracteres, onde o tamanho do índice esquerdo determina o número de *strings* e o tamanho do índice direito especifica o comprimento máximo de cada *string*.

Exemplo: char nome[3][8];

Tabela 17: Exemplo de um vetor de strings

	0	1	2	3	4	5	6	7
0	'U'	'C'	'P'	'e'	'l'	NULL	*lixo*	*lixo*
1	'U'	'C'	'S'	NULL	*lixo*	*lixo*	*lixo*	*lixo*
2	'U'	'**F**'	'P'	'e'	'l'	NULL	*lixo*	*lixo*

Cria um vetor com 3 strings com 7 caracteres + '\0' (NULL) cada uma. Para acessar uma string particular deve-se especificar apenas o índice esquerdo, ou seja, nome[0], nome[1] ou nome[2].

nome[0] -> "UCPel"
nome[1] -> "UCS"
nome[2] -> "UCFel"

Observação: Pode-se acessar também qualquer caracter de qualquer uma das *strings*, isto é feito utilizando os dois índices, como por exemplo, nome[2][1] é caracter 'F'.

11.5 - Inicialização de matrizes e vetores

```
tipo_dos_dados  nome_matriz [tam_1]...[tam_n] = {lista_valores};
```

lista_valores: lista de constantes separadas por vírgulas que são compatíveis em tipo com o tipo base da matriz.

Exemplo:
```
int i[10] = {0,1,2,3,4,5,6,7,8,9};  // vetor i – 1D
            ou
int i[] = {0,1,2,3,4,5,6,7,8,9};
```

Observação: Quando um vetor é declarado e inicializado (ao mesmo tempo) o número de elementos (neste caso 10) pode ser suprimido, ou seja, neste caso é opcional (veja exemplo anterior).

11.6 - Inicialização de um vetor de caracteres

```
char nome_vetor [tamanho] = "string";
```

Exemplo:
```
char str[4] = "sim";
```

0	1	2	3
's'	'i'	'm'	NULL ou '\0'

11.7 - Inicialização de matrizes multidimensionais

```
int y[4][2] = { {1,1}, {2,4}, {3,9}, {4,16} };
```

```
y[0][0] = 1       y[2][0] = 3
y[0][1] = 1       y[2][1] = 9
y[1][0] = 2       y[3][0] = 4
y[1][1] = 4       y[3][1] = 16
```

11.8 - Inicialização de vetores e matrizes sem tamanho

Na inicialização de uma matriz (ou vetor), se não for especificado seu tamanho, então o compilador C cria uma matriz (ou vetor) grande o suficiente para conter todos os inicializadores presentes.

Exemplo:
```
char s[] = "UCPel";          // s ocupa 6 bytes
```

Programa exemplo (23): O programa permite armazenar ***n*** nomes e idades em dois vetores.

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

#define MAX 10

int main(void)
{
 char nome[MAX][41];
 int idade[MAX];
```

```
 int i, n;
 char ch;

 i = 0;
 do {
      printf("Nome: ");
      scanf("%s", nome[i]);          // entrada de um nome (somente um nome)
      fflush(stdin);                 // limpa o buffer de entrada
      printf("Idade: ");
      scanf("%d", &idade[i]);        // entrada de uma idade
      i++;
      printf("Continua [S/N]? ");
      do {
          ch = getchar();
         } while (ch != 'S' && ch != 's' && ch != 'N' && ch != 'n');
     } while (ch != 'N' && ch != 'n' && i < MAX);
 n = i - 1;                    // número de elementos
 for (i = 0; i <= n; i++)
      printf("| %-41s | %d | \n", nome[i], idade[i]);
}
```

Programa exemplo (24): O programa realiza a soma de duas matrizes (A e B) bidimensionais, gerando uma matriz resultante C.

```
#include <stdio.h>

#define MAX 10

int main(void)
{
 int a[MAX][MAX], b[MAX][MAX], c[MAX][MAX];
 int col,lin,j,m,n;

 printf("Informe a ORDEM da MATRIZ: (mxn)\n");
 do {
     printf("Número de linhas (m): ");
     scanf("%d",&m);
    } while (m < 1 || m > MAX);           // m de 1 à 10
 do {
     printf("Número de colunas (n): ");
     scanf("%d",&n);
    } while (n < 1 || n > MAX);           // n de 1 à 10
 for (lin = 1;lin <= m;lin++)
      for (col = 1;col <= n;col++)
          {
           printf("A[%d,%d] = ",lin,col);
           scanf("%d",&a[lin][col]);
           printf("B[%d,%d] = ",lin,col);
           scanf("%d",&b[lin][col]);
          }
 printf("\n");
 for (lin = 1;lin <= m;lin++)
      for (col = 1;col <= n;col++)
           {
            c[lin][col] = a[lin][col] + b[lin][col];
            printf("C[%d,%d] = %d\n",lin,col,c[lin][col]);
           }
}
```

11.9 - Classificação de dados ou ordenação (sort)

Para exemplificar melhor as variáveis do tipo vetor, abaixo são mostrados dois tipos de ordenação, também chamado **sort** (classificação de dados):

Programa exemplo (25): O programa classifica os nomes digitados pelo usuário.

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

#define QUANT 50

int main(void)
{
 char nome[QUANT][41];
 char temp[41];
 int i, j, n;
 char tecla;

 n = -1;
 do {
     n++;
     printf("Nome: ");
     scanf("%40s",&nome[n]);          // scanf – permite a entrada de um nome
     printf("Continua [S/N]? ");
     do {
         tecla = toupper(getchar());  // toupper: converte o caracter para maiúsculo
        } while (!strchr("SN", tecla)); // strchr: verifica se o caracter pertence string
     } while (tecla != 'N' && n < QUANT);

 for (i = 0; i <= n-1; i++)
     for (j = i+1; j <= n; j++)
         if ((strcmp(nome[i], nome[j])) > 0) // strcmp – permite comparar strings
            {
             strcpy(temp, nome[i]);          // strcpy – permite copiar strings
             strcpy(nome[i], nome[j]);
             strcpy(nome[j], temp);
            }
 printf("\nNomes ORDENADOS\n");
 for (i = 0; i <= n; i++)
      printf("Nome: %s\n", nome[i]);
}
```

Observação: As funções **strcpy**, **strcmp** e **strchr** são discutidas na próxima seção.

Programa exemplo (26): O programa utiliza um método de sort chamado ***bubble sort*** (método da bolha) para classificar nomes.

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

#define TRUE      !0
#define FALSE      0

#define QUANT 50

int main(void)
{
 char nome[QUANT][40];
 char temp[40], tecla;
 int i, k, n;
```

```
int sort;

n = 0;
do {
      printf("Nome: ");
      scanf("%s",nome[n]);                // entrada de um nome
      n++;
      printf(" Continua [S/N]? ");
      do {
          tecla = getchar();
         } while (!strchr("SsNn", tecla));
     } while (strchr("Ss", tecla) && n < QUANT);
n = n - 1;                            // n é o número de elementos
```

```
k = n;
do {
       sort = FALSE;
       for (i = 0; i < k; i++)
             if ((strcmp(nome[i],nome[i+1])) > 0)
               {
                strcpy(temp, nome[i]);
                strcpy(nome[i], nome[i+1]);
                strcpy(nome[i+1], temp);
                sort = TRUE;
              }
      k--;
    } while (sort);
```

Buble sort

```
printf("\nNomes ORDENADOS:\n");
for (i = 0; i <= n; i++)
    printf("Nome: %s\n", nome[i]);
}
```

11.10 - Lista de exercícios (vetores)

- ✓ Escreva um programa em C que recebe via teclado um conjunto de letras (máximo 20). Armazene todas as letras em um vetor (**letras**) até que o usuário digite um **ESC** (código 27). Logo após copie todas as letras (***em ordem inversa***) para outro vetor (**inverso**). Ao final imprima os dois vetores.

Exemplo:

Letra: **L**
Letra: **I**
Letra: **M**
Letra: **A**
Letra:
<**esc**>

LIMA
AMIL

	letras		inverso	
	0	'L'	0	'A'
	1	'I'	1	'M'
	2	'M'	2	'I'
	3	'A'	3	'L'

- ✓ Escreva um programa em C que recebe via teclado: **número de idades** (máximo 50) e as respectivas **idades**. Armazene todas as idades em um vetor (**idade**). Logo após a entrada de todas as idades, o programa deve receber via teclado: **idade para consulta**. O programa deve imprimir na tela, o número de idades **antes** da idade de consulta e o número de idades **depois** da idade de consulta.

Exemplo: Número de idades: **6** <**enter**>

Idade: **30** <**enter**>
Idade: **60** <**enter**>
Idade: **10** <**enter**>
Idade: **50** <**enter**>
Idade: **20** <**enter**>
Idade: **40** <**enter**>
Idade para consulta: **50** <**enter**>
Antes: **3**
Depois: **2**
Continua [S/N]? **n**

- ✓ Escreva um programa em C que recebe via teclado um conjunto de **números inteiros** (máximo 50). Armazene todos os números inteiros em um vetor até que o usuário digite 0 (zero). Logo após permita ao usuário consultar um número informando o seu **valor**. O programa deve imprimir na tela a **posição do número no vetor** ou **ERRO: Número não encontrado** (veja exemplos abaixo):

Exemplo:
Número: **50** <**enter**>
Número: **30** <**enter**>
Número: **20** <**enter**>
Número: **10** <**enter**>
Número: **40** <**enter**>
Número: **0** <**enter**>
Valor: **20** <**enter**>
Posição no vetor: **2**
Valor: **40** <**enter**>
Posição no vetor: **4**
Valor: **60** <**enter**>
ERRO: Número não encontrado
Valor: 0 <**enter**>

Observação: O programa termina quando o usuário digitar 0 (zero).

- ✓ Escreva um programa em C que recebe via teclado "n" **conceitos** (**A**, **B**, **C**, **D** e **E**) (máximo 25) até que o usuário digite **ESC**. Armazene todos os conceitos em um vetor (**conceito**). Imprima na tela o número de alunos: **aprovados** (A, B e C), **reprovados** (D) e os **infreqüentes** (E).

conceito

Exemplo:
Conceito: **B**
Conceito: **A**
Conceito: **E**
Conceito: **B**
Conceito: **D**
Conceito: **C**
Conceito: **A**
Conceito: **E**
Conceito: <**esc**>
4 Aprovado(s)
1 Reprovado(s)
3 Infreqüente (s)

0	‘B’
1	‘A’
2	‘E’
3	‘B’
4	‘D’
5	‘C’
6	‘A’
7	‘E’

- ✓ Escreva um programa em C que recebe via teclado “n” (máximo 50) **nomes** (máximo 80 letras). A entrada dos nomes termina quando o usuário digitar apenas <**enter**>. Logo após a entrada de todos os nomes o programa deve permitir a entrada via teclado de uma **letra**. O programa

deve imprimir na tela todos os nomes que começam com a letra especificada pelo usuário. O programa termina quanto o usuário digitar <**esc**> na entrada da letra (conforme exemplos abaixo):

Exemplo: Nome: **Paulo** <**enter**>
Nome: **Roberto** <**enter**>
Nome: **Renato** <**enter**>
Nome: **Pedro** <**enter**>
Nome: **Fabio** <**enter**>
Nome: <**enter**>
Letra: **R**
Nome: **Roberto**
Nome: **Renato**
Letra: **P**
Nome: **Paulo**
Nome: **Pedro**
Letra: **T**
Letra: <**esc**>

- ✓ Escreva um programa em C que recebe via teclado “n” (máximo 30) **nomes** (máximo 40 letras) e **idades**. A entrada dos dados termina quando o usuário digitar 'N' ou 'n' na pergunta "Continua [S/N]?". Logo após a entrada de todos os dados o programa deve imprimir na tela todos os nomes e idades desde o mais velho até o mais novo.

Exemplo: Nome: **Ana** <**enter**>
Idade: **12** <**enter**>
Continua [S/N]? **s**
Nome: **Beatriz** <**enter**>
Idade: **13** <**enter**>
Continua [S/N]? **s**
Nome: **Carla** <**enter**>
Idade: **14** <**enter**>
Continua [S/N]? **N**
Carla 14
Beatriz 13
Ana 12

12. Manipulação de strings

As funções **strcpy**, **strcmp**, **strcat** são necessárias pois uma string nada mais é do que um vetor de caracteres, ou seja, não se pode atribuir vários valores (ao mesmo tempo) para os elementos de um vetor. Isto só pode ser feito quando o vetor é declarado e inicializado ou um de cada vez.

Exemplo: char s[] = “UCPel”; // **correto**

ou

```
s[0] = ‘U’;       // correto – elemento por elemento
s[1] = ‘C’;
s[2] = ‘P’;
s[3] = ‘e’;
s[4] = ‘l’;
s[5] = NULL;
```

ou

```
s = “UCPel”;            // incorreto – erro GRAVE em C
```

12.1 - strcpy

A função **strcpy** (cópia de string) pré-definida do C que permite copiar uma string para outra ou inicializar uma string.

Sintaxe: char ***strcpy** (char *destino, const char *origem);
Prototype: string.h

Funcionamento: A *string* de origem é copiada para a *string* destino.

Programa exemplo (27): Os programas abaixo mostram como copiar caracteres para uma *string*.

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

int main(void)
{
 char erro[] = "Arquivo não Existe\n";

 printf("ERRO FATAL: %s",erro);
}
```

ou

```
#include <stdio.h>

int main(void)
{
 char erro[20];

 strcpy(erro,"Arquivo não existe\n");
 printf("ERRO FATAL: %s",erro);
}
```

12.2 - strcmp

A função **strcmp** (comparação de duas strings) pré-definida do C que permite comparar duas strings.

```
s1 é maior que s2       resultado > 0
s1 é menor que s2       resultado < 0
s1 é igual a s2         resultado == 0
s1 é diferente de s2   resultado != 0
```

Sintaxe: int **strcmp** (const char *s1, const char *s2);
Prototype: string.h

Programa exemplo (28): O programa compara duas *strings*.

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

int main(void)
{
 char s1[41],s2[41];

 printf("String 1: ");
 scanf("%s",s1);
 printf("String 2: ");
 scanf("%s",s2);
 if (strcmp(s1,s2) == 0)
     printf("String 1 é IGUAL a String 2");
 else
     if (strcmp(s1,s2) > 0)
         printf("String 1 é MAIOR a String 2");
     else
         printf("String 1 é MENOR a String 2");
}
```

12.3 - strcat

A função **strcat** (concatenação de duas strings) pré-definida do C que permite a concatenação de uma string no final da outra string.

Sintaxe: char ***strcat** (char *destino, const char *origem);
Prototype: string.h

Funcionamento: A *string* origem é copiado para o final da *string* destino.

Programa exemplo (29): O programa cópia e concatena caracteres a uma *string* resultando na palavra "Pelotas".

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

int main(void)
{
 char s1[] = "Pel",s2[] = "ota",s3[] = "s";
 char sr[15] = "";

 printf("%s\n",s1);
 printf("%s\n",s2);
 printf("%s\n",s3);
 strcpy(sr,s1);         // sr = "Pel"
```

```
 strcat(sr,s2);         // sr = "Pelot"
 strcat(sr,s3);         // sr = "Pelotas"
 printf("%s\n",sr);
}
```

12.4 - strlen

A função **strlen** (comprimento de uma string) pré-definida do C que retorna o comprimento de uma *string*, ou seja, a quantidade de caracteres que a *string* possui no momento.

Observação: O NULL não é contado.

Sintaxe: int **strlen** (const char *s);
Prototype: string.h

Programa exemplo (30): O programa imprime na tela a quantidade de caracteres de uma variável string, neste caso, a variável nome.

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

int main(void)
{
 char nome[41];

 printf("Qual o seu nome: ");
 scanf("%s",nome);
 printf("%s seu nome tem %d caracteres\n", nome, strlen(nome));
}
```

12.5 – strchr

A função **strchr** (Verifica se um caracter pertence a uma string) pré-definida do C que verifica se um caracter (**chr**) pertence a uma string (**str**).

Sintaxe: int **strchr** (const char *s, char ch);
Prototype: string.h

Programa exemplo (31): O programa imprime na tela o número de caracteres de um nome.

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

int main(void)
{
 char nome[41];
 char ch;

 do {
      printf("Qual o seu nome: ");
      fgets(nome,40,stdin);
      printf("Seu nome tem %d caracteres\n", strlen(nome));
      printf("Continua [S]im ou [N]ão ?");
      do {
            ch = getchar();
```

```
        } while (!strchr("SsNn",ch));
    } while (strchr("Ss",ch));
}
```

12.6 – Lista de exercícios (*strings*)

✓ Escreva um programa em C que recebe via teclado um **nome** (máximo 256 caracteres). Logo após a entrada do nome imprima: **número de letras maiúsculas**, **número de letras minúsculas**, **número de vogais** e o **número de consoantes**, conforme exemplo abaixo:

Exemplo: Nome: **Universidade Católica de Pelotas** <**enter**>
(**3**) Letras maiúsculas
(**26**) Letras minúsculas
(**14**) Vogais
(**15**) Consoantes

✓ Escreva um programa em C que recebe via teclado uma **palavra** (máximo 40 caracteres) e uma **letra**. Logo após a entrada do nome e da letra imprima **o número de letras que existe no nome** ou **ERRO: Não existe a letra (?) na palavra (?)**, conforme exemplo abaixo:

Exemplo: Palavra: **Luzzardi** <**enter**>
Letra: **z**
2 letra(s)
Continua [S]im ou [N]ão? **s**
Palavra: **Luzzardi** <**enter**>
Letra: **w**
ERRO: Não existe a letra (w) na palavra (Luzzardi)
Continua [S]im ou [N]ão? **N**

Observação: O programa deve ser encerrado quando o usuário digitar "N" ou "n" na pergunta: **Continua [S]im ou [N]ão?**.

✓ Escreva um programa em C que recebe via teclado uma **palavra** (máximo 20 letras) e uma **posição**. O programa deve imprimir na tela, a letra **antecessora**, a **letra** (da referida posição) e a letra **sucessora**, conforme exemplo abaixo:

Exemplo: Palavra: **Universidade** <**enter**>
Posição: **7** <**enter**>
Antecessora: **s**
Letra: **i**
Sucessora: **d**

OBSERVAÇÃO: O programa deve imprimir na tela as seguintes mensagens de erro, se for o caso: **Letra antecessora não existe**, **Letra sucessora não existe** ou **Posição inválida**.

✓ Escreva um programa em C que recebe via teclado um **nome** (máximo 80 letras). O programa deve imprimir, na tela, as **palavras** do nome **em ordem inversa, uma por linha**, conforme exemplo abaixo:

Exemplo: Nome: **Paulo Roberto Gomes Luzzardi** <**enter**>
Luzzardi
Gomes
Roberto

Paulo

- ✓ Escreva um programa em C que recebe via teclado um **nome** (máximo 60 caracteres). Logo após a entrada do nome o programa deve imprimir (EM LETRA MAIÚSCULA) o **sobrenome da pessoa**, conforme exemplos abaixo:

Exemplo:

Nome: **Paulo Roberto Gomes Luzzardi** <**enter**>
Sobrenome: **LUZZARDI**
Sair [S/N]? **N**
Nome: **Renato Souza** <**enter**>
Sobrenome: **SOUZA**
Sair [S/N]? **s**

Observação: O programa termina quando o usuário digitar 'S' ou 's' na pergunta: **Sair [S/N]?**

- ✓ Escreva um programa em C que recebe via teclado um **nome** (máximo 80 caracteres). Logo após a entrada do nome o programa deve imprimir na tela: **sobrenome**, **primeiro nome** e **demais nomes abreviados**, conforme exe-mplos abaixo:

Exemplo:

Nome: **Paulo Roberto Gomes Luzzardi** <**enter**>
Autor: **Luzzardi, Paulo R. G.**
Sair [S/N]? **N**
Nome: **Renato Lima Souza** <**enter**>
Autor: **Souza, Renato L.**
Sair [S/N]? **s**

Observação: O programa termina quando o usuário digitar 'S' ou 's' na pergunta: **Sair [S/N]?**

- ✓ Escreva um programa em C que recebe via teclado o nome de um **estado** (máximo 80 caracteres). Logo após a entrada do nome do estado imprima: a **sigla** do estado (2 letras maiúsculas), conforme exemplos abaixo:

Exemplo:

Estado: **Rio Grande do Sul** <**enter**>
Sigla: **RS**
Estado: **são paulo** <**enter**>
Sigla: **SP**
Estado: **rio de janeiro** <**enter**>
Sigla: **RJ**
Estado: <**enter**>

Observação: O programa encerra quando o usuário digitar apenas <**enter**> na entrada do nome do estado.

- ✓ Escreva um programa em C que recebe via teclado uma **password** (senha – máximo 8 dígitos). Na entrada da senha deve ser exibido na tela um asterisco (*) para cada letra digitada. Quando o usuário teclar <**enter**> (ou digitar 8 dígitos) o programa deve imprimir na tela a senha digitada.

Exemplo:

Password: ******* <**enter**>
Senha digitada: **pelotas**
Sair [S/N]? **s**

Observação: O programa deve ser encerrado quando o usuário digitar ‘S’ ou ‘s’ na pergunta: **Sair [S/N]?**.

- ✓ Escreva um programa em C que recebe via teclado uma **palavra** (máximo 20 caracteres), **início** e **fim**. Logo após a entrada de todos os dados imprima a **string resultante** ou **ERRO: Fim inválido** ou **Início inválido**, conforme exem-plos abaixo:

Exemplo:

```
Palavra: universidade <enter>
Início: 7 <enter>
Fim: 11 <enter>
String resultante: idade
Continua [S/N]? s
Palavra: eletricidade <enter>
Início: 7 <enter>
Fim: 15 <enter>
ERRO: Fim Inválido
Continua [S/N]? N
```

Observação: O programa termina quando o usuário digitar ‘N’ ou ‘n’ na pergunta: **Continua [S/N]?**.

13. Funções definidas pelo programador

C permite que o programador crie e utilize suas próprias funções.

Forma Geral:

```
tipo_do_retorno nome_da_função (parâmetros)
tipo_dado_base parâmetros;
{
 tipo_dado_base variáveis;          // definição das variáveis locais

 corpo da função;

 return();
}
```

ou

```
tipo_do_retorno nome (tipo_dado_base parâmetros)
{
 tipo_dado_base variáveis;

 corpo da função;

 return();
}
```

tipo_do_retorno: Especifica o tipo de dado do retorno da função. O retorno da função é feita pelo comando ***return*** (valor).

parâmetros: É uma lista, separada por vírgulas, com os nomes das variáveis (e seus tipos) que receberão os argumentos quando a função for chamada.

Observação: O tipo default de uma função é **int**, ou seja, se não for especificado o tipo da função o compilador assume **int**.

Função procedural: É um tipo especial de função que não possui retorno, ou seja, é simplesmente um procedimento. Uma função deste tipo é **void**.

Exemplo:

```
int random (int num)
{
 int x;

 x = rand() % num;
 return(x);
}
```

Chamada da função: `n = random(256);` `// retorna um número aleatório entre 0 e 255`

Programa exemplo (32): O programa possui uma função que calcula o inverso 1/x.

```
#include <stdio.h>

float inverso (float x);            // protótipo da função
```

```
int main(void)
{
 float inv, resp;

 printf("x: ");
 scanf("%f",&x);
 inv = inverso (x);            // chamada da função inverso
 printf("Inverso = %.2f\n", inv);
}

// ---------------------------- Função definida pelo programador

float inverso (float x)
{
 float i;

 i = (float) 1 / x;
 return(i);
}
```

ou

```
float inverso (x)
float x;
{
 return( (float) 1 / x );
}
```

13.1 - Valores de retorno

Todas as funções, exceto aquelas que são declaradas como sendo do tipo **void**, devolvem um valor. O valor é devolvido (retornado) pela função através do comando **return**.

Normalmente são escritas três tipos de funções:

a) Funções que efetuam operações com os parâmetros e retornam um valor com base nas operações.

Programa exemplo (33): O programa calcula e imprime na tela o valor da potência x^y.

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

float potencia (float x, int y);

int main(void)
{
 float base,resp;
 int expoente;

 printf("Base: ");
 scanf("%f",&base);
 printf("Expoente: ");
 scanf("%d",&expoente);
 resp = potencia(base,expoente);          // chamada da função potencia
 printf("Potencia = %7.2f\n",resp);
```

```
}

// ----------------------------- Função definida pelo programador

float potencia (float x, int y)
{
 float valor;

 valor = exp ( log (x ) * y );
 return(valor);
}
```

b) Funções que manipulam informações e retornam um valor que simplesmente indica o sucesso ou o fracasso da manipulação.

Programa exemplo (34): O programa calcula e verifica o determinante de uma equação de segundo grau.

```
#include <stdio.h>

int verifica_determinante (float a, float b, float c);

int main(void)
{
 float a, b, c;
 int retorno;

 printf("a = ");
 scanf("%f",&a);
 printf("b = ");
 scanf("%f",&b);
 printf("c = ");
 scanf("%f",&c);
 retorno = verifica_determinante(a,b,c);
 if (retorno == 0)
     printf("Determinante ZERO\n");
 else
     if (retorno > 0)
         printf("Determinante POSITIVO\n");
     else
         printf("Determinante NEGATIVO\n");
}

// ---------------------------- função definida pelo programador

int verifica_determinante (float a, float b, float c)
{
 float det;

 det = b * b - 4 * a * c;
 if (det == 0)
    return(0);
 else
     if (det > 0)
         return(1);
     else
         return(-1);
}
```

c) Funções que não retornam nenhum valor, ou seja, são puramente procedimentos.

Programa exemplo (35): O programa possui uma função que limpa toda a tela em modo texto.

```
#include <stdio.h>

void limpa_tela (void);

int main(void)
{
 limpa_tela();
}

// ---------------------------- função definida pelo programador

void limpa_tela (void)
{
 int c, l;

 for (l = 1; l <= 25; l++)
     {
      for (c = 1; c <= 80; c++)
          printf("%c",32);          // 32 é o código do caracter espaço
      printf("\n");
     }
}
```

13.2 - Passagem de parâmetros por valor

Forma de chamada de uma função onde o valor do argumento é apenas copiado para o parâmetro formal da função. Portanto, alterações feitas nos parâmetros não terão efeito nas variáveis utilizadas para chamá-la.

Programa exemplo (36): O programa possui uma função que desenha um retângulo na tela.

```
#include <stdio.h>

void desenha_retangulo (int ci, int li, int cf, int lf)
{
 int c, l;

 for (l = li; l <= lf; l++)
     {
      for (c = ci; c <= cf; c++)
          printf("#");
      printf("\n");
     }
}

int main(void)
{
 desenha_retangulo(1, 1, 20, 10);
}
```

Onde: ci -> coluna inicial
li -> linha inicial
cf -> coluna final
lf -> linha final

Atenção: Os parâmetros da função recebem, respectivamente: **ci = 1**, **li = 1**, **cf = 20** e **lf = 10**.

13.3 - Passagem de parâmetros por referência

Forma de chamada de uma função onde o endereço do argumento é passado como parâmetro. Significa que as alterações feitas nos parâmetros afetarão a variável utilizada para chamar a função.

Programa exemplo (37): O programa tem uma função que troca o valor de duas variáveis.

```
#include <stdio.h>

void troca (int *x, int *y)          /* x e y são ponteiros */
{
 int temp;

 temp = *x;
 *x = *y;
 *y = temp;
}

int main(void)
{
 int a, b;

 printf("a = ");                            // imagine que o usuário digitou 3
 scanf("%d",&a);
 printf("b = ");
 scanf("%d",&b);                            // imagine que o usuário digitou 4
 printf("a = %d | b = %d\n", a, b);         // foi impresso na tela a = 3 | b = 4
 troca (&a,&b);
 printf("a = %d | b = %d\n", a, b);         // foi impresso na tela a = 4 | b = 3
}
```

Atenção: Os parâmetros da função recebem, respectivamente: **x -> &a** e **y -> &b**.

13.4 - Funções que devolvem valores não-inteiros

Todas as funções que devolvem valores não-inteiros devem ter seu tipo de retorno declarado.

tipo_do_retorno nome_da_função (tipo_dado_base parâmetros);

Programa exemplo (38): O programa calcula e imprime na tela a divisão de dois valores.

```
#include <stdio.h>

float divisao (int x, int y);

int main(void)
{
 int x, y;
 float resposta;
```

```
 printf("x = ");
 scanf("%d",&x);
 printf("y = ");
 scanf("%d",&y);
 resposta = divisao ( x, y);
 printf("Divisão = %7.2f\n",resposta);
}

// ---------------------------- Função definida pelo programador

float divisao (int x, int y)
{
 return( (float) x / y );
}
```

13.5 - Argumentos argc e argv do main

A função **main** possui dois argumentos **argc** e **argv** intrínsecos utilizados para receber parâmetros da linha de comando do DOS (Sistema Operacional).

argc - contém o número de argumentos na linha de comando.
argv - ponteiro para uma matriz (2D) de caracteres (vetor de strings).
Programa exemplo (39): O programa recebe parâmetros do Sistema Operacional (uma palavra qualquer) e imprime a palavra em sentido inverso. O programa é compilado e executado em um terminal do Sistema Operacional Linux da seguinte forma:

Execução pela linha de comandos (terminal): $ inverte pelotas <**enter**>

Resultado na tela: *satolep*

Programa recebe:

```
argc = 2
argv[0] = "inverte"
agv[1] = "pelotas"
```

```
// inverte.c              // este programa deve obrigatoriamente se chamar inverte.c
                          // depois de compilado será gerado o executável inverte
#include <stdio.h>
#include <string.h>

int main (int argc, char *argv[])
{
 int i, n;

 if (argc != 2)
     printf("Sintaxe: INVERTE <palavra>\n");
 else
     {
      n = strlen(argv[1]);
      for (i = n-1; i >= 0; i--)
          printf("%c", argv[1][i]);
      printf("\n");
     }
}
```

Programa exemplo (40): O programa recebe parâmetros pelo Sistema Operacional (conjunto de caracteres) e ordena (coloca em ordem alfabética), imprimindo-os a seguir.

Execução pela linha de comandos: $ ordena dbeacgf <**enter**>

Resultado na tela: abcdefg

Programa recebe:

```
argc = 2
argv[0] = "ordena"
agv[1] = "dbeacgf"
```

```
// ordena.c

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

int main (int argc, char *argv[])
{
 int i, j;
 int n_car;
 char temp;

 if (argc != 2)
     printf("Sintaxe: ORDENA <palavra>\n");
 else
     {
      n_car = strlen(argv[1]);
      for (i = 0; i < n_car-1; i++)
          for (j = i+1; j < n_car; j++)
              if (argv[1][i] > argv[1][j])
                 {
                  temp = argv[1][i];
                  argv[1][i] = argv[1][j];
                  argv[1][j] = temp;
                 }
       for (i = 0 ; i < n_car ; i++)
           printf("%c",argv[1][i]);
       printf("\n");
      }
}
```

13.6 - Recursividade

Uma função é recursiva, se esta, fizer uma chamada a si própria.

Programa exemplo (41): Este programa calcula o **fatorial** de um número recursivamente.

```
#include <stdio.h>

long int fatorial (unsigned int n);

int main (void)
{
```

```
 unsigned int n;
 long unsigned int fat;

 do {
       printf("Valor [max -> 19] = ");
       scanf("%d",&n);
      } while (n < 0 || n > 19);
  fat = fatorial (n);
  printf("Fatorial de [%d] = [%d]\n", n, fat);
}

long int fatorial (unsigned int n)
{
 long int resp, valor;

 if (n <= 0)
    return (0);
 if (n == 1)
    return (1);
 valor = fatorial (n - 1);
 resp = valor * n;
 return (resp);
}
```

13.7 - Lista de exercícios (funções)

✓ Escreva em C a função **PALAVRAS**. A função recebe uma string (**nome**) e retorna o **número de palavras do nome** (veja exemplo abaixo):

```
#include <stdio.h>

_______________ PALAVRAS( ____________________ );

int main(void)
{
 char nome[41];
 int n;

 printf("Nome: ");
 fgets(nome,40,stdin);
 n = PALAVRAS(nome);
 printf("Seu nome tem %d palavra(s)\n", n);
}
```

Exemplo:

Nome: **Paulo Roberto Gomes Luzzardi** <**enter**>
Seu nome tem **4** palavra(s)

Observação: Não utilizar a função **strlen** do Turbo C.

✓ Escreva em C a função **VERIFICA_DATA**. A função recebe uma string (**data**, no formato: dd/mm/aaaa) e devolve via parâmetros: **dia**, **mês** e **ano**. A função retorna: (**1**) se o dia for inválido, (**2**) se o mês for inválido, (**3**) se o ano for inválido, (**4**) formato inválido e (**5**) se a data estiver correta.

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

_______________ VERIFICA_DATA ( ____________________ );
```

```
int main (void)
{
 char data[11];
 int dia, mes, ano, erro;

 printf("Data [dd/mm/aaaa]: ");
 scanf("%10s", data);
 erro = VERIFICA_DATA(data, &dia, &mes, &ano);
 if (erro == 1)
     printf("ERRO: Dia inválido\n");
 else
     if (erro == 2)
         printf("ERRO: Mês inválido\n");
     else
         if (erro == 3)
            printf("ERRO: Formato inválido\n");
         else
            if (erro == 4)
                  printf("ERRO: Ano Inválido\n");
            else
                {
                 printf("Dia: %d\n", dia);
                 printf("Mês: %d\n", mes);
                 printf("Ano: %d\n", ano);
                }
}
```

Exemplo:

```
Data [dd/mm/aaaa]: 11/07/2002 <enter>
Dia: 11
Mês: 07
Ano: 2002
```

Valores Válidos:

```
Dia [1..31]
Mês [1..12]
Ano [2000.. 2999]
```

✓ Escreva em C a função **PREENCHE**. A função recebe: **número de caracteres** (n) e **caracter** (ch). A função deve imprimir na tela, **n** caracteres **ch**.

```
#include <stdio.h>
____________________ PREENCHE ( ____________________ );

int main (void)
{
 PREENCHE(5,'#');       |   #####
}
```

✓ Escreva em C a função **GERA_STRING**. A função recebe via parâmetros **número de caracteres** (n) e **caracter** (ch). A função devolve na variável **s**, **n** caracteres **ch**.

```
#include <stdio.h>
___________________ GERA_STRING ( _____________ );

int main (void)
{
 char s[10];

 GERA_STRING(5,'#',s);        // s ficará com "#####"
 printf("%s", s);
}
```

✓ Escreva em C a função **VERIFICA_QUADRANTE**. A função recebe um valor para **x** e um valor para **y** e retorna o **número do quadrante** (1,2,3 ou 4).

```
#include <stdio.h>
___________________ VERIFICA_QUADRANTE ( _____________ );
```

```
int main (void)
{
 int x, y, n;

 printf("x = ");
 scanf("%d", &x);
 printf("y = ");
 scanf("%d", &y);
 n = VERIFICA_QUADRANTE(x, y);
 printf("Quadrante: %d\n", n);
}
```

Quadrantes

```
          |
          |
   2º     |     1º
          |
 ------------------------
          |
   3º     |     4º
          |
          |
```

✓ Escreva a função: **final_da_placa**. A função recebe uma placa de automóvel (placa) no formato: xxx9999 e retorna o último dígito da placa.

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

int main (void)
{
 char placa[8];
 int final;

 printf("Qual a placa de seu carro [xxx9999]: ");
 scanf("%7s", placa);
 final = final_da_placa(placa);
 printf("Final da Placa é: %d\n", final);
}
```

✓ Escreva a função: **VOGAIS.** A função recebe uma **string** (nome) e retorna a quantidade de **vogais** da string.

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

int main (void)
{
 char nome[41];
 int vogais;

 printf("Nome: ");
 fgets(nome,40,stdin);
 vogais = VOGAIS(nome);
 printf("Vogais: %d\n", vogais);
}
```

✓ Escreva a função: **HIPOTENUSA**. A função recebe o cateto adjacente (b) e o cateto oposto (a) e retorna o valor da hipotenusa dado pela seguinte fórmula:

Fórmula: $h^2 = a^2 + b^2$

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

int main (void)
{
```

```
 float a, b, h;

 printf("Cateto Adjacente: ");
 scanf("%f", &b);
 printf("Cateto Oposto: ");
 scanf("%f", &a);
 h = HIPOTENUSA(a, b);
 printf("Hipotenusa: %.2f\n", h);
}
```

✓ Escreva em C a função **CONTA**. A função recebe uma string (**nome**) e devolve via parâmetros: **número letras maiúsculas** e o **número letras minúsculas**. A função retorna o **total de letras do nome** (veja exemplo abaixo).

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

_______________ CONTA ( ____________________ );

int main (void)
{
 char nome[41];
 int n, maiusculas, minusculas;

 clrscr();
 printf("Nome: ");
 fgets(nome,40,stdin);
 n = CONTA(nome, &maiusculas, &minusculas);
 printf("Maiúsculas: %d\n", maiusculas);
 printf("Minúsculas: %d\n", minusculas);
 printf("Total de letras: %d\n", n);
 getch();
}
```

Exemplo:

```
Nome: Paulo Roberto Gomes Luzzardi <enter>
Maiúsculas: 4
Minúsculas: 21
Total de letras: 25
```

✓ Escreva em C a função **REAJUSTE**. A função recebe o valor do **salário** e o **índice** de reajuste e retorna o **salário atualizado**.

```
#include <stdio.h>
___________________ REAJUSTE ( _______________ );

int main (void)
{
 float salario, indice, sal;

 printf("Salário (R$):  ");
 scanf("%f", &salario);
 printf("Índice de Reajuste: ");
 scanf("%f", &indice);
 sal = REAJUSTE(salario, indice);
 printf("Salário Atualizado (R$): %.2f\n", sal);
}
```

Exemplo:

```
Salário (R$):  1000 <enter>
Índice de Reajuste: 10 <enter>
Salário Atualizado (R$): 1100
```

✓ Escreva em C a função **CENTRALIZA**. A função recebe uma **mensagem**. A função deve imprimir na tela a mensagem centralizada na tela.

```
#include <stdio.h>
#include <string.h>
____________________ CENTRALIZA ( ____________________ );
```

```
int main (void)
{
 CENTRALIZA(”Jogo de Damas”);
}
```

✓ Escreva em C a função **HIPOTENUSA**. A função recebe o valor da **base** e da **área** de um triângulo. A função deve devolver na variável altura a **altura do triângulo**. A função deve retornar também o **valor da hipotenusa**.

```
#include <stdio.h>
____________________ HIPOTENUSA ( _________________ );

int main (void)
{
 float base, area, h, altura
```

$$área = \frac{base \cdot altura}{2}$$

```
 printf(“Base:  “);
 scanf(“%f”, &base);
 printf(“Área do Círculo: “);
 scanf(“%f”, &area);
 h = HIPOTENUSA(base, area, &altura);
 printf(“Hipotenusa: %.1f\n”, h);
 printf(“Altura: %.1f\n”, altura);
}
```

$$\boxed{hipotenusa^2 = base^2 + altura^2}$$

✓ Escreva em C a função **VERIFICA**. A função recebe um número inteiro (**n**). A função deve devolver na variável resto o **resto inteiro da divisão**. A função deve retornar se o número (**n**) é par (**1**) ou ímpar (**0**).

```
#include <stdio.h>
_____________________ VERIFICA ( ______________________ );

int main (void)
{
 int n, resto;

 printf(“Número: “);
 scanf(“%d”, &n);
 if (VERIFICA(n, &resto))
     printf(“Par\n”);
 else
     printf(“Ímpar\n”);
 printf(“Resto Inteiro da Divisão: %d\n”, resto);
}
```

✓ Escreva as seguintes funções: **STRCPY** (copia strings) e **STRCAT** (concatena strings)

```
#include <stdio.h>
_____________________ STRCPY (__________________________________)
_____________________ STRCAT (__________________________________)

int main (void)
{
 char s[] = “Liber”, r[] = “dade”,t[10];

 STRCPY(t, s);           // função copia s para t -> “Liber”
 STRCAT(t, r);           // insere r no final de t -> “Liberdade”
```

```
 printf("%s\n", t);      // t -> "liberdade"
}
```

14. Ponteiros

Um ponteiro é uma variável que contém um endereço de memória. O endereço pode ser a localização de uma ou mais variáveis na memória RAM ou qualquer outro endereço da memória RAM, como por exemplo, a memória da tela.

Figura 5: Representação de um ponteiro na memória

Exemplo:

```
#include <stdio.h>

int main(void)
{
 char x = 65;
 char *p;

 p = &x;                                    // p recebe o endereço de x (&x)
 printf("Valor de x...: %d\n", *p);         // *p valor é 65
 printf("Caracter ....: %c\n", *p);         // caracter 'A'
 printf("Endereço de x: %p\n", p);          // endereço de x
 printf("Endereço de p: %p\n",&p);          // endereço do ponteiro p
 printf("Sizeof de p: %d\n",sizeof(p));     // ponteiro ocupa 4 bytes
}
```

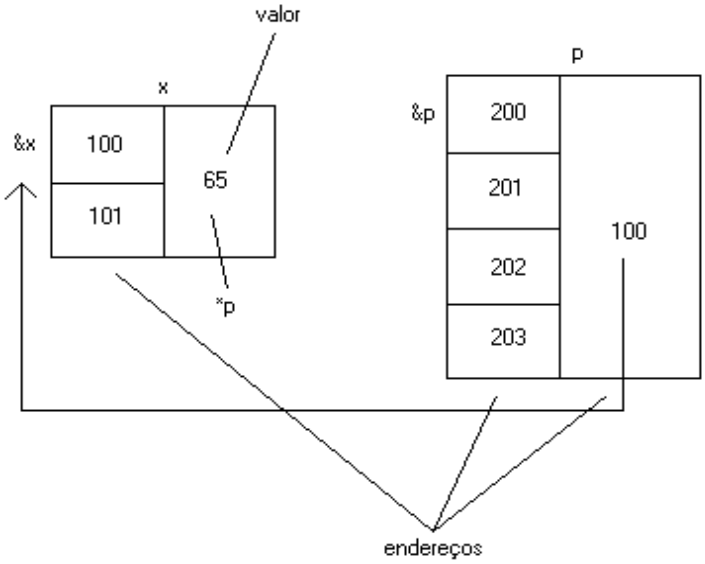


Figura 6: Endereçamento de um ponteiro

Resumo:

```
x -> 65
&x -> 100
p -> &x -> 100
*p -> 65
&p -> 200
```

Observação: Um ponteiro no Linux ocupa 4 bytes na memória RAM.

14.1 - Variáveis ponteiros

Definição:

```
tipo_dado_base *nome_do_ponteiro;
```

tipo_dado_base: qualquer tipo básico válido em C.
nome_da_ponteiro: nome da variável que representa o ponteiro.

O tipo de dados do ponteiro define para que tipos de variáveis o ponteiro pode apontar e qual o tamanho ocupado na memória por estas variáveis.

14.2 - Operadores de ponteiros

& endereço de memória do operando.
***** conteúdo do endereço apontado pelo ponteiro.

Exemplo: ponteiro = &variável;

Logo: variável = *ponteiro;

Observação: Como C sabe quantos bytes **copiar** para a variável apontada pelo ponteiro?

Resposta: O tipo_dado_base do ponteiro determina o tipo de dado que o ponteiro está apontando.

14.3 - Expressões com ponteiros

14.3.1 - Atribuições com ponteiros

Pode-se atribuir o valor (endereço) de um ponteiro para outro ponteiro.

Programa exemplo (42): O programa mostra atribuições utilizando dois ponteiros.

```
#include <stdio.h>

int main (void)
{
 int x = 7;
 int *p1, *p2;

 p1 = &x;                      // p1 recebe o endereço de x
 p2 = p1;                      // p2 recebe o endereço de x
 printf("%p\n", &x);           // impresso endereço de x
 printf("%p\n", p1);           // impresso endereço de x
 printf("%p\n", p2);           // impresso endereço de x
 printf("%d\n", *p1);          // impresso valor de x (conteúdo de p1)
 printf("%c\n", *p1);          // impresso caracter 7 (Bell - conteúdo de p1)
}
```

14.3.2 - Aritmética com ponteiros

14.3.2.1 - Incremento (++)

Faz com que o ponteiro aponte para a localização de memória do próximo elemento de seu tipo_dado_base.

Exemplo:

```
p++;        /* ponteiro aponta para o próximo elemento */
(*p)++;     /* conteúdo do ponteiro é incrementado */
```

Programa exemplo (43): O programa mostra um ponteiro apontando para os elementos de um vetor. Isto é feito através da aritmética de ponteiros.

```
#include <stdio.h>

int main (void)
{
 char x [10] = { 65, 66, 67, 68, 69, 70, 71, 72, 73, 74 };
 char *p;
 int i;

 p = &x [0];
 for (i = 0; i <= 9; i++)
     {
      printf("Endereço: %p | Valor: x [%d] = %c\n", p, i, *p);
      p++;
     }
}
```

Resultado na tela:

```
Endereço: 0xbfb70886 | Valor: x[0] = A
Endereço: 0xbfb70887 | Valor: x[1] = B
Endereço: 0xbfb70888 | Valor: x[2] = C
Endereço: 0xbfb70889 | Valor: x[3] = D
Endereço: 0xbfb7088a | Valor: x[4] = E
Endereço: 0xbfb7088b | Valor: x[5] = F
Endereço: 0xbfb7088c | Valor: x[6] = G
Endereço: 0xbfb7088d | Valor: x[7] = H
Endereço: 0xbfb7088e | Valor: x[8] = I
Endereço: 0xbfb7088f | Valor: x[9] = J
```

Programa exemplo (44): O programa mostra um ponteiro apontando para os elementos de um vetor. Isto é feito através da aritmética de ponteiros.

```
#include <stdio.h>

int main(void)
{
 int x [10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
 int i,*p;

 p = &x[0];
 for (i = 0; i <= 9; i++)
     {
      printf("Endereço: %p | Valor: x [%d] = %d\n", p, i, *p);
      p++;
     }
}
```

Resultado na Tela:

```
Endereço: 0xbfc258c0 | Valor: x[0] = 0
Endereço: 0xbfc258c4 | Valor: x[1] = 1
```

```
Endereço: 0xbfc258c8 | Valor: x[2] = 2
Endereço: 0xbfc258cc | Valor: x[3] = 3
Endereço: 0xbfc258c0 | Valor: x[4] = 4
Endereço: 0xbfc258d0 | Valor: x[5] = 5
Endereço: 0xbfc258d4 | Valor: x[6] = 6
Endereço: 0xbfc258d8 | Valor: x[7] = 7
Endereço: 0xbfc258dc | Valor: x[8] = 8
Endereço: 0xbfc258e4 | Valor: x[9] = 9
```

14.3.2.2 - Decremento (--)

Faz com que o ponteiro aponte para a localização do elemento anterior.

Exemplo:

```
p--;          // ponteiro aponta para o elemento anterior
(*p)++;       // conteúdo do ponteiro é incrementado
```

14.3.3 - Soma (+) e subtração (-)

Exemplo: p = p + 9;

Faz com que o ponteiro **p** aponte para o nono (9º) elemento do tipo_dado_base, após aquele que o ponteiro estava apontando no momento.

Observação: Somente pode-se somar e subtrair números inteiros a ponteiros.

14.3.4 - Comparação de ponteiros

É possível comparar dois ponteiros através utilizando os operadores relacionais.

Exemplo:

```
if (p < q)
   printf("Endereço de p é menor do que q\n");
```

14.4 - Ponteiros e vetores

Em C, o nome de um vetor sem índice é o endereço do primeiro elementos da matriz.

Exemplo:

```
#include <stdio.h>

int main (void)
{
 int x[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
 int *p;

 p = &x[0];        // é igual a p = x
   :
}
```

Para acessar o quinto (5º) elemento podemos escrever:

```
x[4];       ou          *(p+4);
```

Observação: Aritmética de ponteiros pode ser mais rápida que a indexação de vetores e matrizes.

14.4.1 - Indexando um ponteiro

Exemplo:

```
#include <stdio.h>

int main (void)
{
 int x[5] = { 0, 1, 2, 3, 4}
 int *p,t;

 p = x;                        // igual a p = &x[0]
 for (t = 0; t <= 4; t++)
      printf("%d\n", p[t]);
}
```

14.4.2 - Ponteiros e strings

Como o nome de um vetor sem índice é um ponteiro que aponta para o primeiro elemento do vetor, quando utiliza-se funções que recebem *strings* como parâmetros, estas recebem apenas um ponteiro para a *string* e não o valor real da string em si, ou seja, sempre a passagem de parâmetro de uma string é por referência.

Programa exemplo (45): O programa compara duas strings.

```
#include <stdio.h>

int compare (char *s1, char *s2);

int main (void)
{
 char s1[41], s2[41];

 printf("String 1: ");
 fgets(s1,40,stdin);
 printf("String 2: ");
 fgets(s2,40,stdin);
 if (compare (s1,s2) == 0)                // s1 igual s2 == 0
     printf("s1 é igual a s2\n");
 else
     printf("s1 é diferente de s2\n");
}

int compare (char *s1, char *s2)
{
 while (*s1)                   // quando o *s1 for igual '\0' termina o laço
       if (*s1-*s2)
          return (*s1-*s2);
       else
           {
            s1++;
            s2++;
           }
 return ('\0');
```

```
}
```

14.4.3 - Obtendo o endereço de um elemento de um vetor

```
p = &x[2];
```

14.4.4. Vetores de ponteiros

Declaração de um vetor de ponteiros de inteiros de tamanho 10.

```
int *x[10];
```

Atribuição do endereço de uma variável ao terceiro elemento da matriz de ponteiros:

`x[2] = &variável;` **Logo**: Para obter o valor da variável, utiliza-se: `*x[2];`

14.5 - Ponteiros para ponteiros

Uma matriz de ponteiros é igual a apontar ponteiros para ponteiros. Um ponteiro para um ponteiro é uma forma de indireção múltipla.

Programa exemplo (46): O programa utiliza uma variável ponteiro para ponteiro.

```
#include <stdio.h>

int main (void)
{
 int x;
 int *p;
 int **t;

 x = 10;
 p = &x;
 t = &p;
 printf("%d\n", **t);                // imprime o valor de x
}
```

14.6 - Inicialização de ponteiros

Após declarar um ponteiro, e antes de haver uma atribuição de um valor inicial, o ponteiro terá um endereço desconhecido (lixo), ou seja, nunca se deve utilizar um ponteiro antes de atribuir-lhe um valor (endereço).

Pode-se inicializar um ponteiro para caracteres (*string*) da seguinte forma:

Exemplo:

```
int main (void)
{
 char *p = "Paola";
 char *erro = "Falta de memória\n";
```

14.7 - Alocação dinâmica de memória

Permite alocar e liberar uma área de memória para variáveis durante a execução de um programa. Qualquer outro tipo de variável, que não seja um ponteiro, deve ser alocada estaticamente, ou seja, a variável não pode ocupar mais espaço do que foi declarado.

Exemplo:

```
int x [10];
// um inteiro ocupa 2 bytes
// logo 10 inteiros ocupam 20 bytes na memória RAM durante toda a execução do programa
```

14.7.1 – malloc

A função **malloc** (**m**emory **alloc**ation) permite alocar uma porção de memória dinamicamente.

Sintaxe: void ***malloc** (int número_de_bytes);
Prototype: stdlib.h

A função **malloc** reserva espaço na memória RAM (**aloca**) a quantidade de bytes especificada pelo parâmetro da função (número_de_bytes), devolvendo um ponteiro do tipo **void*** apontando para o primeiro byte alocado.

O tipo **void*** representa um ponteiro sem tipo que deve ser "tipado" de acordo com o tipo de dado base do ponteiro. Isto pode ser feito utilizando-se um **cast**.

Quando não houver memória suficiente para alocar o ponteiro, a função **malloc** devolve um ponteiro nulo (**NULL**).

Exemplo:

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

int main (void)
{
 int *p, n;

 printf("Quantidade de elementos: ");
 scanf("%d",&n);
 p = (int *) malloc( n * sizeof (int) );
 if (p == NULL)                          // ou   if (!p)
     printf("ERRO FATAL: Falta de memória\n");
 else
      {
       printf("Ok, memória alocada\n");
       printf("Endereço reservado: %p\n", p);
       printf("Quantidade de elementos alocados: %d\n", n);
       printf("Quantidade de bytes alocados: %d\n", n * sizeof(int));
       free(p);   // veja item 14.7.2
      }
}
```

Resultado do Programa:

```
Quantidade de elementos: 5
Ok, memória alocada
Endereço reservado: 0x804a008
Quantidade de elementos alocados: 5
```

```
Quantidade de bytes alocados: 20
```

14.7.2 – free

A função **free** (livre) permite liberar a porção de memória alocada pela função **malloc**.

Sintaxe: void **free** (void *p);
Prototype: stdlib.h

A função **free** libera a área de memória alocada pela função **malloc**.

Programa exemplo (47): O ponteiro **p** aponta para uma região da memória com 80 bytes reservados (alocados), ou seja, 40 inteiros.

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

int main(void)
{
 int *p, t;

 p = (int *) malloc( 40 * sizeof (int) );
 if (!p)
     printf("Erro Fatal: Falta de memória\n");
 else
     {
      for (t = 0; t <= 39;++t)
          *(p+t) = t;
      for (t = 0; t <= 39; ++t)
           printf("%d\n", *(p+t));
      free(p);
     }
}
```

15. Entrada e saída em disco

Existem dois sistemas de arquivo definidos em C. O primeiro, definido pelo padrão ANSI e UNIX chamado sistema de arquivo **bufferizado** (formatado ou de alto nível) e o segundo, é definido apenas pelo UNIX chamado sistema de arquivos **não-bufferizado** (não-formatado ou de baixo nível).

15.1 - Fila de bytes (stream)

A fila de bytes é um dispositivo lógico de entrada ou saída de dados, independente do dispositivo real. Um arquivo (dispositivo externo) deve ser associado a uma fila de bytes.

C utiliza 5 filas de texto pré-definidas, são elas:

Tabela 18: Filas de texto

Fila	**Função da fila**
stdin	Entrada padrão (**st**an**d**ard **in**put)
stdout	Saída padrão (**st**an**d**ard **out**put)
stderr	Erro padrão (**st**an**d**ard **err**or)
stdprn	Saída para impressora (**st**an**d**ard **pr**i**n**ter)
stdaux	Saída auxiliar (**st**an**d**ard **aux**iliary)

15.1.1 - Filas de texto

Uma fila de texto é uma seqüência de caracteres organizada em linhas. Cada linha é finalizada por um caracter '**\n**'.

Pode ocorrer certas traduções de caracteres, tal como: '**\n**' ser convertida em CR (13) + LF (10). Dessa forma, pode não haver correspondência de 1 para 1 entre os caracteres que o computador escreve (lê) e aqueles no dispositivo externo.

CR – **C**arriage **R**eturn (retorno do carro - cursor)
LF – **L**ine **F**eed (avanço de linha)

CR + LF = <**enter**>

Exemplo de um arquivo texto:

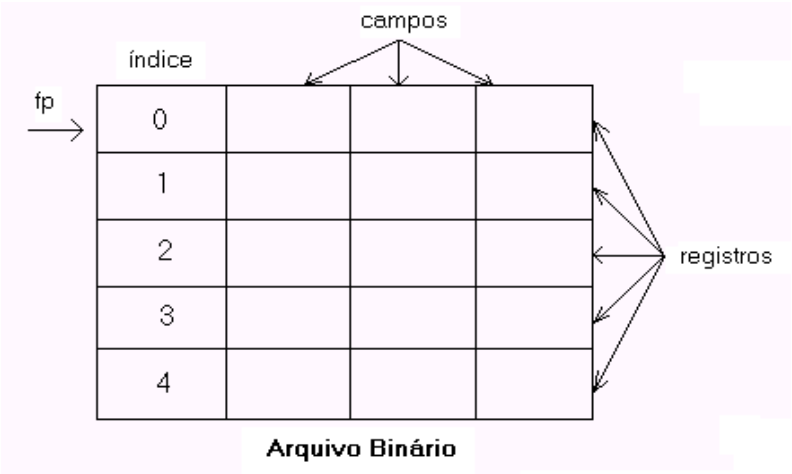
ABC\n
ABCDEF\n
\n
ABCDEFGH\n
EOF

Observações: EOF – **E**nd **O**f **F**ile (fim do arquivo).
Todo arquivo texto possui um nome.

15.1.2 - Filas binárias

Uma fila binária possui correspondência unívoca (1 para 1) com os bytes do dispositivo externo. Portanto nenhuma tradução de caracteres ocorrerá.

Um arquivo binário pode conter tipos de dados diferentes, como por



exemplo: **char**, **int**, **float**, vetores, ponteiros, strings, etc....

Figura 7: Representação de um arquivo

15.2 - Sistema de arquivo bufferizado

A ligação comum que mantém unido o sistema de entrada e saída bufferizado é um ponteiro que aponta para o arquivo. O ponteiro do arquivo identifica um determinado arquivo em disco e é usado pela fila associada a ele para direcionar cada uma das funções de entrada e saída bufferizada para o lugar em que elas operam. Um ponteiro de arquivo é uma variável de ponteiro do tipo **FILE** *. A palavra **FILE** é pré-definida em **"stdio.h"**.

Exemplo:

```
#include <stdio.h>

int main(void)
{
 FILE *fp;
```

15.2.1 - fopen

A função ***fopen*** (**f**ile **open**) permite abrir um arquivo ligando-o a uma fila de bytes.

Sintaxe: FILE ***fopen** (char *nome_arquivo, char *modo);
Prototype: stdio.h

nome_arquivo: Deve ser uma string que contenha: "**drive:\path\nome_do_arquivo**".
modo: É uma string que contém as características desejadas (veja tabela abaixo).

Observação: **t** (arquivo texto) e **b** (arquivo binário)

Tabela 19: Modo de abertura de arquivos

Modo	Significado
“r”	***Abre*** um arquivo-texto para leitura
“w”	***Cria*** um arquivo-texto para gravação
“a”	***Anexa*** a um arquivo-texto
“rb”	Abre um arquivo binário para leitura
“”wb”	Cria um arquivo binário para gravação
“ab”	Anexa a um arquivo binário
“r+”	Abre um arquivo-texto para leitura/gravação
“w+”	Cria um arquivo-texto para leitura/gravação
“a+”	Abre ou cria um arquivo-texto para leitura/gravação
“r+b”	Abre um arquivo binário para leitura/gravação
“w+b”	Cria um arquivo binário para leitura/gravação
“a+b”	Abre um arquivo binário para leitura/gravação
“rt”	Abre um arquivo texto para leitura
“wt”	Cria um arquivo texto para gravação
“at”	Anexa a um arquivo texto
“r+t”	Abre um arquivo-texto para leitura/gravação
“w+t”	Cria um arquivo-texto para leitura/gravação
“a+t”	Abre ou cria um arquivo-texto para leitura/gravação

Observação: Se ocorrer ERRO na abertura de um arquivo, **fopen** devolverá um ponteiro nulo, ou seja, se (fp == NULL) erro.

Exemplos:

```
#include <stdio.h>

int main (void)
{
 FILE *fp;

 if ((fp = fopen("teste.dat","r")) == NULL)
       printf("Erro Fatal: Impossível abrir o arquivo\n");
 else
       {
        printf("Ok, arquivo aberto\n");
        ...
```

ou

```
#include <stdio.h>

int main (void)
{
 FILE *fp;

 fp = fopen("teste.dat","w");
 if (fp == NULL)
     printf("Erro Fatal: Impossível criar o arquivo\n”);
 else
       {
        printf("Ok, arquivo criado com sucesso\n");
         ...
```

15.2.2 – putc

A função **putc** é utilizada para gravar caracteres em um arquivo.

Sintaxe: int **putc** (int ch, FILE *fp);
Prototype: stdio.h

ch é o caracter a ser gravado
fp é o ponteiro do arquivo aberto pela função fopen()

Observação: Se uma gravação for bem sucedida putc() devolverá o caracter gravado, caso contrário devolverá um EOF.

EOF - End of File (Fim de Arquivo)

15.2.3 – getc

A função **getc** é utilizada para ler caracteres de um arquivo.

Sintaxe: int **getc** (FILE *fp);
Prototype: stdio.h

fp é o ponteiro do arquivo aberto por fopen()

15.2.4 – feof

A função **feof** (**f**ile **e**nd **o**f **f**ile) determina se o fim de arquivo foi encontrado. Devolve 0 se não chegou ao fim do arquivo.

Sintaxe: int **feof** (FILE *fp);
Prototype: stdio.h

Programa exemplo (48): O programa lista na tela todas as linhas de um arquivo texto, numerando-as.

Observação: Este programa deve ser compilado e executado em um terminal de texto do sistema operacional da seguinte forma:

$ lista lista.c <enter>

Funcionamento: Todas as linhas do programa fonte "lista.c" são exibidos na tela com as linhas numeradas. Ao preencher toda a tela ocorre uma pausa. A cada parada é informado o nome do arquivo texto que está sendo listado e o número de bytes listados.

```
#include <stdio.h>

int main (int argc, char *argv[])
{
 FILE *fp;
 unsigned int i, v;
 char ch;

 if (argc != 2)
     printf("Sintaxe: LISTA <nome do arquivo>");
 else
     if ((fp = fopen(argv[1],"r")) == NULL)
         printf("ERRO: Arquivo [%s] não EXISTE\n");
     else
         {
          i = v = 1;
          printf("%d: ", i);
          do {
              ch = getc(fp);
              v++;
              if (ch == '\n')
```

```
                {
                 i++;
                 printf("\n%d: ", i);
                }
            else
                 printf("%c", ch);
           } while (!feof(fp));
        printf("[%s]  Byte(s): %d\n", argv[1],v);
        fclose(fp);
      }
}
```

15.2.5 – fclose

A função **fclose** (**f**ile **close**) é utilizada para fechar um arquivo aberto. Antes de fechar o arquivo os dados (que ainda estavam no **buffer**) são gravados.

Sintaxe: int **fclose** (FILE *fp);
Prototype: stdio.h

Observação: Retorna 0 se a operação foi bem sucedida.

15.2.6 - rewind

A função **rewind** estabelece o localizador de posição do arquivo para o início do arquivo especificado.

Sintaxe: void **rewind** (FILE *fp);
Prototype: stdio.h

15.2.7 – getw e putw

As funções **getw** e **putw** são utilizadas para ler e gravar, respectivamente, inteiros em um arquivo.

Função: Leitura de inteiros (**getw**)
Sintaxe: int **getw** (FILE *fp);
Prototype: stdio.h

Função: Gravação de inteiros (**putw**)
Sintaxe: int **putw** (int x, FILE *fp);
Prototype: stdio.h

15.2.8 – fgets e fputs

As funções **fgets** e **fputs** são utilizadas para ler e gravar strings.

Função: Leitura de strings (**fgets**)
Sintaxe: char ***fgets** (char *str, int comprimento, FILE *fp);
Prototype: stdio.h

Função: Gravação de strings (**fputs**)
Sintaxe: char ***fputs** (char *str, FILE *fp);
Prototype: stdio.h

Observação: A função **fgets** lê uma string do arquivo especificado até que leia ou um '**\n**' ou (comprimento - 1) caracteres.

15.2.9 – fread e fwrite

As funções **fread** e **fwrite** são utilizadas para ler e gravar blocos de dados, normalmente uma **struct**.

Função: Leitura de blocos (**fread**)
Sintaxe: int **fread** (void *buffer, int num_bytes, int cont, FILE *fp);
Prototype: stdio.h

Função: Gravação de blocos (**fwrite**)
Sintaxe: int **fwrite** (void *buffer, int num_bytes, int cont, FILE *fp);
Prototype: stdio.h

buffer: É um ponteiro para a região da memória ou o endereço de uma variável que receberá os dados lidos do arquivo pela função **fread** ou que será gravada no arquivo pela função **fwrite**.
num_bytes: Especifica a quantidade de bytes a serem lidos ou gravados.
cont: Determina quantos blocos (cada um com comprimento de num_bytes) serão lidos ou gravados.
fp: É o ponteiro para o arquivo.

15.2.10 – fseek

A função **fseek** é utilizada para ajustar o localizador de posição do arquivo, ou seja, permite selecionar a posição para efetuar operações de leitura e gravação aleatórias.

Sintaxe: int **fseek** (FILE *fp, long int num_bytes, int origem);
Prototype: stdio.h

num_bytes: É o número de bytes desde origem até chegar a posição desejada.

Tabela 20: Origem em arquivos

Origem	**Identificador**
Início do arquivo	SEEK_SET
Posição corrente	SEEK_CUR
Fim do arquivo	SEEK_END

15.2.11 – fprintf e fscanf

As funções **fprintf** e **fscanf** se comportam exatamente como **printf** e **scanf**, exceto pelo fato de que elas operam com arquivos em disco.

Função: Gravação de dados formatados (**fprintf**)
Sintaxe: int **fprintf** (FILE *fp, char *formato, lista argumentos);
Prototype: stdio.h

Função: Leitura de dados formatados (**fscanf**)
Sintaxe: int **fscanf** (FILE *fp, char *formato, lista argumentos);
Prototype: stdio.h

15.2.12 – remove

A função **remove** apaga do disco o arquivo especificado.

Sintaxe: int **remove** (char *nome_arquivo);

Prototype: stdio.h

Exemplos:

Abaixo, são listados três programas: **cria.c**, **lista.c**, **consulta.c**, os quais possuem como registro, uma palavra com no máximo 40 caracteres.

Programa exemplo (49): O programa permite criar um arquivo de palavras permitindo a gravação destas palavras.

```
// cria.c

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

int main (void)
{
 FILE *fp;
 char reg[40];
 char nome_do_arquivo[14];
 unsigned int n;
 char ch;

 printf("Nome do arquivo: ");
 scanf("%13s", nome_do_arquivo);
 if ((fp = fopen(nome_do_arquivo,"r+b")) != NULL)
     {
      printf("ERRO: Arquivo já existe\n");
      fclose(fp);
     }
else
    if ((fp = fopen(nome_do_arquivo,"w+b")) == NULL)
        printf("Erro Fatal: Problema no disco\n");
    else
         {
           n = 0;
           do {
                 printf("%d: Palavra: ", n);
                 scanf("%s", reg);        // scanf não aceita espaços
                 fwrite(reg, sizeof(reg), 1, fp);     // grava o registro
                 n++;
                 printf("Continua [S]im ou [N]ão ?");
                 do {
                     ch = getchar();
                    } while (!strchr("SsNn", ch));
                } while (strchr("Ss", ch));
            fclose(fp);
           }
}
```

Programa exemplo (50): O programa permite abrir o arquivo de palavras e exibe-os na tela.

```
// lista.c

#include <stdio.h>

int main (void)
{
 FILE *fp;
```

```
 char reg[40];
 char nome_do_arquivo[14];
 unsigned int n;

 printf("Nome do arquivo: ");
 scanf("%13s", nome_do_arquivo);
 if ((fp = fopen(nome_do_arquivo,"rb")) == NULL)
     printf("ERRO: Arquivo não EXISTE\n");
 else
     {
      n = 0;
      fread(reg, sizeof(reg), 1, fp);
      while (!feof(fp))
           {
            printf("%d: Palavra: %s\n", n, reg);
            n++;
            fread(reg, sizeof(reg), 1, fp);
           }
     fclose(fp);
    }
}
```

Programa exemplo (51): O programa permite consultar o arquivo de palavras. Para tanto é solicitado, ao usuário, o número do registro para ser calculado a posição deste registro no arquivo. Logo após o registro é exibido na tela.

```
// consulta.c

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

int main (void)
{
 FILE *fp;
 char reg[40];
 char nome_do_arquivo[14];
 unsigned int n,ok;
 long int posicao;
 char ch;

 printf("Nome do arquivo: ");
 scanf("%13s", nome_do_arquivo);
 if ((fp = fopen(nome_do_arquivo,"r+b")) == NULL)
     printf("ERRO: Arquivo não EXISTE\n");
 else
     {
      do {
            printf("Número do registro: ");
            scanf("%d", &n);
            posicao = n * sizeof(reg);
            fseek(fp, posicao, SEEK_SET);
            ok = fread(reg, sizeof(reg), 1, fp);
            if (ok)
                printf("%d: Palavra: %s\n", n, reg);
            else
                 printf("ERRO: Registro NÃO existe\n");
             printf("Continua [S/N] ? ");
             do {
                   ch = getchar();
```

```
                    } while (!strchr("SsNn",ch));
              } while (strchr("Ss",ch));
      fclose(fp);
     }
}
```

15.3 - Sistema de arquivo não bufferizado

Ao contrário do sistema de entrada e saída de alto nível (sistema bufferizado), o sistema de baixo nível (sistema não-bufferizado) não utiliza ponteiros de arquivo do tipo **FILE**, mas sim descritores de arquivos chamados **handles** do tipo **int**, ou seja, uma variável inteira.

15.3.1 – open, creat e close

A função **open** permite abrir/criar um arquivo em disco.

Função: Abre um arquivo (**open**)
Sintaxe: int **open** (char *nomearquivo, int acesso);
Prototype: stdio.h e fcntl.h

Tabela 21: Tipos de acessos em arquivos

Acesso	**Efeito**
O_RDONLY	Apenas leitura
O_WRONLY	Apenas gravação
O_RDWR	Leitura e gravação
O_CREAT	Cria e abre
O_TRUNC	Abre com "truncation"
O_EXCL	Abre exclusiva
O_APPEND	Abre para incluir no fim
O_TEXT	Translação CR para LF
O_BINARY	Sem translação

Observação: Se a função **open** obtiver sucesso (na abertura do arquivo), será devolvido um inteiro positivo. Um valor de retorno -1 (EOF) significa que **open** não pode abrir o arquivo. (O acesso se relaciona ao Ambiente UNIX)

A função **creat** cria um arquivo para operações de leitura e gravação.

Função: Cria um arquivo (**creat**)
Sintaxe: int **creat** (const char *nomearquivo, int modo);
Prototype: stdio.h e fcntl.h

Tabela 22: Modos de acessos em arquivos

Modo	**Efeito**
O_CREAT	Criar um arquivo se ele não existe
O_TRUNC	Truncar o arquivo se ele existe
O_APPEND	Anexar no fim antes de cada gravação
O_EXCL	Excluir. Força a criação se o arquivo existe
O_RDONLY	Acesso somente para leitura
O_WRONLY	Acesso somente para escrita
O_RDWR	Acesso para leitura e escrita

A função **close** fecha um arquivo. Devolve -1 (EOF), se não for capaz de fechar o arquivo.
Função: Fecha um arquivo (**close**)
Sintaxe: int **close** (int fd);

Prototype: stdio.h

15.3.2 – write e read

A função **write** grava tantos caracteres quantos forem o tamanho do buffer apontado pelo ponteiro **buf** para o arquivo em disco especificado por **fd**.

Função: Grava caracteres no arquivo (**write**)
Sintaxe: int **write** (int fd, void *buf, int tamanho);
Prototype: stdio.h

A função **read** copia os dados lidos no **buffer** apontado pelo ponteiro **buf**.

Função: Lê caracteres do arquivo (**read**)
Sintaxe: int **read** (int fd, void *buf, int tamanho);
Prototype: stdio.h

15.3.3 – unlink

A função **unlink** elimina um arquivo do disco (retorno: 0 sucesso ou 1 erro).

Sintaxe: int **unlink** (const char *nomearquivo);
Prototype: stdio.h

15.3.4 – lseek

A função **lseek** devolve o número de bytes (num_bytes) se obtiver sucesso.

Sintaxe: long int **lseek** (int fd, long int num_bytes, int origem);
Prototype: stdio.h

fd: descritor do arquivo
num_bytes: número de bytes desde a origem até a nova posição
origem: SEEK_SET (início), SEEK_CUR (corrente), SEEK_END (fim)

15.3.5 – eof

A função **eof** (**e**nd **o**f **f**ile) devolve: (**1**) **fim de arquivo**, (**0**) **não é fim de arquivo** ou (**-1**) **erro**.

Sintaxe: int int **eof** (int fd);
Prototype: stdio.h

15.3.6 – tell

A função **tell** devolve a posição corrente do descritor.

Sintaxe: long int **tell** (int fd);
Prototype: stdio.h

Programa exemplo (52): O programa permite criar um arquivo de palavras permitindo a gravação destas palavras.

```
// criar.c
```

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

int main (void)
{
 int fd;
 char reg[40];
 char nome_do_arquivo[14];
 unsigned int n;
 char ch;

 printf("Nome do arquivo: ");
 scanf("%13s", nome_do_arquivo);
 if ((fd = open(nome_do_arquivo,O_RDONLY,"r")) == EOF)
    if ((fd = creat(nome_do_arquivo,O_RDWR)) == EOF)
         printf("Erro Fatal: Problema no disco\n");
    else
          {
           n = 0;
           do {
                  printf("%d: Palavra: ", n);
                  scanf("%s", reg);
                  write(fd, reg, sizeof(reg));
                  n++;
                  printf("Continua [S]im ou [N]ão ?");
                  do {
                        ch = getchar();
                       } while (!strchr("SsNn",ch));
                 } while (strchr("Ss",ch));
          close(fd);
         }
 else
     {
      printf("ERRO: Arquivo já EXISTE\n");
      close(fd);
     }
}
```

Programa exemplo (53): O programa permite abrir o arquivo de palavras e exibe-os na tela.

```
// listar.c

#include <stdio.h>
#include <fcntl.h>

int main (void)
{
 int fd;
 char reg[40];
 char nome_do_arquivo[14];
 unsigned int n;
 int error;

 printf("Nome do Arquivo: ");
 scanf("%13s", nome_do_arquivo);
 if ((fd = open(nome_do_arquivo,O_RDONLY,"r")) == EOF)
     printf("ERRO: Arquivo NÃO existe ...\n");
```

```
 else
       {
          n = 0;
          do {
              error = read(fd, reg, sizeof(reg));
              printf("%d: Palavra: %s\n", n, reg);
              n++;
             } while (error);       // error = 0 – fim do arquivo
          close(fd);
          printf("%d palavra(s)\n",n);
         }
}
```

Programa exemplo (54): O programa permite alterar o arquivo de palavras. Para tanto é solicitado, ao usuário, o número do registro para ser calculado a posição deste registro no arquivo. Logo após o registro é exibido na tela e é solicitada a nova palavra.

```
// alterar.c

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

int main (void)
{
 int fd;
 char reg[40];
 char nome_do_arquivo[14];
 unsigned int n;
 long int posicao;
 char ch;

 printf("Nome do arquivo: ");
 scanf("%s", nome_do_arquivo);
 if ((fd = open(nome_do_arquivo,O_RDWR,"w")) == EOF)
     printf("ERRO: Arquivo NÃO existe ...\n");
 else
        {
         do {
               printf("Registro: ");
               scanf("%d", &n);
               posicao = n * sizeof(reg);
               lseek(fd, posicao, SEEK_SET);
               if (read(fd, reg, sizeof(reg)))
                  {
                    printf("%d: Palavra: %s\n", n, reg);
                    printf("NOVA PALAVRA: ");
                    scanf("%s", reg);
                    lseek(fd, posicao, SEEK_SET);
                    write(fd, reg, sizeof(reg));
                  }
              else
                    printf("ERRO: Registro não existe\n");
              printf("Continua [S]im ou [N]ão ? ");
              do {
                   ch = getchar();
                  } while (!strchr("SsNn",ch));
             } while (strchr("Ss",ch));
          close(fd);
```

```
        }
}
```

15.4 - Lista de exercícios (arquivos)

- ✓ Escreva um programa em C que recebe via teclado o **nome de um arquivo texto**. O programa deve imprimir na tela o **número de bytes** (caracteres) e o **número de linhas** do arquivo ou **ERRO: Arquivo não existe**.

Exemplo:

Nome do arquivo texto: **LISTA.C** <**enter**>
(**12345**) Bytes
(**44**) Linhas

ou

ERRO: Arquivo não existe

- ✓ Escreva um programa em C que recebe via teclado o **nome de um arquivo texto**. O programa deve permitir ao usuário consultar o arquivo através da entrada via teclado do **número da linha**. O programa deve imprimir a linha especificada ou **ERRO: Linha não existe**.

Exemplo:

Nome do arquivo texto: **LISTA.C** <**enter**>
Número de linha: **7** <**enter**>
7: int i, j, k;
Número de linha: **70** <**enter**>
ERRO: Linha não existe
Número de linha: **0** <**enter**>

ou

ERRO: Arquivo não existe

- ✓ Escreva um programa em C que recebe via teclado o **nome de dois arquivos texto** (origem e destino). O programa deve copiar o conteúdo do arquivo origem para o arquivo destino.

Exemplo:

Arquivo origem: **LISTA.C** <**enter**>
Arquivo destino: **LISTA.tmp** <**enter**>
(20345) Bytes copiados

- ✓ Escreva um programa em C que recebe via teclado o **nome do arquivo texto fonte** e o **nome do arquivo texto destino**. O programa deve converter o arquivo fonte para **maiúsculo** ou **minúsculo** (conforme escolha do usuário) gerando o arquivo texto destino.

Exemplo:

Arquivo fonte: **LISTA.C** <**enter**>
Arquivo destino: **LISTA.TMP** <**enter**>
[+] Maiúsculo ou [-] Minúsculo: **+**
(1234) Bytes convertidos

- ✓ Escreva um programa em C que recebe via teclado o **nome de um arquivo texto** e uma **palavra**. O programa deve imprimir todas as linhas que possuem esta palavra.

Exemplo:

Nome do arquivo texto: **PALAVRA.C** <**enter**>
Palavra: **if** <**enter**>
23: if (fp == NULL)
33: if (ch == '\n')
37: if (compara(linha,palavra))
41: if (ch != '\n')
59: if (linha[i] == palavra[0])
65: if (linha[k] != palavra[j])
69: if (achei)

- ✓ Escreva um programa em C que recebe via teclado o **nome de um arquivo texto**. O programa deve permitir ao usuário consultar o arquivo através da entrada via teclado do **número inicial** e **número final**. O programa deve imprimir desde a linha inicial até a linha final ou **ERRO: Linhas não existem**.

Exemplo:

Nome do arquivo texto: **LISTA.C** <**enter**>
Número inicial: **7** <**enter**>
Número final: **9** <**enter**>
7: int i, j, k;
8: char tecla;
9: long int bytes = 0;

ou

Número inicial: **70** <**enter**>
Número final: **90** <**enter**>
ERRO: Linhas não existem

- ✓ Escreva um programa em C (**grava.c**) que recebe via teclado o **nome de um arquivo binário**. O programa deve permitir ao usuário inserir nomes (máximo 30 caracteres) neste arquivo via teclado. O programa termina quando o usuário digitar <**enter**> na entrada do nome.

Exemplo:

Nome do arquivo binário: **NOMES.DAT** <**enter**>
Nome: **Beatriz** <**enter**>
Nome: **Eva** <**enter**>
Nome: **Debora** <**enter**>
Nome: **Carla** <**enter**>
Nome: **Fatima** <**enter**>
Nome: **Ana** <**enter**>
Nome: <**enter**>

- ✓ Escreva um programa em C (**ler.c**) que recebe via teclado o **nome de um arquivo binário**. O programa deve ler e imprimir na tela todos os nomes armazenados no arquivo pelo programa "**grava.c**".

Exemplo:

Nome do arquivo binário: **NOMES.DAT** <**enter**>
Nome: **Beatriz**
Nome: **Eva**
Nome: **Debora**
Nome: **Carla**
Nome: **Fatima**
Nome: **Ana**

- ✓ Escreva um programa em C (**sort.c**) que recebe via teclado o **nome de um arquivo binário**. O programa deve ler, ordenar e gravar novamente os nomes no mesmo arquivo.

Exemplo:

Nome do arquivo binário: **NOMES.DAT** <**enter**>
Ok, arquivo ordenado

Observação: Utilize o programa anterior (**ler.c**) para ver os nomes ordenados.

- ✓ Escreva um programa em C (**salva.c**) que recebe via teclado o **nome de um arquivo binário**. O programa deve permitir ao usuário inserir **nome** (máximo 30 caracteres), **idade** e **sexo** ([M]asculino ou [F]eminino) neste arquivo via teclado. O programa termina quando o usuário digitar <**enter**> na entrada do nome.

Exemplo:

Nome do arquivo binário: **DADOS.DAT** <**enter**>
Nome: **Paulo Roberto** <**enter**>
Idade: **41** <**enter**>
Sexo [M/F]: **m** <**enter**>
Nome: **Renato Luis** <**enter**>
Idade: **38** <**enter**>
Sexo [M/F]: **m** <**enter**>
Nome: **Ana Maria** <**enter**>
Idade: **44** <**enter**>
Sexo [M/F]: **f** <**enter**>
Nome: <**enter**>

- ✓ Escreva um programa em C (**list.c**) que recebe via teclado o **nome de um arquivo binário**. O programa deve ler e imprimir na tela todos os dados (nome, idade e sexo) armazenados no arquivo pelo programa "**salva.c**" (veja exemplo abaixo).

Exemplo:

Nome do arquivo binário: **DADOS.DAT** <**enter**>
Nome: **Paulo Roberto**
Idade: **41**
Sexo: **MASCULINO**
Nome: **Renato Luis**
Idade: **38**
Sexo: **MASCULINO**
Nome: **Ana Maria**

Idade: **44**
Sexo: **FEMININO**

- ✓ Escreva um programa em C (**conta.c**) que recebe via teclado o **nome de um arquivo binário**. O programa deve verificar a quantidade de homens e a quantidade de mulheres no aquivo criado pelo programa "**salva.c**".

Exemplo:

Nome do arquivo binário: **DADOS.DAT** <**enter**>
(2) Homens
(1) Mulheres

16. Tipos de dados definidos pelo programador

16.1 - Estruturas

Em C, uma estrutura (*struct*) é uma coleção de campos que fazem referência a uma variável, ou seja, uma variável possui um conjunto de valores (que podem ser de tipos iguais ou diferentes – **agregados heterogêneos**). Uma estrutura propicia uma maneira conveniente de manter-se juntas informações relacionadas.

```
struct nome_da_estrutura {
                          tipo nome_variável;
                          tipo nome_variável;
                          . . .
                         } lista_de_variáveis;
```

Na definição de uma estrutura pode-se omitir o **nome_da_estrutura** ou a **lista_de_variáveis** deste tipo, mas não ambos.

```
struct reg_cliente {                  // nome_da_estrutura
                    char nome[30];
                    float salario;
                    int idade;
                   } cliente, fornecedor;        // lista_de_variáveis
```

16.1.1 - Referência aos elementos da estrutura

Para fazer referência aos elementos individuais de uma estrutura deve-se utilizar o operador "." (ponto), que as vezes é chamado de operador de seleção.

```
nome_variável_estrutura.nome_campo
```

Exemplo: cliente.idade = 18;

16.1.2 - Matrizes de estruturas

É possível criar matrizes de estruturas.

Exemplo: struct **reg_cliente cliente**[100];

Cria-se um **array** (vetor) para guardar 100 ocorrências de cliente.

<u>Acesso ao primeiro elemento da estrutura</u>:

```
strcpy(cliente[0].nome,"Paulo");
cliente[0].salario = 500.00;
cliente[0].idade = 41;
```

<u>Acesso ao último elemento da estrutura</u>:

```
strcpy(cliente[99].nome,"Renato");
cliente[99].salario = 1500.00;
cliente[99].idade = 37;
```

16.1.3 - Passando estruturas para funções

16.1.3.1 - Passando elementos da estrutura

Desta forma, é passado apenas uma cópia do valor para a função.

Chamada da função: função (cliente.idade);

16.1.3.2 - Passando estruturas inteiras

É possível passar uma estrutura (**struct**) inteira para uma função.

Observação: O tipo do argumento deve coincidir com o tipo do parâmetro.

Exemplo:

```
#include <stdio.h>

struct TIPO {
             int a,b;
             char ch;
            };

void function (struct TIPO parm);

int main (void)
{
 struct TIPO arg;

 arg.a = 1000;
 arg.b = 500;
 arg.ch = 'A';
 function (arg);        // arg é o argumento
}

void function (struct TIPO parm)    // parm é o parâmetro tipo estrutura
{
 printf("%d", parm.a);
 printf("%d", parm.b);
 printf("%c", parm.ch);
}
```

16.1.4 - Ponteiros para estruturas

C permite que ponteiros apontem para estruturas da mesma maneira como permite ponteiros para qualquer outro tipo de variável.

Exemplo:

```
struct reg_cliente {
                    char nome[30];
                    float salario;
                    int idade;
                   } cliente;
struct reg_cliente *p;
```

Neste caso **p** é um ponteiro para a estrutura **reg_cliente**.

```
p = &cliente;     // o ponteiro p aponta para a variável cliente
```

Para acessar um elemento de cliente através do ponteiro **p** deve-se escrever das seguintes formas:

```
(*p).idade = 21;        ou          p->idade = 21;
```

16.2 - Campos de bit

C possui um método inerente de acesso a um único **bit** em um **byte**. Este é baseado na estrutura. Um campo de bit é apenas um tipo especial de estrutura que define o comprimento em bits que cada elemento terá.

```
struct nome_tipo_estrutura {
                            tipo nome_1: comprimento;
                            tipo nome_2: comprimento;
                            ...
                            tipo nome_3: comprimento;
                           };
```

Observação: Um campo de **bit** deve ser do tipo **int**, **unsigned int** ou **signed int**.

Exemplo:

```
struct dispositivo {
                    unsigned ativo: 1;
                    unsigned pronto: 1;
                    unsigned erro: 1;
                   } codigo_disp;
```

Esta estrutura define três variáveis de um bit cada, os outros 5 bits ficam desocupados, pois **codigo_disp** ocupa 8 bits, ou seja, 1 byte.

16.3 - Union

Em C, uma **union** (união) é uma localização de memória que é utilizada por várias variáveis diferentes, que podem ser de tipos diferentes (**int** ou **char**).

Exemplo:

```
union u_type {
              int i;
              char ch;
             };
```

Neste caso a variável **i** e **ch** ocupam a mesma localização da memória.

16.4 - typedef

C permite que o programador defina explicitamente novos nomes de tipos de dados utilizando a palavra reservada **typedef.** Realmente não se cria uma nova classe de dados, mas sim, um novo nome para um tipo existente.

typedef tipo_base novo_nome_de_tipo;

Exemplo: typedef float real; **typedef** unsigned char byte;

real salario; byte n_dentes;

16.5 - Tipos de dados avançados

C permite que seja aplicado vários **modificadores de tipo** aos tipos de dados básicos.

modificador_tipo tipo_básico lista_variáveis;

16.5.1 - Modificadores de acesso

16.5.1.1 - O modificador const

Durante a execução, o programa não poderá alterar uma variável declarada com o modificador **const**, exceto dar a variável um valor inicial.

Exemplo: const float versao = 3.30;

16.5.1.2 - O modificador volatile

Utiliza-se o modificador **volatile** para dizer ao C que o valor de uma variável pode ser alterado sem uma especificação explícita do programa.

Exemplo: volatile int clock;

16.5.2 - Especificadores de classe de armazenamento

16.5.2.1 - O especificador auto

O especificador **auto** é utilizado para declarar variáveis locais. (Por default as variáveis locais são **auto**).

16.5.2.2 - O especificador extern

O especificador **extern** diz ao compilador que os tipos de dados e nomes de variáveis que se seguem já foram declarados em um outro lugar (por exemplo, outro programa fonte).

16.5.2.3 - O especificador static

As variáveis **static** são variáveis permanentes dentro de suas próprias funções. São diferentes das variáveis globais, porque não são conhecidas fora de suas funções, mas mantém seus valores entre as chamadas.

16.5.2.4. O especificador register

Se aplica apenas as variáveis **int** ou **char**. O especificador **register** faz com que o compilador mantenha o valor das variáveis declaradas dentro dos registradores da CPU, e não na memória, onde as variáveis normais são armazenadas normalmente. É próprio para variáveis de loop's (laços), pois torna o laço mais rápido.

Observação: Só pode ser utilizado em variáveis locais.

16.5.3 - Operadores avançados

16.5.3.1 - Operadores bit a bit

Se referem ao teste, ajuste ou troca de bits reais por um **byte** ou palavra (podem ser de tipos **char** ou **int**).

Tabela 23: Operadores bit a bit

Operador	Ação
&	*And*
|	*Or*
^	*Or* exclusivo (*XOr*)
~	Complemento de um (*Not*)
>>	Deslocar para a direita (*shift*)
<<	Deslocar para a esquerda (*shift*)

Deslocamento (shift): variável >> número de deslocamentos;

Exemplo:

```
#include <stdio.h>

int main (void)
{
 int a = 128, b, c;            // a = 128   binário -> 10000000

 b = a >> 1;                   // b = 64   binário -> 0100000
 c = b << 1;                   // c = 128  binário -> 1000000
 printf("a = %d\n",a);
 printf("b = %d\n",b);
 printf("c = %d\n",c);
}
```

16.5.3.2 - O operador ?

Pode-se utilizar o operador **?** para substituir comandos **if ... else** que tenham a seguinte forma geral:

```
if (condição)
    comando_1;
else
    comando_2;
```

A principal restrição do operador **?** é que os comandos do **if ... else** devem ser comando simples não podendo serem comandos compostos.

Forma geral: condição ? comando_1: comando_2;

Exemplo:

```
#include <stdio.h>

int main (void)
{
 int a, b;

 printf("a = ");
 scanf("%d",&a);
 printf("b = ");
 scanf("%d",&b);
```

```
 a > b ? printf("Maior (a) -> %d\n",a) : printf("Maior (b) -> %d\n",b);
}
```

16.5.3.3 - Formas abreviadas de C

C tem abreviações especiais que simplificam a codificação de um certo tipo de comando de atribuição.

Exemplo:

x = x + 10; pode ser escrito como x += 10;

Esta forma abreviada funcionará com todos os operadores aritméticos de C.

Forma normal: variável = variável **operador** constante;
x = x + 7;
Forma abreviada: variável **operador**= constante;
x += 7;

16.5.3.4 - O operador ,

Pode-se utilizar a vírgula para juntar várias expressões. O compilador sempre avalia o lado esquerdo da vírgula como **void**. Assim, a expressão do lado direito ficará sendo o valor de toda expressão separada por vírgula.

Exemplo: x = (y = 3, y + 1);

Primeiramente é atribuído o valor 3 a **y** e depois atribuído o valor 4 a **x**. Os parênteses são necessários, porque a vírgula tem precedência mais baixa que o operador de atribuição.

17. Listas lineares

As listas lineares encadeadas (**filas** e **pilhas** alocadas dinamicamente) permitem alocação indeterminada de elementos em uma estrutura de dados. Os elementos são alocados na memória RAM dinamicamente.

Uma lista encadeada tem por característica um **elo** de ligação entre um elemento e outro. Ao contrário de um vetor, onde os elementos são alocados estaticamente e em posições contíguas na memória RAM, uma lista encadeada é alocada dinamicamente, tendo como característica básica, que os elementos são alocados em posições diferentes (aleatórias) na memória.

As listas lineares encadeadas possuem um ***header*** (cabeça) que armazena o primeiro elemento da lista.

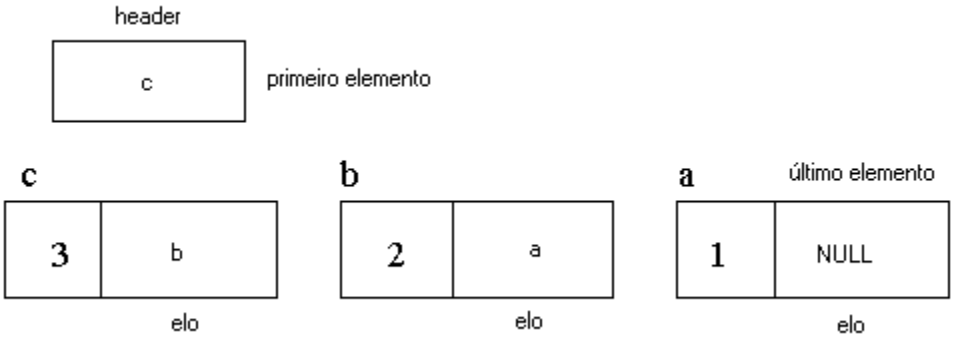
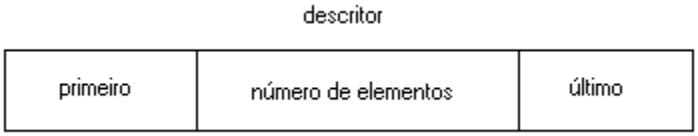


Figura 18: Representação de uma lista encadeada

Dois problemas existem em uma lista encadeada:

a) **Primeiro**: É que as listas são percorridas seqüencialmente, apenas numa direção, ou seja, do primeiro ao último elemento;
b) **Segundo**: É que a informação do número de elementos da lista é obtido somente por uma varredura completa na lista.

Para resolver estes dois problemas pode-se utilizar um descritor da seguinte maneira:



As vantagens de um descritor são:

- Conhecer o número de elementos da lista linear sem ter que percorrê-la;
- Acessar o último elemento facilitando a inserção ou remoção no final da lista.

As listas podem ser organizadas como: **pilhas** ou **filas.**

Pilha: Estrutura linear organizada de forma que a entrada e a saída dos dados é feita na mesma extremidade.
Forma de acesso: LIFO (**L**ast **I**nput **F**irst **O**utput), ou seja, o último elemento a entrar na pilha é o primeiro a sair dela.

Fila: Estrutura linear organizada em forma que a entrada dos dados é feita por uma extremidade da lista linear e, a saída, é feita na outra extremidade.

Forma de acesso: FIFO (**F**irst **I**nput **F**irst **O**utput), ou seja, o primeiro elemento a entrar na fila é o primeiro a sair da fila.

E - Entrada de Dados
S - Saída de Dados

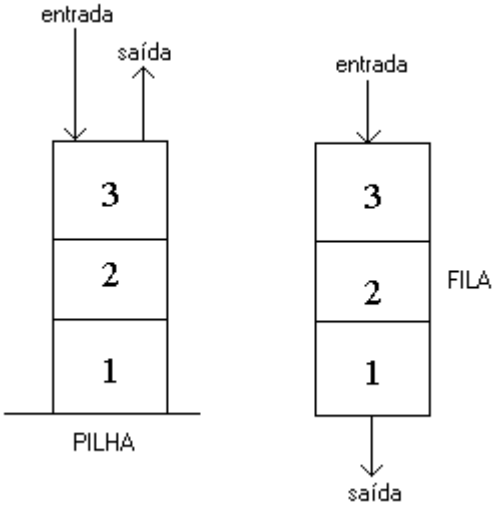


Figura 19: Representação de uma fila e uma pilha

Funcionamento desta pilha:

Entrada: 1, 2 e 3
Saída: 3, 2 e 1

Funcionamento desta fila:

Entrada: 1, 2 e 3
Saída: 1, 2 e 3

17.1 - Implementação de uma pilha

Programa exemplo (55): O programa permite inserir números inteiros em uma pilha. Quando o número digitado for igual à zero (0), todos os números da pilha são listados.

```
// pilha.c

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

#define SUCESSO                1
#define FALTA_DE_MEMORIA       2
#define PILHA_VAZIA            3

struct DADO {
             int info;                    // informação
             struct DADO *elo;            // elo para o próximo elemento
            };

struct PILHA {
```

```
              struct DADO *topo;
             };

void cria_pilha (struct PILHA *p);
int push (struct PILHA *p, int valor);
int pop (struct PILHA *p, int *info);
int consulta_pilha (struct PILHA p, int *info);
void imprime_erro (int erro);

// ------------------------------------ main

int main (void)
{
 struct PILHA p;
 int valor, erro;
 char ch;

 cria_pilha(&p);
 do {
      printf("[P]ush, P[O]p, [C]onsultar ou [F]im ?");
      do {
          ch = toupper(getchar());
         } while (!strchr("POCF", ch));
     if (ch == 'P')
       {
         printf("Valor: ");
         scanf("%d", &valor);
       }
     switch (ch)
         {
           case 'P': erro = push(&p, valor);
                     if (erro != SUCESSO)
                        imprime_erro(erro);
                     break;
          case 'O': erro = pop(&p,&valor);
                    if (erro != SUCESSO)
                       imprime_erro(erro);
                  else
                       printf("Informação Excluída: %d\n",valor);
                  break;
         case 'C': erro = consulta_pilha(p, &valor);
                  if (erro != SUCESSO)
                      imprime_erro(erro);
                  else
                      printf("Valor: %d\n", valor);
                  break;
      }
     } while (ch != 'F');
}

// ------------------------------------ cria_pilha

void cria_pilha (struct PILHA *p)
{
 p->topo = NULL;
}

// ------------------------------------ push

int push (struct PILHA *p, int valor)
```

```
{
 struct DADO *t;

 t = (struct DADO *) malloc(sizeof(struct DADO));
 if (t == NULL)
      return(FALTA_DE_MEMORIA);
 else
     {
      t->info = valor;
      t->elo = p->topo;
      if (p->topo == NULL)
         t->elo = NULL;
      p->topo = t;
      return(SUCESSO);
     }
}

// ------------------------------------ pop

int pop (struct PILHA *p, int *info)
{
 struct DADO *t;

 if (p->topo == NULL)
       return(PILHA_VAZIA);
 else
      {
       t = p->topo;
         *info = t->info;
       p->topo = t->elo;
       free(t);
       return(SUCESSO);
     }
}

// ------------------------------------ consulta_pilha

int consulta_pilha (struct PILHA p, int *info)
{
 struct DADO *t;

 if (p.topo == NULL)
     return(PILHA_VAZIA);
 else
     {
      t = p.topo;
      *info = t->info;
      return(SUCESSO);
     }
}

// ------------------------------------ imprime_erro

void imprime_erro (int erro)
{
 switch (erro)
 {
  case FALTA_DE_MEMORIA: printf("ERRO: Falta de memória, tecle algo\n");
                         break;
       case PILHA_VAZIA: printf("ERRO: Pilha vazia, tecle algo\n");
                         break;
```

```
 }
}
```

17.2 - Implementação de uma fila

Programa exemplo (56): O programa permite inserir números inteiros em uma fila. Quando o número digitado for igual à zero (0), todos os números da fila são listados.

```
// fila.c

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

#define SUCESSO               1
#define FALTA_DE_MEMORIA      2
#define FILA_VAZIA            3

struct DADO {
             int info;               // informação
             struct DADO *elo;       // elo para o próximo elemento
            };

struct FILA {
             struct DADO *primeiro;
             int tamanho;
             struct DADO *ultimo;
            };

void cria_fila (struct FILA *f);
int incluir_fila (struct FILA *f, int valor);
int excluir_fila (struct FILA *f);
int consultar_fila (struct FILA f, int *j);
void imprime_erro (int erro);

// ------------------------------------ main

int main (void)
{
 struct FILA f;
 int valor, erro;
 char ch;

 cria_fila(&f);
 do {
      printf("[I]ncluir, [E]xcluir, [C]onsultar ou [F]im ?");
      do {
          ch = toupper(getchar());
         } while (!strchr("IECF", ch));
     if (ch == 'I')
       {
         printf("Valor: ");
         scanf("%d", &valor);
       }
     switch (ch)
         {
           case 'I': erro = incluir_fila(&f, valor);
                        if (erro != SUCESSO)
                           imprime_erro(erro);
                        break;
```

```
          case 'E': erro = excluir_fila(&f);
                        if (erro != SUCESSO)
                           imprime_erro(erro);
                        break;
         case 'C': erro = consultar_fila(f, &valor);
                        if (erro != SUCESSO)
                            imprime_erro(erro);
                        else
                            printf("Valor: %d\n", valor);
                       break;
      }
     } while (ch != 'F');
}

// ------------------------------------ cria_fila

void cria_fila (struct FILA *f)
{
 f->primeiro = NULL;
 f->tamanho = 0;
 f->ultimo = NULL;
}

// ------------------------------------ incluir_fila

int incluir_fila (struct FILA *f, int valor)
{
 struct DADO *t;

 t = (struct DADO *) malloc(sizeof(struct DADO));
 if (t == NULL)
      return(FALTA_DE_MEMORIA);
 else
     {
      t->info = valor;
      t->elo = NULL;
      if (f->ultimo != NULL)
         f->ultimo->elo = t;
      f->ultimo = t;
      if (f->primeiro == NULL)
         f->primeiro = t;
      f->tamanho = f->tamanho + 1;
      return(SUCESSO);
     }
}

// ------------------------------------ excluir_fila

int excluir_fila (struct FILA *f)
{
 struct DADO *t;

 if (f->primeiro == NULL)
       return(FILA_VAZIA);
 else
      {
       t = f->primeiro;
         printf("Elemento Excluido: %d\n",t->info);
       f->primeiro = t->elo;
       f->tamanho = f->tamanho - 1;
```

```
        free(t);
        if (f->primeiro == NULL)
           f->ultimo = NULL;
        return(SUCESSO);
      }
}

// ------------------------------------ consultar_fila

int consultar_fila (struct FILA f, int *j)
{
 struct DADO *t;

 if (f.primeiro == NULL)
     return(FILA_VAZIA);
 else
     {
      t = f.primeiro;
      *j = t->info;
      return(SUCESSO);
     }
}

// ------------------------------------ imprime_erro

void imprime_erro (int erro)
{
 switch (erro)
 {
  case FALTA_DE_MEMORIA: printf("\nERRO: Falta de memória, tecle algo\n");
                         break;
       case FILA_VAZIA: printf("\nERRO: Fila vazia, tecle algo\n");
                         break;
 }
}
```

17.3 - Lista duplamente encadeada

Uma lista duplamente encadeada possui um elo para o elemento anterior e um elo para o elemento posterior. Possui uma vantagem sobre a lista encadeada, pois este tipo de lista pode ser percorrida em duas direções (para a direita e para a esquerda).

Programa exemplo (57): O programa permite inserir números inteiros em uma lista duplamente encadeada. A inserção pode ser pela **esquerda** ou a **direita**. A exibição dos elementos da lista pode ser feita pela esquerda ou direita.

```
// dupla.c

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

struct ELEMENTO {
                 struct ELEMENTO *anterior;
                 int dado;
                 struct ELEMENTO *posterior;
                };
```

```
struct DESCRITOR {
                  struct ELEMENTO *primeiro;
                  int n;
                  struct ELEMENTO *ultimo;
                 };

void inicializa_descritor (struct DESCRITOR *d);
void insere_direita (struct DESCRITOR *d, int n);
void insere_esquerda (struct DESCRITOR *d, int n);
void exibir_lista_direita (struct DESCRITOR d);
void exibir_lista_esquerda (struct DESCRITOR d);

// ------------------------------------ main

int main (void)
{
 struct DESCRITOR d;
 int n;
 char op;

 inicializa_descritor(&d);
 do {
      printf("Número [0-Sair]: ");
      scanf("%d", &n);
      if (n != 0)
        {
         printf("Inserir a [E]squerda ou [D]ireita ?");
         do {
             op = toupper(getchar());
            } while (!strchr("ED", op));
         switch (op)
         {
           case 'E': insere_esquerda(&d,n);
                     break;
           case 'D': insere_direita(&d,n);
                     break;
         }
        }
    } while (n != 0);
  do {
      printf("Listar: [E]squerda, [D]ireita ou [F]im?");
      do {
          op = toupper(getchar());
         } while (!strchr("EDF", op));
      switch (op)
       {
        case 'E': exibir_lista_esquerda(d);
                  break;
        case 'D': exibir_lista_direita(d);
                  break;
       }
    } while (op != 'F');
}

// ------------------------------------ inicializa_descritor

void inicializa_descritor (struct DESCRITOR *d)
{
 d->primeiro = NULL;
 d->n = 0;
```

```
 d->ultimo = NULL;
}

// ------------------------------------ insere_esquerda

void insere_esquerda (struct DESCRITOR *d, int n)
{
 struct ELEMENTO *p,*q;

 p = (struct ELEMENTO *) malloc(sizeof(struct ELEMENTO));
 if (p == NULL)
     printf("ERRO: Falta de Memória\n");
 else
     {
      if (d->n == 0)
         {
          p->anterior = NULL;
          p->dado = n;
          p->posterior = NULL;
          d->primeiro = p;
          d->n = 1;
          d->ultimo = p;
         }
      else
          {
           q = d->primeiro;
           p->anterior = NULL;
           p->dado = n;
           p->posterior = q;
           q->anterior = p;
           d->primeiro = p;
           d->n = d->n + 1;
          }
     }
}

// ------------------------------------ insere_direita

void insere_direita (struct DESCRITOR *d, int n)
{
 struct ELEMENTO *p,*q;

 p = (struct ELEMENTO *) malloc(sizeof(struct ELEMENTO));
 if (p == NULL)
     printf("ERRO: Falta de memória\n");
 else
     {
      if (d->n == 0)
         {
          p->anterior = NULL;
          p->dado = n;
          p->posterior = NULL;
          d->primeiro = p;
          d->n = 1;
          d->ultimo = p;
         }
      else
          {
           q = d->ultimo;
           p->anterior = q;
```

```
            p->dado = n;
            p->posterior = NULL;
            q->posterior = p;
            d->ultimo = p;
            d->n = d->n + 1;
           }
     }
}

// ------------------------------------ exibir_lista_direita

void exibir_lista_direita (struct DESCRITOR d)
{
 struct ELEMENTO *p;

 p = d.ultimo;
 while (p->anterior != NULL)
       {
        printf("Valor: %d\n", p->dado);
        p = p->anterior;
       }
 printf("Valor: %d\n", p->dado);
}

// ------------------------------------ exibir_lista_esquerda

void exibir_lista_esquerda (struct DESCRITOR d)
{
 struct ELEMENTO *p;

 p = d.primeiro;
 while (p->posterior != NULL)
       {
        printf("Valor: %d\n", p->dado);
        p = p->posterior;
       }
 printf("Valor: %d\n", p->dado);
}
```

18. Lista de exercícios gerais

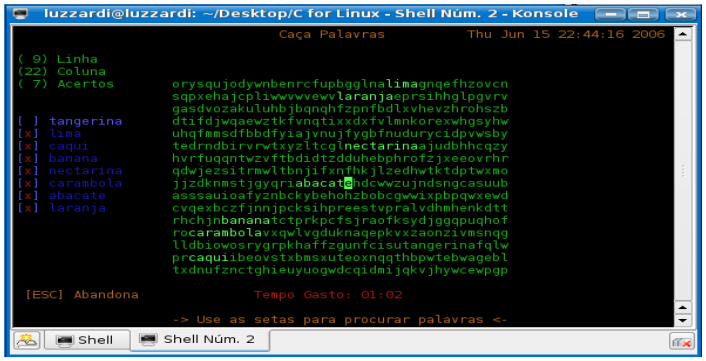
- ✓ Escreva um programa em C que exibe um relógio na tela. O formato do relógio deve ser **hh:mm:ss**.

- ✓ Escreva um programa em C que exibe a data na tela. O formato da data deve ser **dd:mm:aaaa**.

19. Programas exemplos usando ncurses

19.1 Jogo de Caça-Palavras

O jogo de **caça-palavras** abaixo demonstra a utilização de biblioteca "**ncurses**" específica para terminal em modo texto.




```
// palavras.c (Kubuntu - Anjuta)
// Autor: Paulo Roberto Gomes Luzzardi
// Data: 15/06/2006

// ------------------------------------------------------------ includes

#include <string.h>
#include <ctype.h>
#include <curses.h>            // biblioteca ncurses
#include <signal.h>
#include <stdlib.h>
#include <time.h>

// ------------------------------------------------------------ definicoes

#define ESC    27
#define ENTER  13
#define UP    259
#define DOWN  258
#define LEFT  260
#define RIGHT 261

#define COR_TEXTO    COLOR_GREEN
#define COR_PALAVRA  COLOR_GREEN
#define COR_PALAVRAS COLOR_BLUE
#define COR_TELA     COLOR_YELLOW
#define COR_XIS      COLOR_RED
#define COR_ACERTO   COLOR_BLUE
#define COR_FINAL    COLOR_RED
#define COR_RECORDE  COLOR_WHITE
#define COR_TEMPO    COLOR_RED
#define COR_NOME     COLOR_YELLOW
#define COR_RELOGIO  COLOR_YELLOW
#define COR_PARABENS COLOR_CYAN

// ------------------------------------------------------------ prototypes

void tela(void);
void sorteia_palavras(int n);
void exibe_palavras(void);
void exibe_palavras_sorteadas(void);
int procura_palavra(char *s);
void final(int min, int seg);
void troca(int *x, int *y);
void tempo(int c, int l);
void relogio(int *h, int *m, int *s);
int posicao_palavra(int pos);
void STRING(int c, int l, int n, char *s);
void apaga_linha(int c, int l, int n);
```

```
void novo_recorde(char *s, int min, int seg);

// ------------------------------------------------------- prototypes

void inicializa_janelas(void);
void finaliza_janelas(void);
void clrscr(void);
void gotoxy(int c, int l);
void flushall(void);
int RANDOM(int inic, float faixa);
void randomize(void);
void textcolor(int cor);
void textcolorlight(int cor);

// ------------------------------------------------------- inicializa_janelas

void inicializa_janelas(void)
{
 initscr();                    // inicializa a biblioteca curses
 keypad(stdscr, TRUE);         // mapeamento do teclado
 (void) nonl();                // não converte NL em CR/NL
 (void) cbreak();              // não espera por \n
 (void) echo();                // saída em cores
 if (has_colors())
    {
     start_color();
     init_pair(1,COLOR_RED,COLOR_BLACK);
     init_pair(2,COLOR_GREEN,COLOR_BLACK);
     init_pair(3,COLOR_YELLOW,COLOR_BLACK);
     init_pair(4,COLOR_BLUE,COLOR_BLACK);
     init_pair(5,COLOR_CYAN,COLOR_BLACK);
     init_pair(6,COLOR_MAGENTA,COLOR_BLACK);
     init_pair(7,COLOR_WHITE,COLOR_BLACK);
     attrset(COLOR_PAIR(A_BOLD));
    }
}

// ------------------------------------------------------- finaliza_janelas

void finaliza_janelas(void)
{
 endwin();
 system("clear");
}

// ------------------------------------------------------- clrscr

void clrscr(void)
{
 clear();
 refresh();
}

// ------------------------------------------------------- gotoxy

void gotoxy(int c, int l)
{
 move(l-1,c-1);
 refresh();
}

// ------------------------------------------------------- flushall

void flushall(void)
{
 getchar();
}

// ------------------------------------------------------- RANDOM

int RANDOM(int inic, float faixa)
{
 int j;

 j = inic + (int) ((faixa+1.0) * rand() / (RAND_MAX + 1.0));
```

```
 return(j);
}

// ------------------------------------------------------- randomize

void randomize(void)
{
 srand((unsigned int)time((time_t *)NULL));
}

// ------------------------------------------------------- textcolor

void textcolor(int cor)
{
 attrset(COLOR_PAIR(cor));
}

// ------------------------------------------------------- textcolorlight

void textcolorlight(int cor)
{
 attrset(COLOR_PAIR(cor)+A_BOLD);
}

// ------------------------------------------------------- variaveis

FILE *fp;
char *nome_arquivo = "recorde.pal";
struct {
        char nome[31];
        int minutos;
        int segundos;
       } dados,recorde;

unsigned short letras[21][61];

#define NPAL  24
#define PSORT  8

char frutas[NPAL][10] = {"abacaxi","banana","ameixa","pera","goiaba","amora","bergamota",
        "morango","lima","abacate","carambola","laranja","kiwi","tangerina","figo","pitanga"
,
        "pessego","acerola","cereja","nectarina","pomelo","caqui","caju","marmelo"};

int temp[NPAL],acertou[NPAL];
char sit[NPAL],palavra[10];
int click = TRUE,linha[21];

// ------------------------------------------------------- main

int main(void)
{
 int tecla,acertos = 0,troquei;
 int l = 12,c = 40;
 int l1,c1,l2,c2,t,x,y,i,j,k,w;
 int hi = 0,mi = 0,si = 0;
 int ma,sa,incrementa = FALSE;
 int ht,mt,st;
 char s[10];

 inicializa_janelas();
 clrscr();
 randomize();
 textcolor(COR_TEXTO);
 tela();
 sorteia_palavras(PSORT);
 exibe_palavras();
 exibe_palavras_sorteadas();
 relogio(&hi,&mi,&si);
 ma = 0;
 do {
     relogio(&ht,&mt,&st);
     sa = st - si;
     if (sa < 0)
         sa = sa + 60;
```

```
if (sa == 58)
    incrementa = TRUE;
if (sa == 0 && incrementa)
   {
    ma++;
    incrementa = FALSE;
   }
textcolor(COR_TEMPO);
gotoxy(30,22);
printw("Tempo Gasto: %02d:%02d\n",ma,sa);
textcolor(COR_TEXTO);
gotoxy(1,3);
printw("(%2d) Linha",l-4);
   gotoxy(1,4);
   printw("(%2d) Coluna",c-19);
   gotoxy(1,5);
   printw("(%2d) Acertos",acertos);
   gotoxy(c,l);
   tecla = getch();
   textcolor(COR_TEMPO);
   gotoxy(30,22);
    printw("Tempo Gasto: %02d:%02d\n",ma,sa);
   textcolor(COR_RELOGIO);
   tempo(56,1);
   if (tecla != UP && tecla != DOWN && tecla != LEFT && tecla != RIGHT)
      {
       textcolor(COR_TEXTO);
       gotoxy(c,l);
       printw("%c",letras[l][c]);
      }
   if (tecla == ENTER)
      {
       if (click)
          {
           l1 = l;
           c1 = c;
           textcolorlight(COR_PALAVRA);
           gotoxy(c,l);
           printw("%c",letras[l][c]);
           textcolor(COR_TEXTO);
           click = FALSE;
          }
        else
            {
             l2 = l;
             c2 = c;
             click = TRUE;
             if (l1 == l2)
                {
                 troquei = FALSE;
                 if (c2 < c1)
                    {
                     troca(&c1,&c2);
                     troquei = TRUE;
                    }
                 x = 0;
                 for (t = c1;t <= c2;t++)
                     {
                      palavra[x] = letras[l][t];
                      x++;
                     }
                 palavra[x] = (char) NULL;
                 y = procura_palavra(palavra);
                 if (y != -1)
                    {
                     k = 0;
                     j = strlen(frutas[y]);
                     for (i = 0;i < j;i++)
                         {
                          s[k] = frutas[y][i];
                          letras[l][c1+i] = s[k];
                          k++;
                         }
                     s[k] = (char) NULL;
                     textcolorlight(COR_PALAVRA);
```

```
                        gotoxy(c1,l);
                        printw("%s",s);
                        textcolor(COR_TEXTO);
                        w = posicao_palavra(y);
                        textcolor(COR_XIS);
                        gotoxy(2,8+w);
                        printw("x");
                        textcolor(COR_ACERTO);
                        gotoxy(5,8+w);
                        printw("%s",frutas[y]);
                        acertos++;
                        textcolor(COR_TELA);
                        gotoxy(2,5);
                        printw("%2d",acertos);
                        if (acertos == PSORT)
                             final(ma,sa);
                       }
                    else
                        if (!troquei)
                           {
                            letras[l1][c1] = tolower(letras[l1][c1]);
                            gotoxy(c1,l1);
                            printw("%c",letras[l1][c1]);
                           }
                        else
                            {
                             letras[l2][c2] = tolower(letras[l2][c2]);
                             gotoxy(c2,l2);
                             printw("%c",letras[l2][c2]);
                            }
                    }
                }
               }
            switch (tecla)
            {
                     case UP: l--;
                              if (l < 5)
                                 l = 20;
                              break;
               case DOWN: l++;
                           if (l > 20)
                               l = 5;
                           break;
               case LEFT: c--;
                          if (c < 20)
                              c = 60;
                          break;
              case RIGHT: c++;
                          if (c > 60)
                              c = 20;
                          break;
            }
       } while (tecla != 27);
 finaliza_janelas();
 return(1);
}

// ------------------------------------------------------- tela

void tela(void)
{
 int c,l;

 for (l = 0;l < NPAL;l++)
     temp[l] = -1;
 textcolor(COR_TELA);
 gotoxy(33,1);
 printw("CA%cA PALAVRAS",199);
 gotoxy(2,22);
 printw("[ESC] Abandona");
 refresh();
 for (l = 5;l <= 20;l++)
     for (c = 20;c <= 60;c++)
         letras[l][c] = RANDOM(97,25);
 gotoxy(20,24);
```

```
 printw("-> Use as setas para procurar palavras <-",231);
 tempo(56,1);
 fp = fopen(nome_arquivo,"r+b");
 if (fp != NULL)
    {
     fread(&recorde,1,sizeof(recorde),fp);
     textcolor(COR_RECORDE);
     gotoxy(20,3);
     printw("Recordista: ");
     gotoxy(20,4);
     printw("Tempo: ");
     textcolor(COR_NOME);
     gotoxy(32,3);
     printw("%s",recorde.nome);
     gotoxy(27,4);
     printw("%02d:%02d",recorde.minutos,recorde.segundos);
     fclose(fp);
    }
}

// ------------------------------------------------------- sorteia_palavras

void sorteia_palavras(int n)
{
        int i,j,t,x,y,z,h = 0;
        int c,l;

        for (i = 0;i < NPAL;i++)
            {
                 sit[i] = 'N';
                 acertou[i] = TRUE;
                }
        for (i = 0;i <= 20;i++)
                linha[i] = -1;
        i = 1;
        l = RANDOM(5,15);
        linha[l] = l;
        do {
                j = RANDOM(0,NPAL-1);
                if (sit[j] == 'N')
                {
                        t = strlen(frutas[j]);
                        c = RANDOM(20,30);
                        y = c + t;
                    if (y > 60)
                                c = c - t;
                        x = 0;
                        for (z = c;z < c+t;z++)
                            {
                                 letras[l][z] = frutas[j][x];
                                 x++;
                            }
                        sit[j] = 'S';
                        temp[h] = j;
                        h++;
                        i++;
                        l = RANDOM(5,15);
                        while (linha[l] != -1)
                              {
                                   l++;
                                   if (l >= 20)
                                           l = 5;
                              }
                        linha[l] = l;
                }
        } while (i <= n);
}

// ------------------------------------------------------- exibe_palavras

void exibe_palavras(void)
{
        int c,l;

        textcolor(COR_TEXTO);
```

```
        for (l = 5;l <= 20;l++)
                for (c = 20;c <= 60;c++)
                {
                        gotoxy(c,l);
                        printw("%c",letras[l][c]);
                }
}

// ------------------------------------------------------- exibe_palavras_sorteadas

void exibe_palavras_sorteadas(void)
{
        int t,l = 8;

        textcolorlight(COR_PALAVRAS);
        for (t = 0;t < NPAL;t++)
                if (temp[t] != -1)
                   {
                        gotoxy(1,l);
                    printw("[ ] %s",frutas[temp[t]]);
                    l++;
                   }
    textcolor(COR_TEXTO);
}

// ------------------------------------------------------- procura_palavra

int procura_palavra(char *s)
{
 int t,i,j,n;
 char r[10];

 n = strlen(s);
 for (i = 0;i < n;i++)
     s[i] = toupper(s[i]);
 for (t = 0;t < NPAL;t++)
     {
         n = strlen(frutas[t]);
         j = 0;
         for (i = 0;i < n;i++)
             {
              r[j] = toupper(frutas[t][i]);
              j++;
             }
        r[j] = (char) NULL;
         i = (int) strcmp(r,s);
         if (i == 0 && acertou[t])
            {
             acertou[t] = FALSE;
             return(t);
            }
    }
 return(-1);
}

// ------------------------------------------------------- final

void final(int min, int seg)
{
 char tecla;
 int c,l,i;
 int segundos_recorde,segundos_atual;

 fp = fopen(nome_arquivo,"r+b");
 if (fp == NULL)
        {
         fp = fopen(nome_arquivo,"w+b");
         if (fp != NULL)
            {
             textcolor(COR_RECORDE);
             gotoxy(20,22);
             printw("Qual o seu Nome: ______________________________");
                 STRING(37,22,30,dados.nome);
                 dados.minutos = min;
             dados.segundos = seg;
```

```
            fwrite(&dados,1,sizeof(dados),fp);
               novo_recorde(dados.nome,dados.minutos,dados.segundos);
               fclose(fp);
              }
       }
 else
     {
         segundos_recorde = recorde.segundos + recorde.minutos * 60;
         segundos_atual = seg + min * 60;
         if (segundos_atual < segundos_recorde)
            {
                textcolor(COR_RECORDE);
             gotoxy(20,22);
             printw("Qual o seu Nome: _______________________________");
             STRING(37,22,30,dados.nome);
             dados.minutos = min;
             dados.segundos = seg;
             fwrite(&dados,1,sizeof(dados),fp);
             fclose(fp);
             novo_recorde(dados.nome,dados.minutos,dados.segundos);
            }
     }
 textcolorlight(COR_FINAL);
 for (l = 5;l <= 20;l++)
      for (c = 20;c <= 60;c++)
             {
              gotoxy(c,l);
              printw(" -> Winner");
              for (i = 0;i <= 900000;i++); // pausa
             }
 textcolor(COR_TEXTO);
 apaga_linha(20,22,50);
 gotoxy(27,22);
 printf("TEMPO GASTO [%02d:%02d]",min,seg);
 apaga_linha(20,24,41);
 gotoxy(24,24);
 printw("Tecle [ENTER] para finalizar");
 do {
         tecla = getch();
 } while (tecla != ENTER && tecla != ESC);
 system("clear");
 endwin();
 exit(1);
}

// ------------------------------------------------------- novo_recorde

void novo_recorde(char *s, int min, int seg)
{
 textcolor(COR_RECORDE);
 gotoxy(20,3);
 printw("Recordista: ");
 gotoxy(20,4);
 printw("Tempo: ");
 apaga_linha(20,22,50);
 textcolorlight(COR_PARABENS);
 gotoxy(30,22);
 printw("NOVO RECORDE, PARAB%cNS",201);
 textcolor(COR_NOME);
 apaga_linha(32,3,30);
 gotoxy(32,3);
 printw("%s",s);
 gotoxy(27,4);
 printw("%02d:%02d",min,seg);
}

// ------------------------------------------------------- troca

void troca(int *x, int *y)
{
 int t;

 t = *x;
 *x = *y;
 *y = t;
```

```
}

// ------------------------------------------------------- tempo

void tempo(int c, int l)
{
  time_t now;
  char s[30];

  time(&now);
  strcpy(s,ctime(&now));
  gotoxy(c,l);
  printw("%s",s);
}

// ------------------------------------------------------- relogio

void relogio(int *h, int *m, int *s)
{
  time_t now;
  char st[30],temp[5];
  int i,x;

  time(&now);
  strcpy(st,ctime(&now));
  x = 0;
  for (i = 11;i <= 12;i++)
      {
       temp[x] = st[i];
       x++;
      }
  temp[x] = (char) NULL;
  *h = atoi(temp);
  x = 0;
  for (i = 14;i <= 15;i++)
      {
       temp[x] = st[i];
       x++;
      }
  temp[x] = (char) NULL;
  *m = atoi(temp);
  x = 0;
  for (i = 17;i <= 18;i++)
      {
       temp[x] = st[i];
       x++;
      }
  temp[x] = (char) NULL;
  *s = atoi(temp);
}

// ------------------------------------------------------- posicao_palavra

int posicao_palavra(int pos)
{
 int i;

 for (i = 0;i < PSORT;i++)
     if (temp[i] == pos)
         return(i);
 return(-1);
}

// ------------------------------------------------------- STRING

void STRING(int c, int l, int n, char *s)
{
 char tecla;
 int i = 0;

 do {
     gotoxy(c,l);
     tecla = getch();
     gotoxy(c,l);
     printw("  ");
```

```
     gotoxy(c,l);
     if ((tecla >= 'A' && tecla <= 'Z') || (tecla >= 'a' && tecla <= 'z') || tecla == 32)
        {
                        gotoxy(c,l);
                        printw("%c",tecla);
                        s[i] = tecla;
                        i++;
                        c++;
                     }
                  if (tecla == 127 && i >= 1)
                     {
                         i--;
                         s[i] = (char) NULL;
                         c--;
                      gotoxy(c,l);
                         printw("   ");

                     }
                  } while (tecla != ENTER && i < n);
        s[i] = (char) NULL;
}

// ------------------------------------------------------- apaga_linha

void apaga_linha(int c, int l, int n)
{
 int i;

 gotoxy(c,l);
 for (i = 1;i <= n;i++)
     printw(" ");
    }
```

19.2 Relógio em Linux

O seguinte programa demonstra a utilização da biblioteca "**time.h**" exibindo um relógio na tela.

A biblioteca "time.h" possui dois tipos de dados que suportam informação temporal:

time_t: possui a informação de calendário (data + hora) num único número que representa um instante de tempo. Este número representa o total de segundos que passaram desde as zero horas de 1 de Janeiro de 1970 (a data zero para os sistemas UNIX).

struct tm: é usado para representar um instante de tempo no formato dividido por classes (dia, mês, ano, hora, data, etc).

```
struct tm {
           int tm_sec;         // segundos [0..59]
           int tm_min;         // minutos [0..59]
           int tm_hour;        // hours [0..23]
           int tm_mday;        // dia do mês [1..31]
           int tm_mon;         // mês (meses desde Janeiro) [0..11]
           int tm_year;        // ano (desde 1900) [0..]
           int tm_wday;        // dia da semana (desde Domingo) [0..6]
           int tm_yday;        // dia do ano (desde 1 Janeiro) [0..365]
           int tm_isdst;       // informação sobre mudança da hora
          };
```

A função **time**() obtém a informação do relógio do sistema operacional Linux e retorna no formato **time_t**.

Protótipo da função: time_t time(time_t *t);

A função **localtime**() obtém a informação do relógio do sistema operacional Linux no formato **struct tm**.

Sintaxe da função: struct tm **localtime(time_t *t);**

```
// relogio.c

#include <stdio.h>
#include <time.h>
#include <stdlib.h>

#define ESC 27

int main(void)
{
 time_t tempo;
 struct tm *data;
 int h,m,s;
 char tecla;

 do {
     time(&tempo);
     printf("Data em segundos: %ld\n", (long) tempo);
     data = localtime(&tempo);
     printf("Data de Hoje\n");
     printf(" Dia: %d\n", data->tm_mday);
     printf(" Mes: %d\n", data->tm_mon+1);
     printf(" Ano: %d\n", data->tm_year+1900);
     printf("Horario Atual:\n");
     h = data->tm_hour;
     m = data->tm_min;
     s = data->tm_sec;
     printf(" Horas: %d\n",h);
     printf(" Minutos: %d\n",m);
     printf(" Segundos: %d\n",s);
     printf("[ESC] - Abandona - [ENTER] - Atualiza: ");
     tecla = getchar();
    } while (tecla != ESC);
}
```

19.3 Biblioteca “conio.h” + “clock.c”

Abaixo é mostrado uma biblioteca que simula as principais funções da biblioteca “**conio.h**”, tais como: **clrscr**, **gotoxy**, **flushall**, **random**, **randomize**, **textcolor**, **textcolorlight** e **kbhit.**

Protótipos da Funções desta biblioteca:

```
void clrscr(void);
void gotoxy(int c, int l);
void flushall(void);
int random(int n);
void randomize(void);
void textcolor(int cor);
void textcolorlight(int cor);
int kbhit(void);
```

```
// conio.h

// ------------------------------ includes

#include <curses.h>
#include <signal.h>
#include <time.h>
#include <sys/select.h>

// ------------------------------------------------------------ cores

#define BLACK         COLOR_BLACK
#define RED           COLOR_RED
#define GREEN         COLOR_GREEN
#define YELLOW        COLOR_YELLOW
#define BLUE          COLOR_BLUE
#define MAGENTA       COLOR_MAGENTA
#define CYAN          COLOR_CYAN
#define WHITE         COLOR_WRITE

// ------------------------------------------------------------ prototypes

void initconio(void);
void endconio(void);
void clrscr(void);
void gotoxy(int c, int l);
void flushall(void);
int random(int n);
void randomize(void);
void textcolor(int cor);
void textcolorlight(int cor);
int kbhit(void);

// ------------------------------------------------------------ initconio

void initconio(void)
{
 initscr();
 keypad(stdscr, TRUE);
 (void) nonl();
 (void) cbreak();
 (void) echo();
 if (has_colors())
    {
     start_color();
     init_pair(1,COLOR_RED,COLOR_BLACK);
     init_pair(2,COLOR_GREEN,COLOR_BLACK);
     init_pair(3,COLOR_YELLOW,COLOR_BLACK);
     init_pair(4,COLOR_BLUE,COLOR_BLACK);
     init_pair(5,COLOR_CYAN,COLOR_BLACK);
     init_pair(6,COLOR_MAGENTA,COLOR_BLACK);
     init_pair(7,COLOR_WHITE,COLOR_BLACK);
     attrset(COLOR_PAIR(A_BOLD));
    }
}

// ------------------------------------------------------------ endconio

void endconio(void)
{
 endwin();
 system("clear");
}

// ------------------------------------------------------------ clrscr

void clrscr(void)
{
 clear();
 refresh();
}

// ------------------------------------------------------------ gotoxy

void gotoxy(int c, int l)
```

```
{
 move(l-1,c-1);
 refresh();
}

// ------------------------------------------------------- flushall

void flushall(void)
{
 getchar();
}

// ------------------------------------------------------- random

int random(int n)
{
 int t;

 t = rand() % n;
 return(t);
}

// ------------------------------------------------------- randomize

void randomize(void)
{
 srand((unsigned int)time((time_t *)NULL));
}

// ------------------------------------------------------- textcolor

void textcolor(int cor)
{
 attrset(COLOR_PAIR(cor));
}

// ------------------------------------------------------- textcolorlight

void textcolorlight(int cor)
{
 attrset(COLOR_PAIR(cor)+A_BOLD);
}

// ------------------------------------------------------- kbhit

int kbhit(void)
{
 struct timeval tv;
 fd_set read_fd;

 tv.tv_sec=0;
 tv.tv_usec=0;
 FD_ZERO(&read_fd);
 FD_SET(0,&read_fd);
 if (select(1, &read_fd,NULL,NULL,&tv) == -1)
    return(0);
 if (FD_ISSET(0,&read_fd))
    return(1);
 return(0);
}
```

Programa Principal: clock.c

```
// clock.c

// ------------------------------------------------------- includes

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

// ------------------------------------------------------- main

int main(void)
{
```

```
 time_t tempo;
 struct tm *data;
 int h,m,s;
 char tecla;

 initconio();  // função obrigatória
 clrscr();
 textcolor(BLUE);
 do {
     time(&tempo);
     data = localtime(&tempo);
     h = data->tm_hour;
     m = data->tm_min;
     s = data->tm_sec;
     gotoxy(1,1);
     printw("Hora: %02d:%02d:%02d",h,m,s);
    } while (!kbhit());
 endconio();   // função obrigatória
}
```

19.4 Biblioteca "conio.h" completa

Abaixo é mostrado uma biblioteca que simula as principais funções da biblioteca "**conio.h**".

Protótipos das Funções desta biblioteca:

```
void initconio(void);
void doneconio(void);

void nocursor(void);
void  cursor(void);

int wherex(void);
int wherey(void);
int putch(int c);
int getche(void);
int key(void);
int kbhit(void);
int cprintf(char *format, ...);
int cscanf(const char *format, ...);
void clreol(void);
void clrscr(void);
void gotoxy(int x, int y);
void delline(void);
void gettextinfo(struct text_info *r);
void highvideo(void);
void insline(void);
void lowvideo(void);
void normvideo(void);
void textattr(int attribute);
void textbackground(int color);
void textcolor(int color);
void textmode(int unused_mode);
void window(int left, int top, int right, int bottom);
void cputs(char *str);
char *cgets(char *str);
```

Biblioteca: conio.h

```
// conio.h
```

```
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <curses.h>
#include <pwd.h>

int directvideo;
char color_warning;
WINDOW *conio_scr;

struct text_info {
                  unsigned char winleft;
                  unsigned char wintop;
                  unsigned char winright;
                  unsigned char winbottom;
                  unsigned char attribute;
                  unsigned char normattr;
                  unsigned char currmode;
                  unsigned char screenheight;
                  unsigned char screenwidth;
                  unsigned char curx;
                  unsigned char cury;
                 };

enum text_modes {LASTMODE=-1, BW40=0, C40, BW80, C80, MONO=7, C4350=64};

enum COLORS {BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LIGHTGRAY,
   DARKGRAY, LIGHTBLUE, LIGHTGREEN, LIGHTCYAN, LIGHTRED, LIGHTMAGENTA, YELLOW, WHITE};

#define BLINK 128

void initconio(void);
void doneconio(void);

void nocursor(void);
void  cursor(void);

int wherex(void);
int wherey(void);
int putch(int c);
int getche(void);
int key(void);
int kbhit(void);
int cprintf(char *format, ...);
int cscanf(const char *format, ...);

unsigned inp(unsigned port);
unsigned inpw(unsigned port);
unsigned outp(unsigned port, unsigned value);
unsigned outpw(unsigned port,unsigned value);
unsigned inpd(unsigned port);
unsigned outpd(unsigned port, unsigned value);

void clreol(void);
void clrscr(void);
void gotoxy(int x, int y);
void delline(void);
void gettextinfo(struct text_info *r);
void highvideo(void);
void insline(void);
void lowvideo(void);
void normvideo(void);
void textattr(int attribute);
void textbackground(int color);
void textcolor(int color);
void textmode(int unused_mode);
void window(int left, int top, int right, int bottom);
void cputs(char *str);

char *cgets(char *str);

static int txtattr,oldattr;
static int fgc,bgc;
```

```
static int initialized=0;
char color_warning=1;
int directvideo;
WINDOW *conio_scr;

int colortab(int a)
{
 switch(a)
 {
  case 0 : return COLOR_BLACK;
  case 1 : return COLOR_BLUE;
  case 2 : return COLOR_GREEN;
  case 3 : return COLOR_CYAN;
  case 4 : return COLOR_RED;
  case 5 : return COLOR_MAGENTA;
  case 6 : return COLOR_YELLOW;
  case 7 : return COLOR_WHITE;
 }
}

void docolor (int color)
{
 wattrset(conio_scr,0);
 if ((color&128)==128)
     txtattr=A_BLINK;
 else
     txtattr=A_NORMAL;
 if ((color&15)>7)
     txtattr|=A_STANDOUT;
 else
     txtattr|=A_NORMAL;
 txtattr|=COLOR_PAIR((1+(color&7)+(color&112)) >> 1);
 if (((color&127)==15) | ((color&127)==7))
    txtattr=COLOR_PAIR(0);
 if (((color&127)==127) | ((color&127)==119))
    txtattr=COLOR_PAIR(8);
 wattron(conio_scr,txtattr);
 wattron(conio_scr,COLOR_PAIR(1+(color&7)+((color&112)>>1)));
}

static inline int inport (int port)
{
 unsigned char value;
 __asm__ volatile ("inb %1,%0"
                   : "=a" (value)
                   : "d" ((unsigned short) port));
 return value;
}

static inline int inportd (int port)
{
 unsigned int value;
 __asm__ volatile ("inl %1,%0"
                   : "=a" (value)
                   : "d" ((unsigned short)port));
 return value;
}

static inline int inportw (int port)
{
 unsigned short value;
 __asm__ volatile ("inw %1,%0"
                   : "=a" (value)
                   : "d" ((unsigned short)port));
 return value;
}

static inline void outport (unsigned short int port, unsigned char val)
{
 __asm__ volatile (
                   "outb %0,%1\n"
                   :
                   : "a" (val), "d" (port)
                   );
}
```

```
static inline void outportw (unsigned short int port, unsigned int val)
{
  __asm__ volatile (
                    "outw %0,%1\n"
                    :
                    : "a" (val), "d" (port)
                    );
}

static inline void outportd (unsigned short int port, unsigned int val)
{
  __asm__ volatile (
                    "outl %0,%1\n"
                    :
                    : "a" (val), "d" (port)
                    );
}

void initconio (void)
{
 int x,y;

 attr_t dummy1;
 short dummy2;
 initialized=1;
 initscr();
 start_color();
 oldattr=wattr_get(stdscr,&dummy1,&dummy2,NULL);
 nonl();
 raw();
 if (!has_colors() & (color_warning>-1))
    fprintf(stderr,"Atenção: Problema com a cor\n");
 noecho();
 conio_scr=newwin(0,0,0,0);
 keypad(conio_scr,TRUE);
 meta(conio_scr,TRUE);
 idlok(conio_scr,TRUE);
 scrollok(conio_scr,TRUE);
 for (y=0;y<=7;y++)
      for (x=0;x<=7;x++)
         init_pair((8*y)+x+1, colortab(x), colortab(y));
 txtattr=wattr_get(conio_scr,&dummy1,&dummy2,NULL);
 bgc=0;
 textcolor(7);
 textbackground(0);
}

void doneconio (void)
{
 endwin();
}

char *cgets (char *str)
{
 char strng[257];
 unsigned char i = 2;

 if (initialized == 0)
    initconio();
 echo();
 strncpy(strng,str,1);
 wgetnstr(conio_scr,&strng[2],(int) strng[0]);
 while (strng[i] != 0)
       i++;
 i = i - 2;
 strng[1] = i;
 strcpy(str,strng);
 noecho();
 return(&str[2]);
}

void clreol (void)
{
 if (initialized == 0)
```

```
    initconio();
 wclrtoeol(conio_scr);
 wrefresh(conio_scr);
}

void clrscr (void)
{
 if (initialized == 0)
    initconio();
 wclear(conio_scr);
 wmove(conio_scr,0,0);
 wrefresh(conio_scr);
}

int cprintf (char *format, ... )
{
 int i;
 char buffer[BUFSIZ];
 va_list argp;

 if (initialized==0)
    initconio();
 va_start(argp,format);
 vsprintf(buffer,format,argp);
 va_end(argp);
 i = waddstr(conio_scr,buffer);
 wrefresh(conio_scr);
 return(i);
}

void cputs (char *str)
{
 if (initialized == 0)
    initconio();
 waddstr(conio_scr,str);
 wrefresh(conio_scr);
}

int cscanf (const char *format, ...)
{
 int i;
 char buffer[BUFSIZ];
 va_list argp;

 if (initialized == 0)
    initconio();
 echo();
 if (wgetstr(conio_scr,buffer) == ERR)
    return(-1);
 va_start(argp,format);
 i = vsscanf(buffer,format,argp);
 va_end(argp);
 noecho();
 return(i);
}

void delline (void)
{
 if (initialized == 0)
    initconio();
 wdeleteln(conio_scr);
 wrefresh(conio_scr);
}

void cursor(void)
{
 curs_set(1);
}

void nocursor(void)
{
 curs_set(0);
}

int key(void)
```

```
{
 int i;

 if (initialized == 0)
    initconio();
 i=wgetch(conio_scr);
 if (i == -1)
    i = 0;
 return i;
}

int getche (void)
{
 int i;

 if (initialized == 0)
    initconio();
 echo();
 i = wgetch(conio_scr);
 if (i == -1)
    i=0;
 noecho();
 return(i);
}

void gettextinfo(struct text_info *inforec)
{
 attr_t dummy1;
 short dummy2;
 unsigned char xp,yp;
 unsigned char x1,x2,y1,y2;
 unsigned char cols,lines;
 unsigned char dattr,dnattr,a;

 if (initialized == 0)
    initconio();
 getyx(conio_scr,yp,xp);
 getbegyx(conio_scr,y1,x1);
 getmaxyx(conio_scr,y2,x2);
 dattr = (bgc*16)+fgc;
 a = wattr_get(conio_scr,&dummy1,&dummy2,NULL);
 if (a == (a & A_BLINK))
    dattr=dattr+128;
 dnattr = oldattr;
 inforec->winleft = x1 + 1;
 inforec->wintop = y1 + 1;
 if (x1==0)
    x2--;
 if (y1==0)
    y2--;
 inforec->winright = x1 + x2 + 1;
 inforec->winbottom = y1 + y2 + 1;
 inforec->curx = xp + 1;
 inforec->cury = yp + 1;
 inforec->screenheight = y2 + 1;
 inforec->screenwidth = x2 + 1;
 inforec->currmode = 3;
 inforec->normattr = dnattr;
 inforec->attribute=dattr;
}

void gotoxy (int x, int y)
{
 if (initialized == 0)
    initconio();
 y--;
 x--;
 wmove(conio_scr,y,x);
 wrefresh(conio_scr);
}

void highvideo (void)
{
 if (initialized == 0)
    initconio();
```

```
 textcolor(15);
 textbackground(0);
}

void insline (void)
{
 if (initialized == 0)
    initconio();
 winsertln(conio_scr);
 wrefresh(conio_scr);
}

int kbhit (void)
{
 int i;

 if (initialized == 0)
    initconio();
 nodelay(conio_scr,TRUE);
 i = wgetch(conio_scr);
 nodelay(conio_scr,FALSE);
 if (i == -1)
    i = 0;
 return(i);
}

void lowvideo (void)
{
 if (initialized == 0)
    initconio();
 textbackground(0);
 textcolor(8);
}

void normvideo (void)
{
 if (initialized == 0)
    initconio();
 wattrset(conio_scr,oldattr);
}

int putch (int c)
{
 if (initialized == 0)
    initconio();
 if (waddch(conio_scr,c) != ERR)
    {
     wrefresh(conio_scr);
     return(c);
    }
 return(0);
}

void textattr (int attr)
{
 if (initialized == 0)
    initconio();
 docolor(attr);
}

void textbackground (int color)
{
 if (initialized == 0)
    initconio();
 bgc = color;
 color = (bgc * 16) + fgc;
 docolor(color);
}

void textcolor (int color)
{
 if (initialized == 0)
    initconio();
 fgc = color;
 color = (bgc * 16) + fgc;
```

```
 docolor(color);
}

void textmode (int mode)
{
 if (initialized == 0)
    initconio();
}

int wherex (void)
{
 int y;
 int x;

 if (initialized == 0)
    initconio();
 getyx(conio_scr,y,x);
 x++;
 return(x);
}

int wherey (void)
{
 int y;
 int x;

 if (initialized == 0)
    initconio();
 getyx(conio_scr,y,x);
 y++;
 return(y);
}

void window (int left,int top,int right,int bottom)
{
 int nlines,ncols;

 if (initialized == 0)
    initconio();
 delwin(conio_scr);
 top--;
 left--;
 right--;
 bottom--;
 nlines = bottom - top;
 ncols = right - left;
 if (top == 0)
    nlines++;
 if (left == 0)
    ncols++;
 if ((nlines<1) | (ncols<1))
    return;
 conio_scr = newwin(nlines,ncols,top,left);
 keypad(conio_scr,TRUE);
 meta(conio_scr,TRUE);
 idlok(conio_scr,TRUE);
 scrollok(conio_scr,TRUE);
 wrefresh(conio_scr);
}
```

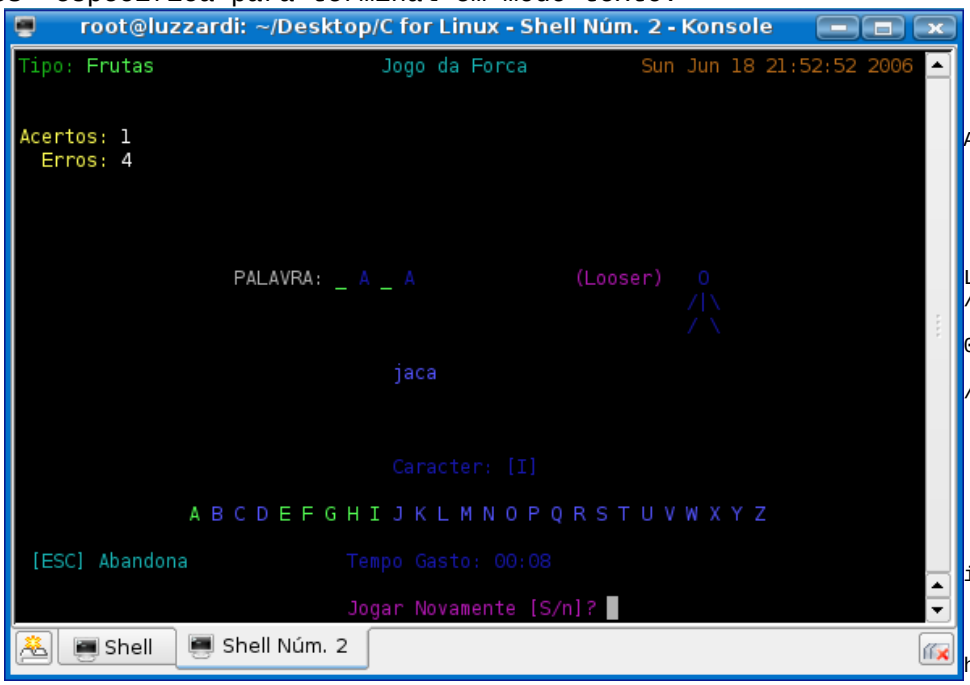
Programa: cores.c

```
// cores.c

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

int main(void)
{
 int cor;

 initconio();  // função obrigatória
 clrscr();
 nocursor();
```

```
 for (cor = 1;cor <= 15;cor++)
        {
         textcolor(cor);
         gotoxy(10,10);
         cprintf("Universidade Católica de Pelotas");
         getche();
        }
 cursor();
    doneconio();      // função obrigatória
   }
```

19.5 Jogo da Forca

O jogo da **forca** demonstra também a utilização de biblioteca **"ncurses"** específica para terminal em modo texto.




```
//
 forca.c
(Kubuntu
-
Anjuta)
//
 Autor:
  Paulo
 Roberto
  Gomes
Luzzardi
//  Data:
 17/06/20
06

//  -----
 --------
 --------
 --------
 --------
 --------
 --------
     ----
includes

 #include
 <string.
h>
 #include
<curses.h>
#include <stdlib.h>
#include <time.h>

// ------------------------------------------------------- definicoes

#define ESC    27
#define ENTER  13
#define SPACE  32

#define LEFT  260
#define RIGHT 261

#define BLACK       COLOR_BLACK
#define RED         COLOR_BLUE
#define GREEN       COLOR_GREEN
#define BROWN       COLOR_YELLOW
#define BLUE        COLOR_BLUE
#define MAGENTA     COLOR_MAGENTA
#define CYAN        COLOR_CYAN
#define WHITE       COLOR_WHITE

#define COR_TEXTO      GREEN
#define COR_TELA       BROWN
#define COR_TEMPO      RED
#define COR_RELOGIO    BROWN
#define COR_WINNER     CYAN
#define COR_LETRAS     BLUE
```

```
#define COR_SELECAO    RED
#define COR_TITULO     MAGENTA
#define COR_STATUS     WHITE

// ------------------------------------------------------- prototypes

void tela(void);
void final(int min, int seg);
void troca(int *x, int *y);
void tempo(int c, int l);
void relogio(int *h, int *m, int *s);
void STRING(int c, int l, int n, char *s);
void apaga_linha(int c, int l, int n);
void boneco(int erro);

// ------------------------------------------------------- prototypes

void inicializa_janelas(void);
void finaliza_janelas(void);
void clrscr(void);
void gotoxy(int c, int l);
void flushall(void);
int Random(int n);
void randomize(void);
void textcolor(int cor);
void textcolorlight(int cor);

// ------------------------------------------------------- inicializa_janelas

void inicializa_janelas(void)
{
 initscr();
 keypad(stdscr, TRUE);
 (void) nonl();
 (void) cbreak();
 (void) echo();
 if (has_colors())
    {
     start_color();
     init_pair(1,COLOR_RED,COLOR_BLACK);
     init_pair(2,COLOR_GREEN,COLOR_BLACK);
     init_pair(3,COLOR_YELLOW,COLOR_BLACK);
     init_pair(4,COLOR_BLUE,COLOR_BLACK);
     init_pair(5,COLOR_CYAN,COLOR_BLACK);
     init_pair(6,COLOR_MAGENTA,COLOR_BLACK);
     init_pair(7,COLOR_WHITE,COLOR_BLACK);
     attrset(COLOR_PAIR(A_BOLD));
    }
}

// ------------------------------------------------------- finaliza_janelas

void finaliza_janelas(void)
{
 endwin();
 system("clear");
}

// ------------------------------------------------------- clrscr

void clrscr(void)
{
 clear();
 refresh();
}

// ------------------------------------------------------- gotoxy

void gotoxy(int c, int l)
{
 move(l-1,c-1);
 refresh();
}

// ------------------------------------------------------- flushall
```

```
void flushall(void)
{
 getchar();
}

// ------------------------------------------------------- Random

int Random(int n)
{
 int t;

 t = rand() % n;
 return(t);
}

// ------------------------------------------------------- randomize

void randomize(void)
{
 srand((unsigned int)time((time_t *)NULL));
}

// ------------------------------------------------------- textcolor

void textcolor(int cor)
{
 attrset(COLOR_PAIR(cor));
}

// ------------------------------------------------------- textcolorlight

void textcolorlight(int cor)
{
 attrset(COLOR_PAIR(cor)+A_BOLD);
}

// ------------------------------------------------------- tempo

void tempo(int c, int l)
{
  time_t now;
  char s[30];

  time(&now);
  strcpy(s,ctime(&now));
  textcolor(COR_RELOGIO);
  gotoxy(c,l);
  printw("%s",s);
}

// ------------------------------------------------------- relogio

void relogio(int *h, int *m, int *s)
{
  time_t now;
  char st[30],temp[5];
  int i,x;

  time(&now);
  strcpy(st,ctime(&now));
  x = 0;
  for (i = 11;i <= 12;i++)
      {
       temp[x] = st[i];
       x++;
      }
  temp[x] = (char) NULL;
  *h = atoi(temp);
  x = 0;
  for (i = 14;i <= 15;i++)
      {
       temp[x] = st[i];
       x++;
      }
```

```
  temp[x] = (char) NULL;
  *m = atoi(temp);
  x = 0;
  for (i = 17;i <= 18;i++)
      {
       temp[x] = st[i];
       x++;
      }
  temp[x] = (char) NULL;
  *s = atoi(temp);
}

// ------------------------------------------------------------ variaveis

unsigned short letras[21][61];

#define NPAL  30
#define PSORT  8

#define NTIPOS 4

char nomes[NTIPOS][10] = {"Cidades","Países","Carros","Frutas"};

char  cidades[NPAL][11]  =  {"pelotas",  "alegrete",  "uruguaiana",  "livramento",  "chui",
"rivera",  "chuy",  "piratini",  "candiota",  "marau",  "erechin",  "cachoerinha",  "caxias",
"gramado",  "canela",  "itaqui",  "cacequi",  "formigueiro",  "parati",  "buzios",  "torres",
"mostarda", "tapes", "agudo","candiota","chiapeta","cassino","casca","cristal","cidreira"};

char  paises[NPAL][11]  =  {"brasil",  "peru",  "uruguai",  "argentina",  "bolivia",  "chile",
"guatemala",  "cuba",  "alemanha",  "suecia",  "inglaterra",  "china",  "russia",  "montenegro",
"tibet",   "nepal",   "servia",   "noruega",   "portugal",   "espanha",   "grecia",   "mongolia",
"escocia", "palestina", "israel", "iraqui","australia","jamaica","egito","congo"};

char carros[NPAL][11] = {"mercedes", "fusca", "dodge", "palio", "omega", "kombi", "fiat",
"ford",  "dakota",  "bmw",  "ka",  "elba",  "gol",  "golf",  "vectra",  "sportage",  "idea",
"corcel",   "parati",   "saveiro",   "peuget",   "citroen",   "toyota",   "nissan",   "renaut",
"frontier", "honda", "subaru", "corrola", "civic"};

char  frutas[NPAL][11]  =  {"abacaxi",  "banana",  "ameixa",  "pera",  "goiaba",  "amora",
"bergamota",  "morango",  "lima",  "abacate",  "carambola",  "laranja",  "kiwi",  "tangerina",
"figo", "pitanga", "framboesa", "acerola", "cereja", "nectarina", "pomelo", "caqui", "caju",
"marmelo", "uva","nectarina","damasco","manga","jaca","jabuticaba"};

// ------------------------------------------------------------ main

int main(void)
{
 int click, sai, acertou;
 int tecla, i, j, k, m, n;
 int c, l;
 int hi, mi, si;
 int ma, sa, incrementa;
 int ht, mt, st;
 char s[10], ch;
 char palavra[10], vetor[26];
 int row, col, lin, acertos, erros, total;

 inicializa_janelas();
 getmaxyx(stdscr, row, col);  // tamanho do terminal
 do {
        ch = 'A';
        acertos = 0;
        total = 0;
        erros = 0;
        sai = FALSE;
        incrementa = FALSE;
        click = TRUE;
        c = 16;
        l = 20;
        hi = mi = si = 0;
        clrscr();
        textcolor(WHITE);
        gotoxy(60,24);
        printw("| row: %d | col: %d |",row,col);
        randomize();
```

```
 textcolor(COR_TEXTO);
 tela();
 for (i = 0;i <= 25;i++)
        vetor[i] = '*';
 relogio(&hi,&mi,&si);
 ma = 0;
 j = Random(NPAL);
 m = Random(NTIPOS);
 switch (m)
 {
 case 0: strcpy(palavra, cidades[j]);
         break;
 case 1: strcpy(palavra, paises[j]);
         break;
 case 2: strcpy(palavra, carros[j]);
         break;
 case 3: strcpy(palavra, frutas[j]);
         break;
 }
textcolor(GREEN);
gotoxy(1,1);
printw("Tipo: ");
textcolorlight(GREEN);
gotoxy(7,1);
printw("%s",nomes[m]);
n = strlen(palavra);
gotoxy(20,10);
printf("PALAVRA: ");
for (i = 0;i < n;i++)
    printw("_ ");
textcolorlight(BROWN);
gotoxy(1,4);
printw("Acertos: ");
gotoxy(1,5);
printw("  Erros: ");
do {
       relogio(&ht,&mt,&st);
       textcolorlight(COR_STATUS);
       gotoxy(10,4);
       printw("%d",acertos);
       gotoxy(10,5);
       printw("%d",erros);
       sa = st - si;
       if (sa < 0)
          sa = sa + 60;
       if (sa == 58)
          incrementa = TRUE;
       if (sa == 0 && incrementa)
          {
           ma++;
           incrementa = FALSE;
          }
       textcolor(COR_TEMPO);
       gotoxy(30,22);
       printw("Tempo Gasto: %02d:%02d\n",ma,sa);
       gotoxy(c,l);
       tecla = getch();
       if (tecla == ESC)
          sai = TRUE;
       textcolor(COR_TEMPO);
       gotoxy(30,22);
       printw("Tempo Gasto: %02d:%02d\n",ma,sa);
       textcolor(COR_RELOGIO);
       tempo(56,1);
       switch (tecla)
       {
          case LEFT:  c-=2;
                      ch--;
                      if (c < 16)
                         {
                          c = 66;
                          ch = 'Z';
                         }
                      break;
          case RIGHT: c+=2;
```

```
                                ch++;
                                if (c > 66)
                                   {
                                    c = 16;
                                    ch = 'A';
                                   }
                                      break;
                        case ENTER:
                        case SPACE:  textcolor(COR_LETRAS);
                                gotoxy(34,18);
                                printw("Caracter: [%c]",ch);
                                textcolorlight(GREEN);
                                gotoxy(c,l);
                                printw("%c",ch);
                                textcolor(COR_SELECAO);
                                col = 29;
                                lin = 10;
                                acertou = FALSE;
                                k = ch - 65;
                                for (i = 0;i < n;i++)
                                    {
                                     if (toupper(palavra[i]) == ch)
                                         {
                                          gotoxy(col,lin);
                                          printw("%c",ch);
                                          if (!acertou  && vetor[k] == '*')
                                             acertos++;
                                          total++;
                                          vetor[k] = '#';
                                          acertou = TRUE;
                                         }
                                     col = col + 2;
                                   }
                                if (!acertou && vetor[k] == '*')
                                    {
                                     erros++;
                                     vetor[k] = '#';
                                    }
                                boneco(erros);
                                if (total == n)
                                   {
                                    textcolor(COR_WINNER);
                                   gotoxy(50,10);
                                   printw("(Winner)");
                                   gotoxy(30,24);
                                   sai = TRUE;
                                   printw("Jogar Novamente [S/n]? ");
                                   do {
                                       tecla = getch();
                                      } while(!strchr("SsNn",tecla) && tecla != ESC);
                                }
                            break;
                     }
             if (erros == 5)
                {
                     textcolorlight(RED);
                     gotoxy(34,14);
                     printw("%s",palavra);
                     textcolor(COR_WINNER);
                     gotoxy(50,10);
                     printw("(Looser)");
                     gotoxy(30,24);
                     sai = TRUE;
                     printw("Jogar Novamente [S/n]? ");
                     do {
                             tecla = getch();
                        } while(!strchr("SsNn",tecla) && tecla != ESC);
                }
            } while (!sai);
    } while (strchr("Ss",tecla) && tecla != ESC);
  finaliza_janelas();
  return(1);
}

// ------------------------------------------------------------ tela
```

```
void tela(void)
{
 textcolor(COR_TITULO);
 gotoxy(33,1);
 printw("Jogo da Forca",199);
 gotoxy(2,22);
 printw("[ESC] Abandona");
 refresh();
 textcolorlight(COR_LETRAS);
 gotoxy(16,20);
 printw("A B C D E F G H I J K L M N O P Q R S T U V W X Y Z");
 tempo(56,1);
}

// ------------------------------------------------------- boneco

void boneco(int erro)
{
 int col = 60, lin = 10;

 switch (erro)
 {
 case 1: gotoxy(col,lin);
         printw(" O");
         break;
 case 2: gotoxy(col,lin+1);
         printw("/|");
         break;
 case 3: gotoxy(col,lin+1);
         printw("/|\\");
         break;
 case 4: gotoxy(col,lin+2);
         printw("/");
         break;
 case 5: gotoxy(col,lin+2);
         printw("/ \\");
         break;
 }
}
```

20. Biblioteca ncurses

20.1 Conceitos Básicos

No linux é necessário incluir a biblioteca **"curses"** para criar programas que executem em terminais em modo texto. Por linha de comando fica assim:

```
$ gcc palavras.c -o palavras -lcurses <enter>
```

No ambiente de programação **anjuta** tem que incluir em ***"Definições"*** ... ***"Definições de Compilação e Link..."*** ... ***"Bibliotecas"*** ... ***"curses"*** (*Screen handling and optimization routines*) ... "Adicionar".

É uma biblioteca que permite construir uma interface textual em terminais Linux em modo texto, para programa escritos em linguagem de programação C.

- A biblioteca **ncurses** permite a manipulação de janelas em um terminal em modo texto.
- Janelas são definidas como ponteiros para o tipo **WINDOW**.
- A janela padrão é chamada **stdscr**.
- A biblioteca disponibiliza o número de linhas da janela stdscr em **LINES** e o número de colunas em **COLS**.
- A primeira função a ser utilizada, obrigatoriamente é a função **initscr**().
- A última função deve ser **endwin**().
- Para atualizar a janela stdscr utiliza-se a função **refresh**.

Declaração de uma janela: WINDOW *win;

Função: initscr()
Uso: inicializa a biblioteca ncurses
Sintaxe: WINDOW *initscr(void);
Retorno: nada

Função: endwin()
Uso: finaliza a biblioteca ncurses
Sintaxe: int endwin(void);
Retorno: OK (sucesso) ou ERR (erro)

Função: printw()
Uso: imprime no terminal
Sintaxe: int printw(char *format, ...);
Retorno: OK (sucesso)

Função: refresh()
Uso: atualiza a janela
Sintaxe: int refresh(void);
Retorno: OK (sucesso) ou ERR (erro)

```
#include <curses.h>

int main(void)
{
 initscr();       // inicializa a biblioteca ncurses
```

```
 printw("Biblioteca ncurses\n");
 printw("Data: %02d/%02d/%02d\n",18,6,2006);
 refresh();
 getchar();

 endwin();        // finaliza a biblioteca ncurses
}
```

Função: getmaxyx()
Uso: verificar o número de linhas e colunas do terminal
Sintaxe: void getmaxyx(WINDOW *win, int row, int col);
Retorno: nada

Função: mvprintw()
Uso: imprime no terminal
Sintaxe: int mvprintw(int lin, int col, char *format, ...);
Retorno: OK (sucesso)

Função: cbreak
Uso: Desabilita o cache de teclado, fazendo com que toques das teclas fiquem disponíveis para o programa curses assim que realizados.
Sintaxe: int cbreak(void);
Retorno: OK (sucesso) ou ERR (erro)

Função: keypad()
Uso: Habilita ou desabilita a leitura de teclas especiais do teclado
Sintaxe: int keypad(WINDOW *win, bool buffer);
Retorno: OK (sucesso) ou ERR (erro)
Observação: buffer = TRUE -> habilita as teclas especiais
buffer = FALSE -> desabilita as teclas especiais

Função: clear()
Uso: limpa a janela no terminal
Sintaxe: int clear(void);
Retorno: OK (sucesso)

```
#include <curses.h>

int main(void)
{
 int row, col;

 initscr();
 clear();
 getmaxyx(stdscr,row,col);
 mvprintw(1, 1, "| row: %d | col: %d | ",row, col);
 mvprintw(2, 1, "| LINES: %d | COLS: %d | ",LINES, COLS);
 refresh();
 getchar();
 endwin();
}
```

Função: move()
Uso: move o cursor para a posição (lin, col)
Sintaxe: int move(int lin, int col);
Retorno: OK (sucesso) ou ERR (erro)

```
move(10,7);
printw("Jogo da Forca");
refresh();
```

Função: getch()
Uso: Lê um caractere no teclado
Sintaxe: int getch(void);
Retorno: caracter ou ERR (erro)
Observação: Em modo nodelay, retorna o caracter imediatamente (se nenhuma tecla foi pressionada, retorna ERR). No modo delay espera até que uma tecla seja pressionada

Função: nonl()
Uso: Desabilita a conversão da tecla enter em '\n' ao ler o teclado
Sintaxe: int nonl(void);
Retorno: OK (sucesso) ou ERR (erro)

Função: noecho()
Uso: Desabilita o eco de teclas pressionadas para a tela (na aparece na tela as teclas digitadas)
Sintaxe: int noecho(void);
Retorno: OK (sucesso) ou ERR (erro)

Função: nodelay()
Uso: Habilita ou desabilita o modo nodelay, em que qualquer toque de tecla é enviado ao programa imediatamente (em vez de esperar um enter)
Sintaxe: int nodelay(WINDOW *win, bool buffer);
Retorno: OK (sucesso) ou ERR (erro)
Observação: buffer = TRUE -> habilita o modo nodelay
buffer = FALSE -> desabilita o modo nodelay

20.2 Funções de inicialização

initscr(): primeira função obrigatória é a função.

keypad(): Habilita a leitura das teclas **enter**, **esc**, **setas**, **F1.**

Habilitar: keypad(stdscr,**TRUE**);
Desabilitar: keypad(stdscr,**FALSE**);

cbreak(): desabilita a propriedade do terminal em ficar esperando o usuário digitar **enter.**

curs_set(): utilizado para definir o tipo de cursor:

(0)cursor invisível
(1)cursor visível
(2)cursor muito visível.

echo() ou noecho(): com **echo()** as teclas digitadas aparecem no terminal, caso contrário, em **noecho()** as teclas digitadas não aparecem na tela.

20.3 Funções de entrada

Existem três tipos de funções de entrada na biblioteca ncurses:

- **getch()**: Lê um caracter

- **scanw()**: Lê uma string formatada
- **getstr()**: Lê strings

getch(): lê um caracter do teclado

```
int getch(void);
int wgetch(WINDOW *win);
int mvgetch(int y, int x);
int mvwgetch(WINDOW *win, int y, int x);
```

Estas funções permitem ler um único caractere do terminal. Elas dependem das funções: **cbreak** e **noecho**.

cbreak(); a leitura dos caracteres só começa quando a tecla ***enter*** ser pressionada
noecho(); se esta função estiver ativa, os caracteres digitados não aparecerão na tela

scanw(): Lê uma string formatada

```
int scanw(char *fmt [, arg] ...);
int wscanw(WINDOW *win, char *fmt [, arg] ...);
int mvscanw(int y, int x, char *fmt [, arg] ...);
int mvwscanw(WINDOW *win, int y, int x, char *fmt [, arg] ...);
```

getstr(): Lê string via teclado

```
int getstr(char *str);
int getnstr(char *str, int n);
int wgetstr(WINDOW *win, char *str);
int wgetnstr(WINDOW *win, char *str, int n);
int mvgetstr(int y, int x, char *str);
int mvwgetstr(WINDOW *win, int y, int x, char *str);
int mvgetnstr(int y, int x, char *str, int n);
int mvwgetnstr(WINDOW *, int y, int x, char *str, int n);
```

20.4 Funções de saída

Existem três tipos de funções de saída na biblioteca ncurses:

- **addch()**: imprime caracteres com atributos
- **printw()**: imprime *strings* formatadas
- **addstr()**: imprime *strings*

addch(): Imprime um caracter na tela

```
int addch(chtype ch);
int waddch(WINDOW *win, chtype ch);
int mvaddch(int y, int x, chtype ch);
int mvwaddch(WINDOW *win, int y, int x, chtype ch);
int echochar(chtype ch);
int wechochar(WINDOW *win, chtype ch);
```

printw(): Imprime uma string na tela

```
int printw(char *fmt [, arg] ...);
int wprintw(WINDOW *win, char *fmt [, arg] ...);
int mvprintw(int y, int x, char *fmt [, arg] ...);
```

```
int mvwprintw(WINDOW *win, int y, int x, char *fmt [, arg] ...);
```

addstr(): Imprime uma string na tela

```
int addstr(const char *str);
int addnstr(const char *str, int n);
int waddstr(WINDOW *win, const char *str);
int waddnstr(WINDOW *win, const char *str, int n);
int mvaddstr(int y, int x, const char *str);
int mvaddnstr(int y, int x, const char *str, int n);
int mvwaddstr(WINDOW *win, int y, int x, const char *str);
int mvwaddnstr(WINDOW *win, int y, int x, const char *str, int n);
```

20.5 Função de Formatação de texto

A função **attron** permite formatar texto através de características como: itálico, negrito ou sublinhado.

Forma de usar:

```
attron(A_BOLD);
 attron(A_BLINK);
 printw("C for Linux, by Luzzardi, 2006");
 attron(A_BLINK);
attroff(A_BOLD);
```

Outra forma de formatação:

```
addch('A' | A_BOLD | A_UNDERLINE);
```

Formatação	Efeito
A_NORMAL	Nenhuma formatação
A_STANDOUT	Melhor modo de iluminação
A_UNDERLINE	Sublinhado
A_REVERSE	Fundo preto, texto branco (inverso do normal)
A_BLINK	Piscante
A_DIM	Brilho parcial
A_BOLD	Negrito
A_PROTECT	Modo protegido
A_INVIS	Modo invisível
A_ALTCHARSET	Alterna os caracteres
A_CHARTEXT	Máscara de bit para extrair um caracter

21. Outros Comandos

21.1 Comandos do Pré-Processador

Comandos que são executados antes do programa fonte ser compilado.

Inclusão de Arquivos de Header (cabeçalhos de funções):

```
#include <header.h>
     ou
#include "header_usuário.h"
```

Definição de Macro-Substituição:

```
#define <identificador> <valor>

#define true  !0
#define false  0
    ou
#define TRUE  !0
#define FALSE  0
```

Condições para Compilação:

```
#if <condição>

#else

#endif
```

ou

```
#ifdef <macro>

#else

#endif
```

Exemplo:

```
#include <stdio.h>

#define EXISTE_DEFINE 100     // existe ou não
#define SELECT  1     // selecione antes de compilar

#if (SELECT == 1)
      #define c 10
      #define l 10
#else
      #define c 20
      #define l 20
#endif

#ifdef EXISTE_DEFINE
     #define x 100
#else
     #define x 200
#endif
```

```
int main(void)
{
 printf("c = %d\n",c);
 printf("l = %d\n",l);
 printf("x = %d\n",x);
}
```

Resultado do Programa quando:

```
#define EXISTE_DEFINE 100     // existe ou não
#define SELECT  1     // selecione antes de compilar

c = 10
l = 10
x = 100
```

Resultado do Programa quando:

```
// #define EXISTE_DEFINE 100  // não existe esta linha
#define SELECT  2

c = 20
l = 20
x = 200
```

21.2 Hierarquia dos operadores aritméticos, relacionais, lógicos e bit a bit

Operador	*Operação*	*Forma de Avaliação*
()	*parênteses*	*esquerda para direita*
[]	*conchetes*	*esquerda para direita*
++	*incremento*	*direita para esquerda*
--	*decremento*	*direita para esquerda*
(tipo)	*cast*	*direita para esquerda*
*	*conteúdo do ponteiro*	*direita para esquerda*
&	*endereço de memória*	*direita para esquerda*
-	*unário (-7)*	*direita para esquerda*
~	*complemento de um*	*direita para esquerda*
!	*não*	*direita para esquerda*
*	*multiplicação*	*esquerda para direita*
/	*divisão*	*esquerda para direita*
%	*resto inteiro da divisão*	*esquerda para direita*
+	*adição*	*esquerda para direita*
-	*subtração*	*esquerda para direita*
>>	*deslocamento para direita*	*esquerda para direita*
<<	*deslocamento para esquerda*	*esquerda para direita*

Operador	*Operação*	*Forma de Avaliação*
>	*maior que*	*esquerda para direita*
>=	*maior ou igual*	*esquerda para direita*
<=	*menor ou igual*	*esquerda para direita*
<	*menor*	*esquerda para direita*
`==	*igual*	*esquerda para direita*
!=	*diferente*	*esquerda para direita*
&	*and bit a bit*	*esquerda para direita*
^	*ou exclusivo bit a bit*	*esquerda para direita*
|	*ou inclusive bit a bit*	*esquerda para direita*
&&	*e (and)*	*esquerda para direita*
||	*ou (or)*	*esquerda para direita*
`=	*atribuição*	*direita para esquerda*
+=	*atribuição (adição)*	*direita para esquerda*
-=	*atribuição (subtração)*	*direita para esquerda*
*=	*atribuição (multiplicação)*	*direita para esquerda*
/=	*atribuição (divisão)*	*direita para esquerda*
%=	*atribuição (resto inteiro divisão)*	*direita para esquerda*
>>=	*atribuição (deslocamento direita)*	*direita para esquerda*
<<=	*atribuição (deslocamento esquerda)*	*direita para esquerda*
&=	*atribuição (and)*	*direita para esquerda*
^=	*atribuição (ou exclusivo)*	*direita para esquerda*
|=	*atribuição (ou inclusive)*	*direita para esquerda*

21.3 Declaração de constantes tipadas

A linguagem C, padrão ANSI, permite criar constantes tipadas.

```
int main(void)
{
 const double pi = 3.1416;
 const int max = 7;
 float x[max];
```

21.4 Ponteiro para Ponteiro

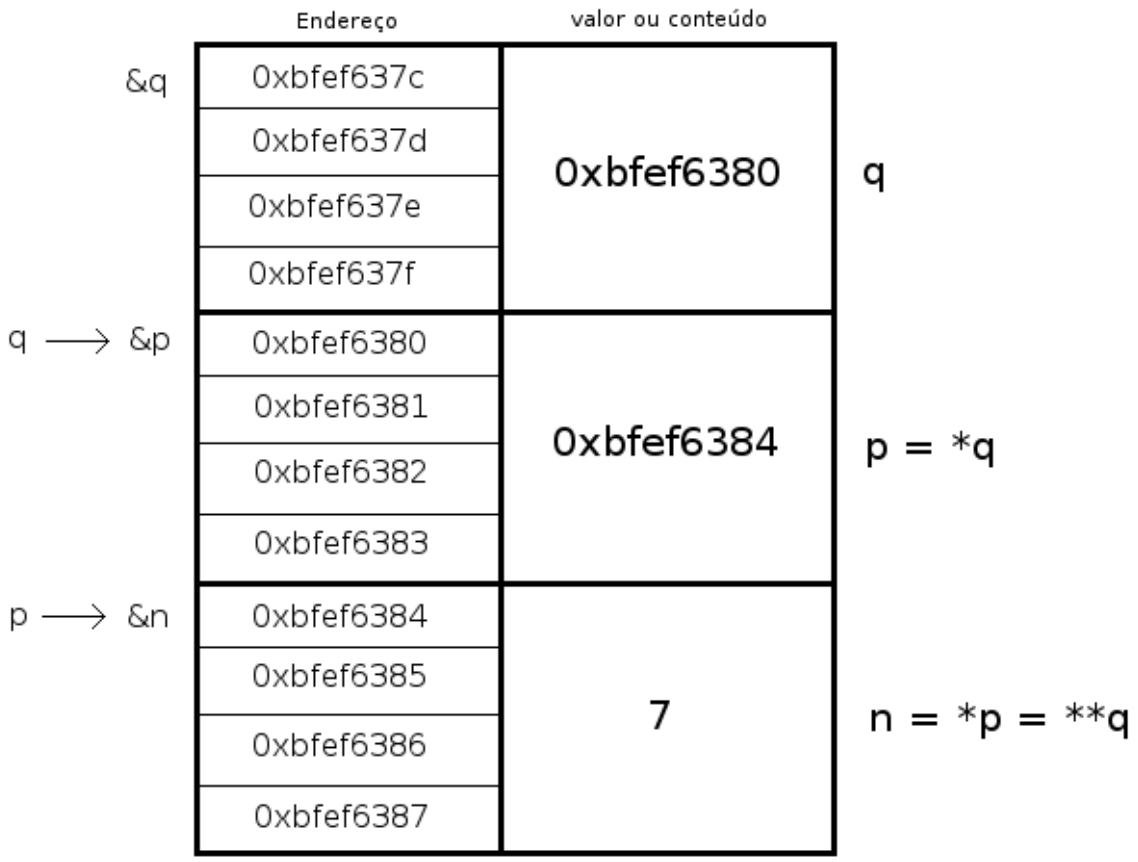
Um ponteiro para um ponteiro é uma forma de dupla indireçao. Um ponteiro comum (int *p), seu valor é o endereço de uma variável. Quando é definido um ponteiro para ponteiro, o primeiro ponteiro contém o endereço do segundo, que também aponta para a variável do primeiro ponteiro.

```
// pointer.c
```

```
# include <stdio.h>

int main(void)
{
 int n = 7;
 int *p,**q;

 p = &n;
 q = &p;
 printf("Valor de (n) = %d\n",n);
 printf("Endereço de (&n) = %p\n\n",&n);

 printf("Ponteiro (p) = %p\n",p);
 printf("Conteudo do Ponteiro (*p) = %d\n",*p);
 printf("Endereco do Ponteiro (&p) = %p\n\n",&p);

 printf("Ponteiro (q) = %p\n",q);
 printf("Conteudo do Ponteiro (*q) = %p\n",*q);
 printf("Conteudo do Ponteiro (**q) = %d\n",**q);
 printf("Endereco do Ponteiro (&q) = %p\n\n",&q);
}
```

Resultado do Programa:

```
Valor de (n) = 7
Endereço de (&n) = 0xbfef6384

Ponteiro (p) = 0xbfef6384
Conteudo do Ponteiro (*p) = 7
Endereco do Ponteiro (&p) = 0xbfef6380

Ponteiro (q) = 0xbfef6380
Conteudo do Ponteiro (*q) = 0xbfef6384
Conteudo do Ponteiro (**q) = 7
Endereco do Ponteiro (&q) = 0xbfef637c
```


Endereço
valor ou conteúdo
&q
0xbfef637c
0xbfef637d
0xbfef637e
0xbfef637f
0xbfef6380
q
q → &p
0xbfef6380
0xbfef6381
0xbfef6382
0xbfef6383
0xbfef6384
p = *q
p → &n
0xbfef6384
0xbfef6385
0xbfef6386
0xbfef6387
7
n = *p = **q

22. Programas escritos em C

O programa abaixo, **calc.c,** simula uma calculadora com quatro operações:
[+] Adição
[-] Subtração
[*] Multiplicação
[/] Divisão

```
// calc.c

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

int main(void)
{
 float x,y,resp;
 char op;
 int erro = 0;

 printf("Digite um valor: ");
 scanf("%f",&x);
 getchar();
 printf("Operador [+-*/]: ");
 op = getchar();
 printf("Digite outro valor: ");
 scanf("%f",&y);
 if (op == '+')
     resp = x + y;
 else
     if (op == '-')
         resp = x - y;
     else
         if (op == '*')
             resp = x * y;
         else
             if (op == '/')
                if (y != 0)
                    resp = x / y;
                else
                    {
                     printf("ERRO: Divisão por Zero\n");
                     return(1);
                    }
             else
                 {
                  printf("ERRO: Operador Inválido\n");
                  return(2);
                 }
 printf("Resposta: %.2f\n",resp);
 return(0);
}
```

O programa abaixo, **calculadora.c,** simula uma calculadora com 10 operações:
[+] Adição
[-] Subtração

```
[*] Multiplicação
[/] Divisão
[R] Raiz Quadrada
[P] Potência
[I] Inverso
[S] Seno
[C] Cosseno
[T] Tangente
```

```
// calculadora.c

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

#define PI 4*atan(1)

int main(void)
{
 float x, y, resp, rads;
 char op;
 int erro = 0;

 printf("Digite um valor: ");
 scanf("%f",&x);
 getchar();         // apaga enter do buffer de teclado
 printf("Operadores:\n");
 printf("[+] Adição\n");
 printf("[-] Subtração\n");
 printf("[*] Multiplicação\n");
 printf("[/] Divisão\n");
 printf("[R] Raiz Quadrada\n");
 printf("[P] Potência\n");
 printf("[I] Inverso\n");
 printf("[S] Seno\n");
 printf("[C] Cosseno\n");
 printf("[T] Tangente\n");
 printf("Qual a sua Opção? ");
 op = getchar();
 if (strchr("+-*/Pp",op))
    {
     printf("Digite outro valor: ");
     scanf("%f",&y);
    }
 if (op == '+')
     resp = x + y;
 else
     if (op == '-')
          resp = x - y;
     else
          if (op == '*')
              resp = x * y;
          else
              if (op == '/')
                 if (y != 0)
                      resp = x / y;
                  else
                      erro = 1;
               else
                   if (op == 'R' || op == 'r')
                        if (x >= 0)
                             resp = sqrt(x);
                        else
                             erro = 2;
```

```
                    else
                        if (op == 'P' || op == 'p')
                            resp = pow(x,y);
                         else
                            if (op == 'I' || op == 'i')
                                if (x != 0)
                                    resp = 1 / x;
                                 else
                                    erro = 1;
                            else
                                if (op == 'S' || op == 's')
                                    {
                                     rads = x * PI / 180.0;
                                     resp = sin(rads);
                                    }
                                 else
                                    if (op == 'C' || op == 'c')
                                       {
                                        rads = x * PI / 180.0;
                                        resp = cos(rads);
                                       }
                                    else
                                        if (op == 'T' || op == 't')
                                            if (x == 90.0 || x == 270.0)
                                                erro = 3;
                                            else
                                                {
                                                 rads = x * PI / 180.0;
                                                 resp = tan(rads);
                                                }
                                         else
                                            {
                                             printf("ERRO: Operador Inválido\n");
                                             return(1);
                                            }
 if (erro == 0)
     printf("Resposta: %.4f\n",resp);
 else
     if (erro == 1)
          printf("ERRO: Divisão por Zero\n");
     else
          if (erro == 2)
              printf("ERRO: Raiz Negativa\n");
          else
              if (erro == 3)
                 printf("ERRO: Tangente Infinita\n");
 return(0);
}
```

O programa abaixo, **juliano.c,** exibe o dia da semana (segunda, terça, quarta, quinta, sexta, sábado ou domingo) de uma determinada data. Para tanto, é utilizado o **Cálculo do Dia Juliano**.

Como calcular: http://www.rio.rj.gov.br/planetario/cent_pesq_calc.htm

```
// juliano.c

// --------------------------- Prototypes

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

// --------------------------- Programa Principal
```

```
int main(void)
{
 int dia, mes, ano, resto;
 int A, B, C;
 long int D, E, dj;
 float resp;

 printf("Dia [1..31]: ");
 scanf("%d",&dia);
 if (dia < 1 || dia > 31)
     printf("ERRO FATAL: Dia Inválido\n");
 else
     {
      printf("Mes [1..12]: ");
      scanf("%d",&mes);
      if (mes < 1 || mes > 31)
          printf("ERRO FATAL: Mês Inválido\n");
      else
          {
           printf("Ano [1583]: ");
           scanf("%d",&ano);
           if (ano < 1583)
               printf("ERRO FATAL: Ano Inválido\n");
           else
               {
                if (mes < 3)
                   {
                    ano = ano - 1;
                    mes = mes + 12;
                   }
                A = ano / 100;
                B = A / 4;
                C = 2 - A + B;
                resp = 365.25 * (ano + 4716);
                D = (long int) resp;
                resp = 30.6001 * (mes + 1);
                E = (long int) resp;
                dj = D + E + dia + C - 1524;
                resto = dj % 7;
                switch (resto)
                {
                  case 0: printf("Segunda-Feira\n");
                          break;
                  case 1: printf("Terça-Feira\n");
                          break;
                  case 2: printf("Quarta-Feira\n");
                          break;
                  case 3: printf("Quinta-Feira\n");
                          break;
                  case 4: printf("Sexta-Feira\n");
                          break;
                  case 5: printf("Sábado\n");
                          break;
                  case 6: printf("Domingo\n");
                          break;
                }
              }
            }
      }
```

```
}
```

O programa abaixo, **cpf.c**, permite verificar se um determinado CPF é válido ou inválido, calculando e exibindo os dígitos verificadores do CPF digitado.

Como calcular: http://www.geocities.com/CapeCanaveral/4274/cgcancpf.htm

```
// cpf.c

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

// ------------------------------- Programa Principal

int main(void)
{
 char cpf[15];
 int i, t, n, final;
 int digit1, digit2;
 int valor = 0, fator;

 printf("Cpf [99999999999]: ");
 scanf("%s",cpf);
 n = strlen(cpf);
 if (n != 11 && n != 9)
    {
     printf("Formato: 9 ou 11 d¡gitos numéricos\n");
     return;
    }
 if (n == 10)
    {
     n = n - 1;
     cpf[n] = (char) NULL;
    }
 for (i = 0;i < 9;i++)
     valor = valor + (cpf[i] - 48) * (10 - i);
 fator = valor / 11;
 fator = fator * 11;
 valor = valor - fator;
 if (valor < 2)
     digit1 = 0;
 else
     digit1 = 11 – valor;
 valor = 0;
 for (i = 0;i < 9;i++)
     valor = valor + (cpf[i] - 48) * (11 - i);
 valor = valor + digit1 * 2;
 fator = valor / 11;
 fator = fator * 11;
 valor = valor - fator;
 if (valor < 2)
     digit2 = 0;
 else
     digit2 = 11 – valor;
 final = digit1 * 10 + digit2;
 printf("Dígito Verificador: ");
 if (final >= 10)
```

```
     printf("%d\n",final);
 else
     printf("%02d\n",final);
 if (n == 11)
    {
     printf("Cpf Digitado: ");
     for (i = 0;i <= 7;i++)
         {
          printf("%c",cpf[i]);
          t = t + 1;
          if (t == 3)
             {
              t = 0;
              printf(".");
             }
         }
     printf("%c-",cpf[8]);
     for (i = 9;i <= 10;i++)
         printf("%c",cpf[i]);
     printf("\n");
    }
 if (digit1 == cpf[9] - 48 && digit2 == cpf[10] - 48)
     printf("Status: CPF Válido\n");
 else
     printf("Status: CPF Inválido\n");
}
```

Os programas abaixo, **conio.c** e **conio.h**, mostram uma forma alternativa de utilizar algumas funções do arquivo de header conio.h: **clrscr**, **gotoxy**, **getche**, **getch**, **kbhit**, **textcolor** e **textbackcolor.**

clrscr: limpa a tela em modo texto.
gotoxy: posiciona o cursor na coluna, linha.
getche: lê e mostra um caracter na tela sem enter.
getch: lê e não mostra um caracter na tela sem enter.
kbhit: verifique se a tecla enter é presionada ou não.
textcolor: altera a cor do texto.
textbackcolor: altera a cor de fundo.

```
// conio.c

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

#define ESC 27

int main(void)
{
 int tecla;

 clrscr();
 do {
     textcolor(BLUE);
     gotoxy(10,5);
     printf("Tecla: ");
```

```
     tecla = getche();        // troque por tecla = getch();
     textbackcolor(GREEN);
     gotoxy(10,7);
     printf("Código: %d -> Caracter: %c\n", tecla, tecla);
    } while (tecla != ESC);
}

// conio.h

#include <termios.h>
#include <unistd.h>
#include <sys/time.h>

#define BLACK    0
#define RED      1
#define GREEN    2
#define BROWN    3
#define BLUE     4
#define MAGENTA  5
#define CYAN     6
#define DARKGRAY 7
#define SUBLINHA 8

#define black    0
#define red      1
#define green    2
#define brown    3
#define blue     4
#define magenta  5
#define cyan     6
#define darkgray 7
#define sublinha 8

// ---------------------------- função: clrscr

void clrscr(void)
{
 printf("\x1B[2J");
}
// ---------------------------- função: gotoxy

void gotoxy(int c, int l)
{
 printf("\033[%d;%df",l, c);
}

// --------------------------- função: getch

int getch(void)
{
 struct termios oldt, newt;
 int ch;

 tcgetattr(STDIN_FILENO, &oldt);
 newt = oldt;
 newt.c_lflag &= ~(ICANON | ECHO);
 tcsetattr(STDIN_FILENO, TCSANOW, &newt);
 ch = getchar();
 tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
 return(ch);
```

```
}

// --------------------------- função: getche

int getche(void)
{
 struct termios oldt, newt;
 int ch;

 tcgetattr(STDIN_FILENO, &oldt);
 newt = oldt;
 newt.c_lflag &= ~(ICANON | ECHO);
 tcsetattr(STDIN_FILENO, TCSANOW, &newt);
 ch = getchar();
 tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
 putchar(ch);
 return(ch);
}

// --------------------------- função: kbhit

int kbhit(void)
{
 struct timeval tv;
 fd_set read_fd;

 tv.tv_sec=0;
 tv.tv_usec=0;
 FD_ZERO(&read_fd);
 FD_SET(0,&read_fd);
 if (select(1, &read_fd, NULL, NULL, &tv) == -1)
    return(0);
 if (FD_ISSET(0,&read_fd))
    return(1);
 return(0);
}

// --------------------------- função: textcolor

void textcolor(int cor)
{
 printf("\033[0m");
 switch (cor)
 {
   case BLACK: printf("\033[30m");
                 break;
     case RED: printf("\033[31m");
                 break;
   case GREEN: printf("\033[32m");
                 break;
   case BROWN: printf("\033[33m");
                 break;
    case BLUE: printf("\033[34m");
                 break;
 case MAGENTA: printf("\033[35m");
                 break;
    case CYAN: printf("\033[36m");
                 break;
case DARKGRAY: printf("\033[37m");
                 break;
```

```
case SUBLINHA: printf("\033[38m");
                 break;
  }
}

// --------------------------- função: textbackcolor

void textbackcolor(int cor)
{
 switch (cor)
 {
      case BLACK: printf("\033[40m");
                    break;
        case RED: printf("\033[41m");
                    break;
      case GREEN: printf("\033[42m");
                    break;
      case BROWN: printf("\033[43m");
                    break;
       case BLUE: printf("\033[44m");
                    break;
    case MAGENTA: printf("\033[45m");
                    break;
       case CYAN: printf("\033[46m");
                    break;
   case DARKGRAY: printf("\033[47m");
                    break;
  }
}
```

O programa abaixo, **cores.c** mostra as cores disponíveis nas funções ***textcolor*** e ***textbackcolor*** da biblioteca ***conio.h***.

```
// cores.c

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

int main(void)
{
 textcolor(BLACK);
 printf("Black\n");
 textcolor(RED);
 printf("Red\n");
 textcolor(GREEN);
 printf("Green\n");
 textcolor(BROWN);
 printf("Brown\n");
 textcolor(BLUE);
 printf("Blue\n");
 textcolor(MAGENTA);
 printf("Magenta\n");
 textcolor(CYAN);
 printf("Cyan\n");
 textcolor(DARKGRAY);
 printf("DarkGray\n");
 textcolor(SUBLINHA);
 printf("Sunlinha\n");
```

```
 textbackcolor(BLACK);
 printf("\nBlack");
 textbackcolor(RED);
 printf("\nRed");
 textbackcolor(GREEN);
 printf("\nGreen");
 textbackcolor(BROWN);
 printf("\nBrown");
 textbackcolor(BLUE);
 printf("\nBlue");
 textbackcolor(MAGENTA);
 printf("\nMagenta");
 textbackcolor(CYAN);
 printf("\nCyan");
 textbackcolor(DARKGRAY);
 printf("\nDarkGray");
 getchar();
}
```