

# LINGUAGEM C - NOTAS DE AULA

Jorge Surian

## Apresentação (Aula 01T)

Estas Notas de Aula, visam aumentar a produtividade dos alunos nas aulas teóricas, evitando a cópia das teorias expostas. Grande parte dos exemplos analisados em sala de aula e enunciados de exercícios constam nesta apostila, além da resolução de alguns destes. Os EC (exercícios de classe), serão normalmente baseados nos exercícios complementares de cada Aula a serem desenvolvidos preferencialmente no laboratório.

Habitualmente antes de resolvermos exemplos ou exercícios, elaboraremos o algoritmo, que nada mais é que uma seqüência de operações cuja execução produz um resultado que é a resposta de um problema proposto.

Um programa de computador nada mais é que a codificação de um algoritmo numa linguagem de programação. Linguagens como C, Pascal, BASIC, ALGOL, Clipper, COBOL, etc., são chamadas de procedurais, devido ao fato das instruções serem executadas de forma seqüencial, enquanto que as linguagens baseadas no conceito de eventos como C++, Visual BASIC, Visual Objects, utilizam outra estratégia de programação (Programação Orientada ao Objeto), a ser vista em outro módulo do curso (OOP), em C utilizaremos a **metodologia estruturada**.

Os algoritmos podem ser estruturados ou não, conforme exemplificamos a seguir no cálculo do máximo divisor comum entre dois números inteiros positivos, com operações elementares:

### Linear

```
Leia m,n
(1) se n = 0 então
  imprima m
  pare
  k <- m - Int(m / n) * n
  m <- n
  n <- k
  vá para (1)
```

### *Em Quick BASIC teríamos*

```
input m : input n
10 if n = 0 then
    print m
end if
k = m - Int(m / n) * n
m = n
n = k
goto 10
```

### *Em BASICA teríamos*

```
10 input m : input n
20 if n <> 0 then 50
30 imprima m
40 end
50 k = m - Int(m / n) * n
60 m = n
70 n = k
80 goto 20
```

O Algoritmo de Euclides anteriormente apresentado , apesar de muito simples, executado normalmente por qualquer criança de primeiro grau, ganha contornos de aparente dificuldade quando transcrito para o GW-BASIC, um dialeto BASIC, que exige numeração de linhas, que nos obrigou a alterar um pouco a estratégia de resolução do problema. Já a versão em Quick BASIC, poderia ser transcrita por qualquer pessoa com um mínimo de conhecimento em algoritmos e BASIC.

### **Estruturado**

```
Inteiros m,n
Leia m,n
enquanto n <> 0 faça
k <- m - Int(m / n) * n
m <- n
n <- k
imprima m
pare
```

### **Em C teríamos**

```
main()
{
    int m,n,k;
    scanf("%d",&m);
    scanf("%d",&n);
    while (n != 0) {
        k = m - m / n * n;
        m = n;
        n = k;
    }
    printf("%d",m);
}
```

### **Em Clipper teríamos**

```
input to a
input to b
do while n <> 0
    k = m - Int(m / n) * n
    m = n
    n = k
enddo
? m
```

Nas Linguagens estruturadas a representação fica idêntica, quer em C quer em Clipper. Ficaria a mesma forma em Quick BASIC, Pascal ou COBOL Estruturado, daí a preferência na abordagem dita estruturada. No decorrer deste curso nos deteremos detalhadamente no estudo dos algoritmos estruturados e analisaremos alguns (poucos, porém importantes) casos onde a programação linear é mais adequada que a estruturada.

## Simulação

Por melhor que seja o conhecimento do Analista/Programador, este só poderá ter certeza que sua “estratégia” foi bem sucedida após testar o programa num computador.

Bons programadores tem poucas surpresas quando testam seus programas, fazendo normalmente umas poucas correções para que o programa funcione de maneira adequada.

Programadores iniciantes criam uma estratégia (algoritmo) muitas vezes ineficiente e normalmente “correm” para o computador visando testar o programa e acabam por perder um tempo enorme. Não é raro ouvirmos de iniciantes a frase “Nunca conseguirei fazer um programa ...”. Certamente, não conseguirá mesmo, caso não tenha uma estratégia definida.

Imagine uma “tarefa” muito simples, como por exemplo fritar um ovo. Caso você não soubesse “operar” o fogão provavelmente não conseguiria acendê-lo. Se nunca tivesse visto alguém quebrar um ovo, provavelmente não conseguiria fazê-lo sem perder parte de seu conteúdo. A fritura seria algo desastroso, caso você desconhecesse a utilidade do óleo. E o sabor seria frustrante, caso o tempero utilizado fosse açúcar.

O Programador iniciante que não simula seu algoritmo, se compara ao cozinheiro desastrado descrito acima. E como simular?

Basta “agir” como se fossemos o próprio computador, ou seja devemos “fingir” que nosso raciocínio é baseado no **conteúdo de variáveis**. De fato, usualmente antes de tomarmos alguma decisão analisamos uma série de fatores (as tais variáveis dos programas). Desta forma, caso fizermos uma análise apurada sobre os conteúdos das variáveis, poderemos “descobrir” a função de cada programa.

Primeiramente vamos resolver o problema como aprendemos no primeiro grau.

Supondo  $M = 120$  e  $N = 28$ .

|           |   |     |    |   |       |
|-----------|---|-----|----|---|-------|
| Dividendo | = | 120 | 28 | 8 |       |
| Divisor   | = | 28  | 8  | 4 | → MDC |
| Quociente | = | 4   | 3  | 2 |       |
| Resto     | = | 8   | 4  | 0 |       |

Simulação:

| M   | N  | K | Impresso | Instruções                              | Comentários                           |
|-----|----|---|----------|---|---------------------------------------|
| 120 | 28 |   |          | Declarações e Leituras                  |                                       |
|     |    | 8 |          | Enquanto $n < 0$<br>$k = m - m / n * n$ | Cálculo do Resto                      |
| 28  | 8  |   |          | $m = n; n = k$                          | Novos Divisor e Dividendos            |
|     |    | 4 |          | Enquanto $n < 0$<br>$k = m - m / n * n$ | Voltando ao Teste<br>Cálculo do Resto |
| 4   | 0  |   |          | $m = n; n = k$                          | Novos Divisor e Dividendos            |
|     |    |   |          | Enquanto $n < 0$                        | N é Zero                              |
|     |    |   | 4        |   | Achou MDC!                            |

**Observação Final:** Obviamente caso você se sinta seguro a resolver diretamente o problema na linguagem diretamente no computador, faça isto. Porém se surgirem dificuldades, não se esqueça de fazer o algoritmo e a simulação. Lembre-se do cozinheiro desastrado, que se tivesse consultado uma simples receita (todo algoritmo não passa de uma receita sofisticada), poderia ao menos fritar o ovo!

## Exercícios (Aula 01L)

Nunca esqueça que os comandos da linguagem C devem ser escritos SEMPRE em letras minúsculas!

Apresentação do Interpretador Classic C.

- 1- Elabore programa que apresente mensagem alô mundo!
- 2- Elabore programa para o cálculo do máximo divisor comum entre 2 números.
- 3- Elabore programa que imprima o seu nome.

## **Linguagem C (Aula 02T)**

### **Objetivos:**

- Tornar o Aluno apto a programar em C (pequenos programas).
- Conhecimento do C Clássico (Classic C).
- Conhecimento do Turbo C (compilador voltado ao aprendizado e também usado comercialmente).

### **Estrutura do Curso:**

- Aulas Teóricas: Exposição, Exemplos e Exercícios.
- Aulas Práticas: Processar Programas Prontos e Elaboração de Exercícios.

### **Introdução**

Desenvolvida nos laboratórios Bell na década de 70, a partir da Linguagem B (criada no final dos anos 60 por Ken Thompson), que foi reformulada por Brian Kernighan e Dennis M. Ritchie e posteriormente renomeada para C.

Podendo ser considerada como uma linguagem de médio nível, pois possui instruções que a tornam ora uma linguagem de alto nível e estruturada como o Pascal, se assim se fizer necessário, ora uma linguagem de baixo nível pois possui instruções tão próximas da máquina, que só o Assembler possui.

De fato com a linguagem C podemos construir programas organizados e concisos (como o Pascal), ocupando pouco espaço de memória com alta velocidade de execução (como o Assembler). Infelizmente, dada toda a flexibilidade da linguagem, também poderemos escrever programas desorganizados e difíceis de serem compreendidos (como usualmente são os programas em BASIC).

Devemos lembrar que a linguagem C foi desenvolvida a partir da necessidade de se escrever programas que utilizassem recursos próprios da linguagem de máquina de uma forma mais simples e portátil que o assembler.

Uma análise superficial dos programas escritos em C e Clipper (Aula 01 - Algoritmos Estruturados), nos permite perceber que a linguagem C supera em muito em dificuldade o programa análogo em Clipper. Ora, então porque não desenvolvermos programas somente em Clipper?

A inúmeras razões para a escolha da linguagem C como a predileta para os desenvolvedores “profissionais”. As características da Linguagem C servirão para mostrar o porquê de sua ampla utilização.

### **Características da Linguagem C**

- Portabilidade entre máquinas e sistemas operacionais.
- Dados compostos em forma estruturada.
- Programas Estruturados.
- Total interação com o Sistema Operacional.
- Código compacto e rápido, quando comparado ao código de outras linguagem de complexidade análoga.

### **Aplicações Escritas em C**

Atualmente, nos USA, C é a linguagem mais utilizada pelos programadores, por permitir, dadas suas características, a escrita de programas típicos do Assembler, BASIC, COBOL e Clipper, sempre com maior eficiência e portabilidade, como podemos constatar pelos exemplos abaixo relacionados:

- Sistema Operacional: UNIX (Sistema Operacional executável em micro computadores e em mainframes).
- Montadores: Clipper (O utilitário de banco de dados mais usado no Brasil).
- Planilhas: 1,2,3 e Excel (A planilha eletrônica com maior volume de vendas mundial).
- Banco de Dados: dBase III, IV e Access (o gerenciador de base de dados mais utilizado no mundo).
- InfoStar: O Editor de Texto mais utilizado nos USA no Sistema Operacional UNIX.
- Utilitários: FormTool (Editor de formulário mais vendido no mundo).
- Aplicações Gráficas: Efeitos Especiais de filmes com Star Trek e Star War.
- Linguagens como o Power Builder e o Visual Basic, respectivamente as linguagens mais utilizadas nos EUA e no Brasil.

No Brasil utilizada por empresas especializadas na elaboração de vinhetas e outros efeitos especiais.



Quadro de características de linguagens:

| Linguagens<br>Características Ideais | Assembler     | BASIC<br>Pascal | Clipper       | COBOL   | C          |
|--------------------------------------|---------------|-----------------|---------------|---------|------------|
| Executáveis Curtos                   | ótimo ☺       | fraco ☹         | péssimo<br>☹☹ | fraco ☹ | ótimo<br>☺ |
| Executáveis Rápidos                  | ótimo ☺       | bom             | razoável      | fraco ☹ | bom        |
| Portáteis                            | péssimo<br>☹☹ | bom             | ótimo ☺       | ótimo ☺ | bom        |
| Manipulação de Bits                  | ótimo ☺       | razoável        | péssimo<br>☹☹ | fraco ☹ | ótimo<br>☺ |

O quadro anterior, deixa claro o porquê da revolução causada pela Linguagem C, dados os inúmeros pontos fortes da linguagem e a inexistência de pontos fracos da mesma. Não devemos concluir apressadamente que poderemos desenvolver tudo em C e abandonarmos todas as outras linguagens, pelos seguintes motivos:

- Alta base de programas escritos em Assembler, COBOL, BASIC, Pascal.
- Conhecimento amplo de linguagens concorrentes como COBOL, Clipper e BASIC.
- Melhor adaptação de alguma linguagem para tarefa específica como Gráficos (Pascal), Inteligência Artificial (Prolog, LISP), matemáticas (Pascal e FORTRAN), aplicações comerciais (COBOL, Clipper e BASIC).
- As versões atuais das linguagens BASIC, C, Clipper, Pascal e COBOL, estão muito mais semelhantes do que o eram ao tempo em que este quadro foi elaborado, portanto na prática muitas vezes este quadro, meramente teórico, pode tornar-se inaplicável.

A médio prazo, podemos afirmar que a linguagem C deverá ir desalojando as outras linguagens podendo até mesmo tornar-se um padrão de direito, porém devemos lembrar que algumas linguagens foram propostas para isto (Algol, PL/1) e não só não conseguiram atingir seus objetivos como praticamente desapareceram.

A tabela anterior não esgota todas as possibilidades, pois alguns dialetos de linguagem destacam recursos que na maior parte das implementações da linguagem não são muito fortes. O inverso também pode ocorrer, de modo que as vezes uma implementação corrige algum defeito da linguagem (por exemplo aumenta drasticamente seu desempenho) e piora outras (portabilidade inexistente). Este problema afeta sensivelmente o BASIC e as implementações xBase (Fox, dBase e Clipper) e em menor grau o Pascal e a própria linguagem C.

**Nota de Revisão:** Apesar da Linguagem C ter crescido de cerca de 3% das instalações no Brasil, quando da escrita da primeira versão desta apostila (1989), para cerca de 20% das instalações atualmente (1997), o número atual esconde uma realidade que deve ser mencionada. Muitas das instalações usuárias da linguagem C, também são usuárias do VB (Visual Basic) da Microsoft. Desta forma, a linguagem C está presente normalmente para construção de rotinas mais sofisticadas impossíveis de implementação em VB. Contrariamente as empresas usuárias do Delphi (Borland) e do Power Builder (Sybase), geralmente estão abrindo mão em suas instalações da Linguagem C, tendo em vista a capacidade destas implementações em realizar tarefas geralmente destinadas a linguagem C. A linguagem Java, que é claramente baseada em C, também começa a penetrar em áreas onde C reinava abosulta.

**Exemplo:** Desejo desenvolver um processador de textos, a partir do início (não possui qualquer código de programa pronto). Qual linguagem escolherei?

*Solução:*

| Características Ideais | Classe      | Valor |
|------------------------|-------------|-------|
| Executáveis Curtos     | Não Importa | 0     |
| Executáveis Rápidos    | Fundamental | 3     |
| Portáteis              | Desejável   | 1     |
| Simplicidade           | Desejável   | 1     |
| Manipulação de Bits    | Necessário  | 2     |

**Onde 0- Não Importa, 1- Desejável, 2- Necessário, 3- Fundamental**

Juntando as Características do Projeto as Características da Linguagem, temos:

| Linguagens<br>Características Ideais | Pt | Assembler | Basic<br>Pascal | Clipper | COBOL | C      |
|--------------------------------------|----|-----------|-----------------|---------|-------|--------|
| Executáveis Curtos                   | 0  | 7x0=0     | 1x0=0           | 0x0=0   | 0*0=0 | 7*0=0  |
| Executáveis Rápidos                  | 3  | 7*3=21    | 5x3=15          | 3x3=9   | 1x3=3 | 5x3=15 |
| Portáteis                            | 1  | 0*1=0     | 3*1=3           | 1*1=1   | 5*1=5 | 7*1=7  |
| Simplicidade                         | 1  | 0*1=0     | 5*1=5           | 7*1=7   | 7*1=7 | 5*1=5  |
| Manipulação de Bits                  | 2  | 7*2=14    | 3*2=5           | 1*2=2   | 0*2=0 | 7*2=14 |
|                                      |    | → 35 ←    | 27              | 19      | 15    | 41     |

Onde 0- Péssimo, 1- Fraco, 3- Razoável, 5- Bom e 7- Ótimo

Resposta: Linguagens Adequadas C ou Assembler. Caso se dê prioridade a portabilidade C é ainda mais adequada. O número talvez surpreendentemente alto para Pascal e BASIC serve para demonstrar o porquê da existência de Editores feitos total ou parcialmente em Pascal. O fraco desempenho do Clipper e do COBOL era esperado pois são linguagens mais apropriadas para manipulação de dados. Na prática devemos observar a existência de dois critérios também importantes, que são a existência de programadores especialistas na linguagem adequada e a possibilidade de junção de mais de uma linguagem numa implantação qualquer.

### **Exercícios**

- 1- Desejo criar um vírus de computador, qual a linguagem ideal?
- 2- Desejo criar um utilitário de Banco de Dados, semelhante a dBase. Qual a linguagem ideal?
- 3- Supondo ter uma empresa concluído ser ideal desenvolver internamente sua folha de pagamento. O Gerente de Informática, atento ao movimento do Mercado definiu a escrita dos programas em C, visando poder migrar a aplicação entre ambientes. Supondo ainda que a equipe desconhece totalmente a Linguagem C, mas que é bastante gabaritada em COBOL, em sua opinião nosso Gerente foi feliz em sua decisão? Justifique.

### **Exercícios (Aula 02L)**

- 1- Leia seu nome e o imprima.
- 2- Leia dois números e apresente seu produto.
- 3- Leia três números e apresente sua média.

### **C- Uma Visão Geral - Instruções de Entrada e Saída (Aula 03T)**

Toda linguagem de programação de alto nível suporta o conceito de “Tipo de Dado”, que define um conjunto de valores que a variável pode armazenar, e os tipos mais comuns encontrados nas linguagens de programação, ou seja, inteiro, real e caractere. Diferentemente do Pascal que é fortemente tipada onde a mistura entre um número inteiro e um real podem causar erros, C suporta livremente tipos caracteres e inteiros na maioria das expressões!

Em geral os compiladores C realizam pouca verificação de erros em tempo de execução, verificação de limites de matrizes ou compatibilidade entre tipos de argumentos, cabendo esta responsabilidade ao programador. Assim você decide onde uma verificação de erro é ou não mais necessário.

Por ser capaz de manipular bits, bytes e endereços, C se adapta bem a programação a nível

de sistema. E tudo isto é realizado por apenas 43 palavras reservadas no Turbo C, 32 nos compiladores padrão ANSI e 28 no C Padrão. Como curiosidade, o IBM BASIC que é um interpretador BASIC com fins puramente educativos tem 159 comandos.

Como curiosidade apresentamos a seguir quadro com as palavras reservadas do C Padrão.

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

## Fundamentos de C

Em se tratando de programação a expressão “Se você não souber, jamais irá aprender.” é uma verdade absoluta. É muito comum ouvirmos queixas do tipo “nunca conseguirei escrever um programa”, ou “só sendo louco ou gênio para descobrir a solução”. Estas expressões geralmente são ditas por estudantes que desconhecem o fato de que cada elemento da linguagem (comandos, funções) não existe sozinho, mas somente combinados a outros elementos.

Desta forma a orientação que adotaremos neste início do curso se deterá mais na compreensão geral do programa, do que a análise detalhada de cada comando ou função utilizada. De fato apresentaremos alguns comandos fundamentais para a escrita de programas básicos e apenas nos utilizaremos de sua sintaxe mais elementar (posteriormente estudaremos cada um deles mais detidamente), construiremos os primeiros programas do curso.

Exemplo 1: Programa mostra a idade.

```
/* Exemplo Idade */
main()
{
int idade;
idade = 40;
printf("Sua idade e' %d anos. \n", idade);
}
```

Este programa simplesmente imprime “Sua idade e’ 40 anos.” saltando uma linha (/n) em seu término.

## Comandos Básicos - 1ª. Parte

### Instruções de Entrada e Saída

O objetivo de escrevermos programas é em última análise, a obtenção de resultados (Saídas) depois da elaboração de cálculos ou pesquisas (Processamento) através do fornecimento de um conjunto de dados ou informações conhecidas (Entradas).

Para que nosso programa possa receber dados e alocá-los em variáveis, que serão responsáveis por armazenar as informações iniciais, nossa linguagem deverá conter um conjunto de instruções que permitam ao operador interagir com o programa fornecendo os dados quando estes forem necessários.

#### **scanf()**

Uma das mais importantes e poderosas instruções, servirá basicamente para promover leitura de dados (tipados) via teclado.

Sua forma geral será: `scanf("string de controle", lista de argumentos);`

Posteriormente ao vermos sua sintaxe completa, abordaremos os recursos mais poderosos da <string de controle>, no momento bastará saber que:

`%c` - leitura de caracter

`%d` - leitura de números inteiros

`%f` - leitura de números reais

`%s` - leitura de caracteres

A lista de argumentos deve conter exatamente o mesmo número de argumentos quantos forem os códigos de formatação na <string de controle>. Se este não for o caso, diversos problemas poderão ocorrer - incluindo até mesmo a queda do sistema - quando estivermos utilizando programas compilados escritos em C. Felizmente ao utilizarmos o Classic C, apenas uma mensagem de erro será apresentada, para que possamos corrigir o programa sem outros inconvenientes.

Cada variável a ser lida, deverá ser precedida pelo caracter `&`, por razões que no momento não convém explicarmos, mas que serão esclarecidas no decorrer do curso. Para seqüência de caracteres (`%s`), o caracter `&` não deverá ser usado.

### Exemplo: Programa para ler e mostrar uma idade

```
/* Exemplo Le e Mostra Idade */
main()
{
int idade;
char nome[30];
printf("Digite sua Idade: ");
scanf("%d",&idade);
printf("Seu Nome: ");
scanf("%s",nome); /* Strings não utilizar '&' na leitura */
printf("%s Sua idade e' %d anos. \n", nome, idade);
}
```

### Laboratório (Aula 03L)

- 1- Leia o número e o nome dos elementos do grupo e os apresente.
- 2- Leia o nome e as notas de um aluno. Apresente seu nome e sua média.
- 3- Leia uma letra, um número inteiro, um número com casas decimais e uma string, depois os apresente.

### Instruções de Entrada e Saída (continuação) (Aula 04T)

O Comando printf, usado anteriormente, segue o mesmo padrão de scanf(), porém é destinado a apresentação dos dados, enquanto aquele destina-se a leitura dos dados.

#### printf()

É outro dos mais poderosos recursos da linguagem C, printf() servirá basicamente para a apresentação de dados no monitor.

Sua forma geral será: printf(“string de controle”, lista de argumentos);

Necessariamente você precisará ter tantos argumentos quantos forem os comandos de formatação na “string de controle”. Se isto não ocorrer, a tela exibirá sujeira ou não exibirá qualquer dado.

Os caracteres a serem utilizados pelo printf() em sua <string de controle>, no momento serão os mesmos de scanf().

Exemplo: Dado um número, calcule seu quadrado.

```
main()
{
int numero;
printf("Digite um Numero: ");
scanf("%d",&numero);
printf("O %d elevado ao quadrado resulta em %d. \n",
numero,numero*numero);
}
```

### ***Funções em C***

Conceitualmente, C é baseada em blocos de construção. Assim sendo, um programa em C nada mais é que um conjunto de funções básicas ordenadas pelo programador. As instruções printf() e scanf(), vistas anteriormente, não fazem parte do conjunto de palavras padrões da linguagem (instruções), pois não passam elas mesmas de funções escritas em C! Esta abordagem, permite a portabilidade da linguagem, pois seus comandos de entrada e saída, não são parte do conjunto básico da linguagem, livrando-a desta forma dos problemas de suporte aos diversos padrões de vídeos, teclados e sistemas operacionais existentes.

Cada função C é na verdade uma sub-rotina que contém um ou mais comandos em C e que executa uma ou mais tarefas. Em um programa bem escrito, cada função deve executar uma tarefa. Esta função deverá possuir um nome e a lista de argumentos que receberá. As funções em C são muito semelhantes as usadas no Pascal, com a diferença que o próprio programa principal é apenas uma função que se inicia com a palavra reservada main() podendo receber parâmetros diretamente do DOS, por exemplo.

Exemplo: Programa principal chamando função alo.

```
main()
{
alo();
}
alo()
{
printf("Alô!\n\n");
}
```

Retomemos o exemplo do cálculo de um número elevado ao quadrado.

**Exemplo: Quadrado com função**

```
main()
{
int num;
printf("Digite um numero: ");
scanf("%d", &num);
sqr(num); /* sqr recebe "num" do programa principal */
}
sqr()
int x; /* x é um "parâmetro" recebido do programa principal
no caso x "vale" o conteúdo de num */
{
printf("%d ao quadrado e' %d ", x, x*x);
}
```

**Nota:** O argumento simplesmente é o valor (em "num") digitado no programa principal (em scanf) e enviado a função sqr.

Um conceito importante e normalmente confundido é a diferença conceitual entre "argumento" e "parâmetro" que em resumo pode ser definido da seguinte forma: "Argumento" se refere ao *valor* que é usado para chamar uma função. O termo "Parâmetro" se refere à variável em uma função que recebe o valor dos argumentos usados na função. A distinção que deve ser compreendida é que a variável usada como argumento na chamada de uma função não tem nenhuma relação com o parâmetro formal que recebe o valor dessa variável.

**Exercício: Passagem de variáveis entre rotinas.**

```
int x;
main()
{
int a;
printf("Digite um valor: ");
scanf("%d", &a);
x = 2 * a + 3;
printf("%d e %d", x, soma(a));
}

soma(z)
int z;
{
x = 2 * x + z;
return(x);
}
```

## **Laboratório (Aula 04L)**

Retome os exercícios das Aulas 1L, 2L e 3L refazendo-os usando funções.

Utilize os caracteres abaixo para apresentação de forma mais sofisticada dos resultados.  
Utilize a função `cls()`, disponível no Classic C para apagar a tela.

Observemos o Quadro de Operadores Especiais suportados por `printf()`

| <b>Código</b>   | <b>Significado</b>                         |
|-----------------|--|
| <code>\b</code> | Retrocesso (BackSpace)                     |
| <code>\f</code> | Salto de Página (Form Feed)                |
| <code>\n</code> | Linha Nova (Line Feed)                     |
| <code>\r</code> | Retorno do Carro (cr)                      |
| <code>\t</code> | Tabulação Horizontal (TAB)                 |
| <code>\'</code> | Caracter com apóstrofo                     |
| <code>\0</code> | Caracter Nulo ou Fim de String (Seqüência) |
| <code>\x</code> | Representação de byte na base hexadecimal  |

Exemplo: `printf("\x41");` causa a impressão da letra A na tela.

## **Tomada de Decisão (Aula 05T)**

Comandos Básicos - 2ª. Parte

Análogo a outras linguagens, sua forma geral será

```
if <condição>  
    <comando>;  
else  
    <comando>;
```

Exemplo 1: Programa Adulto, Jovem ou Velho.

```
main()  
{  
  int i;  
  printf("Digite sua idade: ");  
  scanf("%d", &i);  
  if (i > 70)  
    printf("Esta Velho!");  
  else  
    if (i > 21)  
      printf("Adulto");  
    else  
      printf("Jovem");  
}
```

Observação: A expressão avaliada, deverá obrigatoriamente estar entre parênteses.

### Exemplo 2: Maior entre três números

```
main()
{
int a,b,c;
cls();
printf("Digite o 1º Número: ");
scanf("%d",&a);
printf("\nDigite o 2º Número: ");
scanf("%d",&b);
printf("\nDigite o 3º Número: ");
scanf("%d",&c);
if (a > b)
    if (a > c)
        printf("\nO Maior é %d",a);
    else
        printf("\nO Maior é %d",c);
else
    if (b > c)
        printf("\nO Maior é %d",b);
    else
        printf("\nO Maior é %d",c);
}
```

### Exemplo 3: Maior entre três números (Segunda Solução)

```
main()
{
int a,b,c,d;
cls();
printf("Digite o 1º Número: ");
scanf("%d",&a);
printf("\nDigite o 2º Número: ");
scanf("%d",&b);
printf("\nDigite o 3º Número: ");
scanf("%d",&c);
if (a > b)
    d = a;
else
    d = b;
if (c > d)
    printf("\nO Maior é %d",c);
else
    printf("\nO Maior é %d",d);
}
```

Exemplo 4: Dados 2 números apresente-os ordenados.

```
main()
{
int a,b,t;
cls();
printf("Digite o 1º Número: ");
scanf("%d",&a);
printf("\nDigite o 2º Número: ");
scanf("%d",&b);
if (a < b) {
    t = a;
    a = b;
    b = t;
}
printf("\nOrdenados: %d e %d ",b,a);
}
```

## Contagem

Um dos grandes benefícios dos sistemas de processamento de dados está em sua confiabilidade (precisão nos cálculos) e rapidez (infinitamente superior ao ser humano), desta forma é ideal para processamento de elevado número de operações **repetitivas**. O processamento de uma Folha de Pagamentos, a Emissão de Notas Fiscais, a Geração de Estatísticas de Faturamento, são típicas tarefas a serem realizadas por processamento eletrônico de dados.

Todas as linguagens importantes dispõe de recursos destinados a forçar a repetição de execução de um determinado número de instruções. Diferentemente do BASIC e do Pascal, o "loop" *for* é a instrução mais poderosa na criação de estruturas de repetição. Neste momento, abordaremos apenas sua sintaxe simplificada, que lembra bastante a sintaxe do FORTRAN (comando DO) e semelhante àquelas linguagens citadas anteriormente.

Sua forma mais simples é:

*for* (<início>;<condição>;<incremento>) comando;

Exemplo 1: Contagem de 1 a 100 ficaria

```
main()
{
int cont;
for (cont = 1; cont <= 100; cont++)
    printf("%d",cont);
}
```

**Exemplo 2:** Elaborar programa que imprima a tabuada de um número dado.

```
main()
{
int cont,num;
printf("Digite um Numero: "); scanf("%d",&num);
for (cont = 0; cont <= 10; cont++)
    printf("%2d * %2d = %2d \n",num,cont,num * cont);
}
```

**Nota:** O número ‘2’ antes do ‘d’ causa a representação em vídeo de 2 casas, permitindo o alinhamento da tabuada!

### Strings (Seqüências)

Em C uma string é simplesmente uma matriz de caracteres finalizada por um código ASCII zero. Sua declaração será:

```
char str[<inteiro>];
```

onde a variável str poderá conter no máximo o número de letras igual ao especificado. Por exemplo se declararmos

```
char a[80];
```

A variável “a” poderá conter 80 elementos onde a primeira letra estará em a[0] e a octogésima em a[79].

Algo muito importante a ser lembrado é que C não verifica comprimento de strings como o BASIC ou Pascal. A forma mais simples de evitar-se erros é que a string seja suficientemente grande para conter o que você quiser colocar nela. Um caractere zero binário é colocado no final de sua string, portanto se você digitar a palavra alô para uma variável chamada “saudar”, esta deverá ter sido declarada no mínimo como:

```
“char saudar[03];”.
```

```
A 1 ô \0
1 2 3 4
```

**Exemplo 3:** Apresente um nome digitado pelo usuário.

```
main()
{
int cont;
char nome[10];
printf("Digite um Nome: "); scanf("%s",nome);
for (cont = 0; cont <= 10; cont++)
    printf("%c",nome[cont]);
printf("\n\n%s",nome);
}
```

## Laboratório (Aula 05L)

- 1- Dados 3 números, imprima o maior deles e o menor deles.
- 2- Dados 3 números, imprima a média destes números. Verifique se todos são positivos. Caso algum número seja negativo, indique ao usuário que a média não será calculada.
- 3- Elabore programa que leia “n” números digitados e apresente sua média.
- 4- Elabore programa que imprima a média de “n” números, excluindo o menor deles.
- 5- Dados “n” números apresente a média entre o maior e o menor número da seqüência fornecida.

## Tipos de Dados (Aula 06T)

No momento dispomos de conhecimento para elaboração de programas básicos para construção de pequenos programas, pois conhecemos instruções de entrada de dados (scanf), de saída (printf), tomada de decisão (if) e de contagem (for).

Veremos a seguir Tipos de Dados da linguagem C, variáveis, operadores, e demais instruções básicas. Devemos procurar compreender a utilidade das declarações que serão exibidas a seguir, relacionando estes recursos com os exemplos e exercícios vistos anteriormente.

Semelhante ao BASIC, Pascal e COBOL, a linguagem C necessita que todas as variáveis tenham seus tipos definidos. C aceita tipos básicos (caractere, inteiro, ponto flutuante, dupla precisão e sem valor) e modificadores (sinal, sem sinal, longo e curto) que podem alterar os tipos básicos. Observemos as tabelas abaixo para melhor entendimento:

**Tabela de Tamanhos e Escala de Tipos Básicos**

| Tipo   | Extensão | Escala Numérica em bits |
|--------|----------|-------------------------|
| char   | 8        | 0 a 255                 |
| int    | 16       | -32768 a 32767          |
| float  | 32       | 3.4E-38 a 3.4E+38       |
| double | 64       | 1.7E-308 a 1.7E+308     |
| void   | 0        | sem valor               |

### Tabela com Combinações possíveis de Tipos Básicos e seus Modificadores

| Tipo                  | Extensão | Escala                   | Obs.             |
|-----------------------|----------|--------------------------|------------------|
| char                  | 8        | -128 a 127               |                  |
| unsigned char         | 8        | 0 a                      | 255              |
| signed char           | 8        | -128 a 127               | ver char         |
| int                   | 16       | -32768 a 32767           |                  |
| unsigned int          | 16       | 0 a 65535                |                  |
| signed int            | 16       | -32768 a 32767           | ver int          |
| short int             | 16       | -32768 a 32767           | ver int          |
| unsigned short int    | 16       | 0 a 65535                | ver unsigned int |
| signed short int      | 16       | -32768 a 32767           | ver int          |
| long int              | 32       | -2147483648 a 2147483647 |                  |
| signed short int      | 32       | -2147483648 a 2147483647 | ver long int     |
| unsigned short int    | 32       | 0 a 4294967295           |                  |
| float                 | 32       | 3.4E-38 a 3.4E+38        |                  |
| double                | 64       | 1.7E-308 a 1.7E+308      |                  |
| long double<br>double | 64       | 1.7E-308                 | a 1.7E+308 ver   |

Nota: Alguns modificadores resultam em combinações exatamente iguais a de outras declarações. Algumas declarações apresentadas acima não são suportadas pelo Classic C, notadamente aquelas com 64 bits, só disponíveis nas mais potentes implementações de compiladores C.

Exemplo 1: Mesmo número com 2 representações diferentes e Erro.

```
main()
{
int a;
printf("Digite um numero: ");
scanf("%d", &a);
printf("%d %5d %f", a, a, a);
}
```

Para a = 67, teremos: "67 67 0.000000"! Observe que a representação em ponto flutuante foi truncada!

Exemplo 2: Mesmo número com 2 representações diferentes Corretas.

```
main()
{
float a;
printf("Digite um numero: ");
scanf("%f", &a);
printf("%f %e", a, a);
}
```

Simulando obtemos:  
Digite um numero: 65  
65.000000 6.500000E+01

Exemplo 3: Caracter sendo tratado como número, sem qualquer ERRO de execução ou de compilação.

```
main()
{
int a;
printf("Digite um numero: ");
scanf("%d", &a);
printf("%d %u %c", a, a, a);
}
```

Para a = 67, teremos 67 67 C

Para a = -191, teremos -191 65345 A

Exercício4: Elabore tabela de Conversão de Temperaturas entre as Escalas Celsius e Fahreint.

Algoritmo:

*inteiro fahr*

*flutuante celsius*

*para fahr de 0 até 300 passo 20 faça*

*imprima fahr*

*celsius = 5.0/9.0\*(fahr-32)*

*imprima celsius*

Programa:

```
main()
{
int fahr;
float celsius;
for (fahr = 0; fahr <= 300; fahr = fahr + 20) {
    printf("%4d", fahr);
    celsius = (5.0/9.0)*(fahr-32);
    printf("\t%6.1f\n", celsius);
}
}
```

Note que quando dois ou mais comandos devam ser executados, devemos **obrigatoriamente** utilizar chaves para delimitar a seqüência de instruções a ser observada. Analisando o programa a seguir, percebemos que:

```

if (x == 0) {
    printf("A");
    printf("B");
}
else
    printf("C");

```

*Se  $x = 0$  então serão impressos AB senão impresso C*

```

if (x == 0)
    printf("A");
printf("B");

```

*Se  $x = 0$  então serão impressos AB senão impresso B*

**Exercício:** Programa Fatorial.

Algoritmo:

*inteiro  $p, r, s$*

*$s = 1$*

*leia  $r$*

*para  $p$  de  $r$  até 1 faça*

*$s = s * r$*

*imprima  $s$*

Linguagem C:

```

main()
{
    int i, j;
    float f=1;
    printf("Digite um numero: ");
    scanf("%d", &j);
    for(i=1; i<=j; i++)
        f=f*i;
    printf("O fatorial de %d e' %7.0f.", j, f);
}

```

Alternativamente poderíamos resolver este mesmo problema utilizando funções.

```

main()
{
    int j;
    printf("Digite um numero: ");
    scanf("%d", &j);
    printf("O fatorial de %d e' %d", j, fat(j));
}

```

```

int fat(n)
int n;
{
int i, f=1, z=3;
for(i=1; i<=n; i++)
    f = f*i;
return f;
}

```

## Recursividade

A Recursividade é o ato da função chamar a si mesma com a finalidade de resolver determinado problema. O problema do fatorial também pode ser utilizado para demonstrar a recursividade como segue:

```

main()
{
int j;
printf("Digite um numero: ");
scanf("%d", &j);
printf("O fatorial de %d e' %d", j, fat(j));
}

```

```

int fat(n)
int n;
{
int i, f=1, z=3;
if (n == 1)
    return 1;
else
    return fat(n-1) * n;
}

```

## Laboratório (Aula 06L)

- 1- Dado um número inteiro e uma razão, calcule a somatória dos primeiros 5 termos da Progressão Aritmética, determinada por estes números.
- 2- Processe os 3 exemplos do fatorial vistos na aula de teoria.
- 3- Exiba os primeiros "n" termos da PA de razão 2 e termo inicial 4.
- 4- Exiba os primeiros "n" termos da PA de razão 3 e termo inicial 3.

## Variáveis e Operadores (Aula 07T)

*Todas as variáveis devem ser delaradas desta forma: "tipo lista\_de\_variáveis;"*

Exemplos:

```
int i,j,l;
short int si;
unsigned int ui;
double balanco, consolidacao;
```

De maneira semelhante ao que ocorre no Pascal e demais linguagens estruturadas, em C as variáveis tem seus valores localizados nas rotinas onde foram declaradas (escopo).

Basicamente, as variáveis podem ser declaradas fora das funções (globais) que valem para todas as funções do programa. Podem ser declaradas dentro de uma função (locais) sendo desconhecida no restante do programa. Além disso podem ser usadas para passagem de valores entre funções (parâmetros).

Exemplo 1: Variáveis locais e globais; parâmetros.

```
int soma;                /* global */
main()
{
    int cont;            /* local */
    soma = 0;           /* soma(global) = 0 */
    for (cont = 0; cont < 10; cont ++){
        total(cont);
        display();
    }
}

total(x)
int x;                  /* x é parâmetro e vale cont */
{
    soma = x + soma;
}

display()
{
    int cont;          /* cont é local e difere de cont da função main()
    */
    for (cont = 0; cont < 10; cont++) printf("-");
    cont++;            /* equivale a cont = cont + 1 */
    printf("A Soma atual é %d\n", soma);
}
```

Resultando em 0 1 3 6 10 15 21 28 36 45

## ***Operadores***

C é uma das linguagens com maior número de operadores, devido possuir todos os operadores comuns de uma linguagem de alto nível, porém também possuindo os operadores mais usuais a linguagens de baixo nível. Para fins didáticos, dividiremos os operadores em aritméticos, lógicos e de bits. No momento abordaremos apenas as duas primeiras classes.

### **Operadores Aritméticos**

| Operador | Ação                     |
|----------|--------------------------|
| +        | Adição                   |
| *        | Multiplicação            |
| /        | Divisão                  |
| %        | Resto de Divisão Inteira |
| -        | Subtração o menos unário |
| --       | Decremento               |
| ++       | Incremento               |

### **Operadores Relacionais e Lógicos**

| Operador | Ação               |
|----------|--------------------|
| >        | Maior que          |
| >=       | Maior ou igual que |
| <        | Menor que          |
| <=       | Menor ou igual que |
| ==       | Igual a            |
| !=       | Diferente de       |
| &&       | Condição “E”       |
|          | Condição “OU”      |
| !        | Não                |

**Observação:** Em C o resultado da comparação será ZERO se resultar em FALSO e DIFERENTE DE ZERO no caso de obtermos VERDADEIRO num teste qualquer. Programadores experientes utilizam-se desta conclusão em alguns programas, onde “inexplicavelmente” algo é testado contra ZERO.

### **Comparações e Testes**

*Observemos antes de mais nada que ++x é diferente de x++!*

Se

```
x = 10;  
y = ++x;
```

então

x = 11 (pois x foi incrementado) e y = 11

porém Se

```
x = 10;  
y = x++;
```

então

x = 11 e y = 10 (pois x foi atribuído a y ANTES de ser incrementado)

Se

```
x = 1;  
y = 2;  
printf(“%d == %d e’ %d\n”,x,y,x==y);
```

então resultaria em 1 == 2 0 (pois a expressão é falsa)

```
if (10 > 4 && !(10 < 9) || 3 <= 4)
```

resultaria em Verdadeiro pois dez é maior que quatro E dez não é menor que nove OU três é menor ou igual a quatro

### ***Operador sizeof***

Este operador retorna o tamanho da variável ou tipo que está em seu operando. Por exemplo “sizeof(char)” resultaria em 1.

### ***Conversões de Tipos***

Quando forem misturadas variáveis de diferentes tipos, o compilador C converterá os operandos para o tipo de operando maior, de acordo com as regras descritas a seguir:

- 1- Todo char e short int é convertido para int. Todo float é convertido para double.
- 2- Para os demais pares de operandos valem as seguintes regras em seqüência:
  - 2.1- Se um operando for long double, o outro também o será.
  - 2.2- Se um operando for double, o outro também o será.
  - 2.3- Se um operando for long, o outro também o será.
  - 2.4- Se um operando for unsigned, o outro também o será.

**Nota:** Devemos observar que o compilador C é bastante flexível e pouco vigilante, comportando-se de maneira muito diferente de um compilador Clipper ou Pascal, sempre vigilantes com relação aos tipos das variáveis. De fato aqueles compiladores podem gerar executáveis misturando tipos, porém a ocorrência de erros de execução é quase inevitável. Ao contrário destes compiladores, os compiladores C “ajeitam” as coisas para o programa funcionar da “melhor maneira possível”, o que não significa em hipótese alguma que os resultados serão os esperados por programadores “relapsos”. Assim esta boa característica dos compiladores C, pode transformar-se numa autêntica “bomba relógio” para programas não muito bem elaborados.

## Laboratório (Aula 07L)

1- Observe o programa a seguir.

```
main()
{
  int i=1, j=2, k=3, l=4;
  i++;
  k=++i;
  l=j++;
  ++j;
  printf("%d %d %d %d", i, j, k, l);
}
```

Constata os resultados do programa acima.

- 2- Elabore programa contendo 2 variáveis x e y, que atribuam valores a i e j desta forma:  $i = ++x$  e  $j = y++$ . Constata os resultados.
- 3- Dados 3 números, os imprima em ordem crescente usando apenas 1 comando printf.
- 4- Dados 2 números inteiros, imprima 1 se ambos forem positivos ou negativos, 2 se tiverem sinais opostos ou 3 se um deles for zero.

5- Observe o programa a seguir.

```
main()
{
  int i=1, j=2, k, l;
  i++;
  k=++i+k;
  l=j++ +1;
  ++j;
  printf("%d %d %d %d", i, j, k, l);
}
```

Processe o programa 3 vezes. Justifique os resultados.

- 6- Utilize função DOSTIME, disponível no Classic C.

7- Processe o programa a seguir e constate os resultados

```
main()
{
int i=1,j=2,k,l;
i++;
k=++i +k;
l=j++ +l;
++j;
printf("%d %d %d %d",i,j,k,l);
}
```

## Tomadas de Decisão - Parte II (Aula 08T)

Analisaremos mais detidamente o comando “if” e também os comandos “switch” e “?”, destinados a promover desvios condicionais em nossos programas.

**if**

### *Sintaxe*

```
if <condição>
    <bloco de comandos>;
[else
    <bloco de comandos>;]
```

Exemplos: Comparações Simples, uso ou não do else.

```
if (t == 0)
    printf("T vale Zero");
if (t == 2 || t = 4) {
    printf("T vale Dois\n");
    printf("ou T vale Quatro");
}
else {
    printf("T nao e' 2 ou 4\n");
    if (t > 10)
        printf("E supera 10");
    else
        printf("Mas nao supera 10");
}
```

**Exemplo:** Evitar-se divisões por Zero, usando recursos do comando if.

```
main()
{
int a,b;
printf("Digite 2 números: ");
scanf("%d %d",&a,&b);
if (b)
    printf("%f",a/b);
else
    printf("Nao posso dividir por zero\n");
}
```

### ***Operador ?***

A declaração Se—Então—Senão pode ser substituída por:

Exp1 ? Exp2 : Exp3

Se Exp1 for verdadeira, então Exp2 é avaliada tornando-se o valor de Exp1, senão Exp3 é que será avaliada e tornar-se-á o valor da expressão.

Exemplo: Uso do ?.

```
x = 10;
y = (x > 20) ? 200 : 100;
assim y valerá 100
```

### ***Comando switch***

Diversas vezes precisamos determinar se um valor encontra-se numa lista de valores. Apesar de podermos usar uma seqüência de ifs, este recurso além de não ser elegante, por vezes confunde o entendimento do programa. Vejamos uma opção melhor: o comando switch.

Sintaxe:

```
switch <variável> {
    case <constante 1> :
        <comandos>;
        [break;]
    case <constante 2> :
        <comandos>;
        [break;]
    case <constante 3> :
        <comandos>;
        [break;]
    [default :
        <comandos>;]
}
```

Observe que “break” serve para terminar a seqüência de comandos em execução, por serem opcionais, se forem suprimidos permitem que o “case” a seguir seja executado, sem haver qualquer quebra na seqüência do processamento.

Exemplo: Frases Montadas

```
main()
{
    int t;
    for (t = 0; t < 10; t ++ )
        switch (t) {
            case 1:
                printf("Agora");
                break;
            case 2:
                printf("e'");
            case 3:
            case 4:
                printf("hora ");
                printf("de todos os homens bons\n");
                break;
            case 5:
            case 6:
                printf("trabalharemos");
                break;
            case 7:
            case 8:
            case 9:
                printf("-");
        }
}
```

Resultará em:

Agora é hora de todos os homens bons  
hora de todos os homens bons  
hora de todos os homens bons  
trabalharem trabalharem ---

## Laboratório (Aula 08L)

- 1-Elabore programa que solicite 2 números e uma operação matemática elementar (+-\*/) e a execute.
- 2-Elabore programa que imprima a seqüência de frases abaixo, usando switch.  
Verde Verde Vermelho  
Amarelo  
Amarelo  
Amarelo  
Azul Branco e Preto  
Branco e Preto
- 3-Elabore programa que 'z' termine com o valor 4 se 'a' for maior que 10 ou 5 se 'a' for menor ou igual a 10. Os comandos if e switch não poderão ser utilizados.
- 4-Elabore um menu de opções com 4 situações diversas, utilizando switch.
- 5-Elabore programa que permita a 5 pessoas escolherem sua cor favorita entre Verde, Vermelho, Amarelo, Azul, Laranja ou Roxo e exiba os resultados.
- 6-Elabore programa que permita a escolha entre 1, 2 e 3 ou indique erro de escolha.

## Loops (Aula 09T)

Estruturas de repetição normalmente usadas nos programas, terão em C três sintaxes distintas (for, while e do-while), cada uma delas indicada para determinado tipo de necessidade.

### for

*Sintaxe:*

*for (<início>;<condição>;<incremento>) <comando>;*

Além da sintaxe vista anteriormente, “for” permite a escrita de expressões mais elaboradas, sem qualquer paralelo nas linguagens BASIC, Pascal e COBOL, como pode ser vista a seguir:

```
for (x=0, y=0; x+y<100; ++x, y=y+x)
printf ("%d", x+y);
```

Esta instrução inicializaria x e y com zero, incrementando x de 1 em 1 e y receberia seu valor acrescido do de x. O resultado a cada iteração seria impresso desta forma: 0 (x=0 e y=0) 2 (x=1 e y=1) 5 (x=2 e y=3) 9 14 e assim sucessivamente.

**Exemplo 1: Contagem simples com condição no teste “for”.**

```
main()
{
int i,j,resposta;
char feito = ' ';
for (i=1;i<100 && feito != 'N';i++) {
    for (j=1;j<10;j++) {
        printf("Quanto e' %d + %d? ",i,j);
        scanf("%d",&resposta);
        if (resposta != i+j)
            printf("Errou!\n");
        else
            printf("Acertou!\n");
    }
    printf("Mais? (S/N) ");
    scanf("%c",&feito);
}
}
```

**Exemplo 2: Contagem com funções nos argumentos do “for”.**

```
main()
{
int t;
for (prompt();t=readnum();prompt())
    sqrnum(t);
}

prompt()
{
printf("Digite um número inteiro!");
}

readnum()
{
int t;
scanf("%d",&t);
return t;
}

sqrnum(num)
int num;
{
printf("%d\n",num*num);
}
```

### ***Loops Infinitos***

```
for(;;)
printf("Este loop rodará eternamente!\n");
```

A ausência de condições de inicialização, continuidade e terminação, causarão um processo contínuo e teoricamente infinito (veremos posteriormente a intrusão **break**, que tem a capacidade de encerrar um processo assemelhado ao exemplificado).

### ***Loop Vazio***

```
for(i=0;i<10;i++);
```

A presença do ponto e vírgula finalizando o comando, força a execução do loop sem que seja executado qualquer outro comando.

### ***Loop Finito***

Ao contrário de outras linguagens que não permitem o término do loop a não ser quando a condição de finalização for satisfeita, a linguagem C permite que um loop seja interrompido antes de seu término normal (desestruturação) sem que exista qualquer tipo de inconveniente. O comando “break” causa a interrupção conforme pode ser visto a seguir:

```
for(;;) {
    scanf("%d",&c);
    if (c == 'A')
        break; /* interrompe o que deveria ser um anel eterno */
}
printf("Fim do Loop!");
```

### **Conclusão**

As características do comando “for”, como pudemos notar são bem superiores as dos comandos “for” das linguagens BASIC, Clipper, Pascal e COBOL. Enquanto em algumas daquelas linguagens é recomendável que seu uso seja evitado, em C seu uso é normalmente o mais recomendado, pois substitui geralmente com vantagem seu análogo “while”, como veremos a seguir.

### ***Loop while***

#### *Sintaxe:*

```
while <condição> <comando>;
```

O loop se repete, enquanto a condição for verdadeira.

### Exemplo: Contagem

```
main()
{
  int i;
  while (i < 10) {
    printf("%d", i);
    i--;
  }
}
```

### ***Loop do/while***

Ao contrário das estruturas “for” e “while” que testam a condição no começo do loop, “do / while” sempre a testa no final, garantido a execução ao menos uma vez da estrutura. Este comando é similar (porém não idêntico) ao “repeat” da linguagem Pascal.

Sintaxe:

```
do {
  <comandos>;
} while <condição>;
```

Exemplo: Término determinado pelo usuário.

```
main()
{
  int num;
  do {
    scanf("%d", &num);
  } while (num < 100);
}
```

### **Laboratório (Aula 09L)**

- 1- Retome o programa da tabuada, porém agora imprimindo todas de 1 ao 10.
- 2- Elabore programa que decida se um número informado é primo.
- 3- Imprima todos os números primos dentro de um intervalo fornecido pelo usuário.
- 4- Dada uma seqüência de números reais, calcule sua média e desvio padrão.
- 5- Dada uma seqüência de números reais, imprima a mediana da série.
- 6- Retome o exercício 2 e o resolva usando do/while e while.

## Comandos Desestruturadores (Aula 10T)

Vimos anteriormente o comando “break” finalizando opções do comando “switch” e também terminando um loop “for”. Apesar deste último tipo de uso não ser recomendado por alguns defensores da programação estruturada, quando usado de forma adequada (isto é aumentando a velocidade de execução sem prejudicar a compreensão do programa), não há qualquer inconveniente para esta utilização. Aproveitamos para lembrar que a linguagem C é conhecida como “a linguagem dos profissionais” não tendo a pretensão de ser compreendida por leigos como por exemplo as chamadas linguagens de comitês (BASIC e COBOL). O comando “break” é análogo ao comando “exit” da linguagem Clipper.

### break

*Exemplo: Loops encadeados terminados com uso do “break”*

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

Observe que break, quebra apenas a estrutura “for” mais interna, a externa será processada até o final normalmente.

### continue

O comando “**continue**” funciona de maneira análoga ao “**break**”, contudo ao invés de forçar o encerramento do loop, força nova iteração (semelhante a instrução “loop” da linguagem Clipper) saltando o código entre seu uso e a marca de término do loop.

*Exemplo: Imprimir somente os números pares entre 1 e 100.*

```
main()
{
  int x;
  for (x=0; x<100; x++) {
    if (x%2) continue;
    printf("%d", x);
  }
}
```

Desta forma toda vez que for gerado um número ímpar “if” será executado saltando o comando “printf”. “Continue” assemelha-se ao “loop” da linguagem Clipper.

## **goto**

De maneira geral o comando “goto” é considerado o grande vilão das linguagens de programação e geralmente seu uso é desaconselhado quando não proibido por equipes de desenvolvimento e por autores de manuais de linguagens.

Devemos dizer a priori, que uma instrução apenas não teria o poder de causar tantos estragos, porém certos programadores ...

“Goto” apesar de não ser imprescindível para escrita de programas em linguagem C e de ter seu uso restrito a condições particulares, pode ajudar não só a tornar a execução de um programa mais rápida como também mais clara (!!), conforme veremos posteriormente.

“Goto” necessita de um rótulo para poder operar. O rótulo nada mais é que um identificador válido em C seguido de dois pontos. Além disso o rótulo deverá pertencer a função onde se encontra o “goto”. Um uso indevido do “goto” está exemplificado a seguir:

```
x = 1;
loop1:
x++;
if (x<100)
    goto loop1;
```

Preferencialmente deveríamos ter usado uma estrutura “while” ou “for” para elaboração deste trecho de programa.

### ***Exemplo: Uso adequado do “goto”***

O esboço de programa a seguir, estruturado, é bastante confuso, conforme é fácil observar:

```
feito = 0;
for (...) {
    for (...) {
        while (...) {
            if (...) {
                feito = 1;
                break;
            }
            .
            .
            .
        }
        if (feito) break;
    }
    if (feito) break;
}
if (feito)
    break;
```

Vejamos a solução desestruturada (usando Goto)

```
for (...) {
    for (...) {
        while (...) {
            if (...)
                goto stop;
        }
        .
        .
        .
    }
}

stop:
printf("Saída!");
```

Normalmente soluções não estruturadas são mais rápidas que aquelas estruturadas, e neste caso a solução também é muito mais clara, justamente pela utilização adequada do “goto”.

## Velocidade x Estilo x Clareza

As três versões do programa de verificação de um número primo, servem para nos mostrar as diferenças de desempenho que podem ocorrer, independente da linguagem utilizada, somente em função do algoritmo que utilizarmos. Processe estes exemplos e constate as diferenças:

Versão 1: Algoritmo Estruturado, sem conceituação matemática:

```
main()
{
    int i,ini,fim,n,j,nao;
    char ac[80];
    cls();
    printf("Digite extremo inferior: "); scanf("%d",&ini);
    printf("\nDigite extremo superior: ");scanf("%d",&fim);
    dostime(ac, 2);
    puts(ac);
    for(i=ini;i<=fim;i++) {
        nao = 1;
        for(j=2;j<i;j++)
            if(i % j == 0)
                nao = 0;
        if (nao || i == 2)
            printf("%d ",i);
    }
    printf("\n");
    dostime(ac, 2);
    puts(ac);
}
```

## Versão 2: Algoritmo Estruturado com conceituação matemática.

```
main()
{
    char ac[80];
    int j,i,ini,fim,n,nao;
    double r;
    cls();
    printf("Digite extremo inferior: "); scanf("%d",&ini);
    printf("\nDigite extremo superior: ");scanf("%d",&fim);
    dostime(ac, 2);
    puts(ac);
    for(i=ini;i<=fim;i++) {
        nao = 1;
        j = 2;
        r = i;
        r = sqrt(r);
        while (j<=r) {
            if(i % j == 0)
                nao = 0;
            j++;
        }
        if (nao || i == 2)
            printf("%d ",i);
    }
    printf("\n");
    dostime(ac, 2);
    puts(ac);
}
```

### Versão 3: Algoritmo com conceituação matemática, com liberdades na estruturação

```
main()
{
    char ac[80];
    int j,i,ini,fim,n,nao;
    double r;
    cls();
    printf("Digite extremo inferior: "); scanf("%d",&ini);
    printf("\nDigite extremo superior: ");scanf("%d",&fim);
    dostime(ac, 2);
    puts(ac);
    for(i=ini;i<=fim;i++) {
        nao = 1;
        if (i % 2 == 0)
            nao = 0;
        else {
            j = 3;
            r = i;
            r = sqrt(r);
            while (j<=r) {
                if (i % j == 0) {
                    nao = 0;
                    break;
                }
                j =j + 2;
            }
        }
        if (nao || i == 2)
            printf("%d ",i);
    }
    printf("\n");
    dostime(ac, 2);
    puts(ac);
}
```

### **Laboratório (Aulas 10L,11L e 12L)**

Elabore um jogo onde o humano tenta descobrir o número do computador e vice-versa. Os números devem estar entre 1 e 1023, sendo que o jogo só termina quando um (ou ambos) jogadores acertarem o número de seu oponente. O Empate ocorre quando os dois jogadores acertarem o número de seu oponentes na mesma jogada.

### Solução:

```
/* Autores: Deusdeth, Luis Fernando, Ana */
int verific=0,numm,numh,ia=0,ib=1023,contm=0,conth=0,resp,palpm;
char nome[32];
main ()
{
  cls ();
  apresent ();
  nome_hum ();
  num_maq ();
  while (verif == 0){
    joga_hum ();
    joga_maq ();
    rotina_verif ();
    rotina_verif1 ();
    rotina_verif2 ();
  }
  rotina_venceu ();
}

apresent ()
{
  puts ("|*****|");
  puts ("|*          J O G O      A L O - V E J A          *|");
  puts ("|***** ( A P R E S E N T A C A O ) *****|");
  puts ("|* - JOGO ENTRE A MICRO E VOCE. O MICRO IRA' ESCOLHER UM *|");
  puts ("|*   NUMERO E VOCE OUTRO NUM INTERVALO ENTRE 1 E 1023.   *|");
  puts ("|* - CADA UM IRA' TENTAR DESCOBRIR O NUMERO DO Oponente, *|");
  puts ("|*   ATE' QUE UM DOS JOGADORES ADIVINHE O NUMERO DO OUTRO*|");
  puts ("|* - O MICRO IRA' INFORMAR SE O SEU PALPITE FOI CORRETO, *|");
  puts ("|*   BAIXO OU ALTO.                                     *|");
  puts ("|* - VOCE DEVERA' FAZER O MESMO, INFORMANDO:           *|");
  puts ("|*   (1) PARA UM CHUTE BAIXO;                          *|");
  puts ("|*   (2) PARA UM CHUTE ALTO;                          *|");
  puts ("|*   (3) PARA CERTO.                                    *|");
  puts ("|*****|");
}

nome_hum ()
{
  printf ("INFORME O SEU NOME: ");
  gets (nome);
}

num_maq ()
{
  numm=rand()/32;
}
```

```

joga_hum ()
{
printf ("%s, tente acertar o numero que eu escolhi : ",nome);
scanf ("%d",&numh);
puts  ("o resultado sera divulgado apos a jogada do micro");
conth=conth+1;
puts ("*****");
puts  (" ");
}

joga_maq ()
{
palpm=(ia+ib+1)/2;
printf ("%s, acho que voce pensou no numero %d",nome,palpm);
puts  (" ");
printf ("digite (1) baixo, (2) alto ou (3) certo : ");
scanf ("%d",&resph);
contm=contm+1;
puts ("*****");
puts  (" ");
}

rotina_verif ()
{
if (numh == numm)
    verif = verif + 1;
else
    if (resph == 3)
        verif = verif + 1;
}

rotina_verifl ()
{
if (numh > numm){
    puts (" ");
    printf ("seu chute foi alto");
    puts (" ");
}
else
    if (numh < numm){
        puts (" ");
        printf ("seu chute foi baixo");
        puts (" ");
    }
}
}

```

```

rotina_verif2 ()
{
if (resph == 1)
    ia = palpm;
else
    if (resph == 2)
        ib = palpm;
}

rotina_venceu ()
{
if (numh == numm)
    if (resph == 3)
        printf("\nOcorreu Empate! \n\n* fim do jogo.\n");
    else {
        puts (" ");
        printf("* %s parabens, voce acertou em %d tentativas.",nome,conth);
        puts (" ");
        puts ("* fim do jogo.");
        puts (" ");
    }
else
    if (resph == 3){
        puts (" ");
        printf("* %s o micro acertou em %d tentativas.",nome,contm);
        puts (" ");
        puts ("* fim do jogo.");
        puts (" ");
    }
}

```

### **Exercício:**

Elabore programa baseado no Jogo da Velha, onde o jogador humano enfrenta a máquina. Observe que o computador nunca deverá ser derrotado, podendo eventualmente derrotar o jogador humano.

### **Matrizes (Aula 11T)**

Correspondem a elementos do mesmo tipo, agrupados sob o mesmo nome e diferenciados entre si através de índices. Na linguagem C, todos os vetores ou matrizes consistem em posições contíguas, sendo que o endereço mais baixo corresponde ao primeiro elemento e o endereço mais alto ao último elemento.

### **Declaração**

Deve ser feita nestes formatos:

```
char nome[20];
```

```

float preco[30];
int m[5][3]; /* bidimensional */
char c[3] = {'f','i','m'}; /* declarada e inicializada */
char d[3] = "fim"; /* idem */
int a[5] = {1,10,3,5,30} /* declarada e inicializada, numérica */

```

**Exemplo 1: Imprima 5 números na ordem oposta a que forem informados.**

#### Solução sem matriz

```

main()
{
int a,b,c,d,e;
scanf("%d %d %d %d %d",&a,&b,&c,&d,&e);
printf("%d %d %d %d %d",e,d,c,b,a);
}

```

#### Solução com matriz

```

main()
{
int i,a[5];
for (i=1;i<=5;i++) {
printf("Elemento %d: ",i);
scanf("%d",&a[i]);
}
puts(" ");
for (i=5;i>0;i--)
printf("Elemento %d: ",a[i]);
}

```

**Exemplo 2: Tabuada em matrizes.**

```

main()
{
int a[10], i, t = 3;
for (i=0;i<11;i++)
a[i] = i * t;
for (i=0;i<11;i++)
printf("%d * %d = %d",i,t,a[i]);
}

```

**Exercício:** Retome o Exercício da Aula de Laboratório 8, exercício 3 e o resolva com o uso de matrizes.

## Laboratório (Aula 11L)

Jogo da Velha: Continuação.

## Ordenação (Aula 12T)

Exemplo: Dada uma série de números inteiros, ordene-a de forma crescente.

Algoritmo:

```
leia n
para i de 1 até n
  leia a[i]
para i de 1 até n-1 faça
  para j de i+1 até n faça
    se a[i] > a[j] então
      m = a[i]
      a[i] = a[j]
      a[j] = m
para i de 1 até n
  imprima a[i]
```

Em C teríamos:

```
main()
{
int x,y,z,w,a[100];
do
{
traca_linha()
puts("\Programa para ordenar 'n' numeros digitados.");
traca_linha();
printf("Quantos Numeros (0=fim) -> ");
x = leintf();
if (x==0) break;
traca_linha()
for (y = 0; y < x; y++) {
printf("a[%2d]= ",y);
a[y] = leintf();
}
traca_lina();
for (y=0;y<(x-2),y++)
for (z=y+1;z<(x-1);z++)
if (a[y] > a[z]) {
w = a[y];
a[y] = a[z];
a[z] = w;
}
for (y=0;y<(x-1);y++)
printf("%d\n",a[y]);
} while(1);
}

traca_linha()
{
int x;
for (x=1;x != 80; x++)
putchar('=');
putchar('\n');
}

leintf()
{
char s[20];
gets(s);
return(atoi(s));
}
```

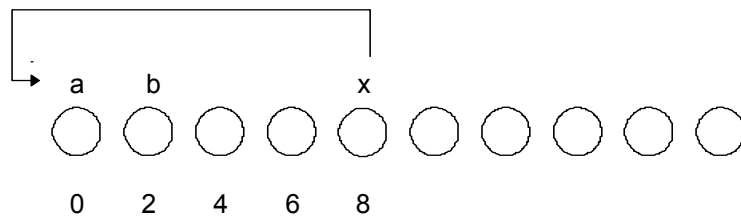
## Laboratório (Aula 12L)

Jogo da Velha: Continuação.

## Ponteiros - Apresentação (Aula 13T)

### Introdução

Ponteiros são endereços, isto é, são variáveis que contêm um endereço de memória. Se uma variável contém o endereço de outra, então a primeira (o ponteiro) aponta para a segunda.



“x” o “ponteiro” aponta para o “inteiro” a

### **Operadores**

&- (E comercial) que fornece o endereço de determinada variável. Não confundir com o operador lógico de operações de baixo nível, de mesmo símbolo. Atribui o endereço de uma variável para um ponteiro.

\*- (Asterístico) que acessa o conteúdo de uma variável, cujo endereço é o valor do ponteiro. Não confundir com o operador aritmético de multiplicação de mesmo símbolo. Devolve o valor endereçado pelo ponteiro.

## Exemplos:

```
main()
{
int *pont, cont, valor;
cont = 100;
pont = &cont;
val = *pont;
printf("%d",val);    /* 100 */
}
```

```
main()
{
char a,b,*x;
b = 'c';
p = &a;
*x = b;
printf("%c",a);    /* c */
}
```

```
main()
{
int x,y,*px,*py;
x = 100;
px = &x;    /* px tem o endereço de x */
py = px;    /* py tem o endereço de x */
y = *py;    /* y vale 100, pois recebe o conteúdo de x */
           /* , através do ponteiro py */
printf("%d %d",x,y);
}
```

## Laboratório (Aula 13L)

- 1- Executar Programa de Ordenação de uma Série de Números.
- 2- Dado um nome invertá-o.
- 3- Calcule a média de “n” números, posteriormente imprimindo-os.
- 4- Dada uma série com “n” números, imprima a média dos n-1 maiores termos, usando matrizes obrigatoriamente.

5- Simule a execução do programa abaixo:

```
#include "stdio.h"
main()
{
int i,k,*pi,*pk;
char a;
i = 2; k = 0;
puts("Qual sera o valor de k? ");
pk = &k;
pi = &i;
*pk = i;
printf("para *pk = i, temos k= %d\n",k);
k = *pi;
printf("para k = *pi, temos k= %d\n",k);
scanf("%c",&a);
}
```

6- Simule a execução do programa abaixo:

```
main()
{
int x,y,*px,*py;
printf("Digite um valor: ");
scanf("%d",&x);
px = &x;
y = *px;
printf("digitou= %d e y= %d\n",x,y);
*px = 8;
printf("valor mudou para %d\n",x);
}
```

7- Simule a execução do programa a seguir:

```
main()
{
char a,b,*p;
b = 'c';
p = &a;
*p = b;
printf("%c",a);
}
```

## **Ponteiros - Conceitos de Endereços (Aula 14T)**

### **Aritmética de Ponteiros**

São válidas as operações de soma e subtração, sendo que seu resultado depende do tipo de variável apontada pelo ponteiro.

## Supondo que

```
int *p, x;  
char *q, a;
```

se

```
q = &a;  
p = &x;
```

e ainda que

```
a—endereço 100  
x—endereços 101/102
```

então `q++` --> q “apontará” para o endereço 101  
`p++` --> p “apontará” para o endereço 103

Este conceito é particularmente importante no que se refere a matrizes pois como se sabe, matriz nada mais é que um conjunto de variáveis do mesmo tipo, dispostas seqüencialmente em memória.

Por conterem endereços, ponteiros permitem apenas as operações de soma e subtração.

Supondo que:

```
int i, *pi;  
char c, *pc;  
float f, *pf;
```

Supondo ainda que `pi`, `pc` e `pf` apontem para `i`, `c` e `f` que estariam com os seguintes endereços: 2340, 2345 e 2350.

## *Se*

`pc = pc + 1`, então `pc` valerá 2346, pois variáveis caracteres possuem apenas 1 byte.

`pi = pi + 1`, então `pi` valerá 2342 (!), pois variáveis inteiras ocupam 2 bytes.

`pf = pf + 5`, então `pf` valerá 2370 (!), pois variáveis pt. flutuante ocupam quatro bytes.

## *Exemplo*

```
int i, *pi;  
char c, *pc;  
float f, *pf;
```

Supondo ainda que `pi`, `pc` e `pf` apontem para `i`, `c` e `f` que estariam com os seguintes endereços: 2340, 2345 e 2350.

## ***Se***

`pc = pc + 1`, então `pc` valerá 2346, pois variáveis caracteres possuem apenas 1 byte.  
`pi = pi + 1`, então `pi` valerá 2342 (!), pois variáveis inteiras ocupam 2 bytes.  
`pf = pf + 5`, então `pf` valerá 2370 (!), pois variáveis pt. flutuante ocupam quatro bytes.

### **Exemplo: Atribuições indiretas**

```
main()
{
  int x, y, *px, *py;
  x = 100;
  px = &x; /* px tem o endereço de x */
  py = px; /* py tem o endereço de x */
  y = *py; /* y vale 100, pois recebe o conteúdo de x, através do
           ponteiro py */
  printf("%d %d", x, y);
}
```

## ***Comparação de Ponteiros***

Por se tratar de endereços os ponteiros podem ser comparados.

### **Exemplo:**

```
if (p > q) puts("p aponta para endereço maior que q");
```

## **Ponteiros - Prática - Conceitos Básicos (Aula 14L)**

Processar exemplos vistos em teoria.

## **Ponteiros Conceitos Avançados (Aula 15T)**

### **Strings**

Consideremos o trecho de código abaixo:

```
char linha[80], *p; *p1;
p = &linha[0];
p1 = linha; /* p1 e p possuem o mesmo endereço, i.é, */
if (p==p1)
puts("iguais!"); /* apontam para o 1º elemento da matriz! */
```

Exemplo: Inverter os elementos de uma string, usando ponteiros.

Solução: Pela maneira convencional teríamos:

```
main()
{
char str[80];
int i;
printf("Digite uma palavra: "); gets(str);
for (i=strlen(str)-1;i>=0;i--)
printf("%c",str[i]);
}
```

com ponteiros teríamos:

```
main()
{
char str[80],*p;
int i;
printf("Digite uma palavra: "); gets(str);
p = str;
for(i = strlen(str) - 1;i>=0;i--)
printf("%c",*(p+i));
}
```

Exemplo: Uso de Ponteiros e String em mesma variável

```
char *p = "Imprimindo Algo\n";
main()
{
int i;
printf("%s\n - por ponteiro...\n\n",p);
for(i=0;p[i];i++)
printf("%c",p[i]);
puts("\n - por caracteres em ponteiro ");
}
```

## Velocidade

O acesso aos elementos da matriz é mais rápido quando feito através de endereços do que seria caso fosse executado pela maneira convencional, porém expressões que envolvam cálculos para se obter um elemento específico da matriz devem ser feitas preferencialmente pelo método convencional, pois expressões complexas envolvendo ponteiros são processadas em velocidade semelhante àquelas feitas pelo modo convencional, que apresenta a vantagem de ser mais facilmente compreendido.

## Matrizes e Ponteiros

*Em C você pode indexar um ponteiro como se este fosse uma matriz!*

Exemplo:

```
main()
{
int i,*p,a[5];
for (i=0;i<=4;i++) {
    printf("Digite o %d elemento",);
    scanf("%d",&a[i]);
}
p = a;
for (i=0;i<=4;i++)
    printf("%d %d %d\n",a[i],p[i],*(p+i));
}
```

## **Ponteiros - Prática - Conceitos Avançados (Aula 15L)**

Processar Exemplos vistos em teoria

## **Ponteiros - Pilhas (Aula 16T)**

Uma pilha é uma lista de variáveis do mesmo tipo (semelhantes a uma matriz, ou mesmo uma matriz), onde utilizamos o conceito de que “o primeiro que entra é o último a sair”. Imaginemos um bloco de folhas. Normalmente utilizaremos primeiro a última folha do bloco (“a de cima”), enquanto que a primeira folha colocada (“a de baixo”) será a última a ser utilizada. Nos exemplos a seguir serão utilizadas duas funções: push() e pop(). Usaremos push() para inserir elementos na pilha e pop() para sacá-los.

Exemplo: Faça com que seja criada uma pilha com no máximo 50 elementos, onde números positivos empilhem elementos na pilha, 0 tire o último elemento da pilha e -1 cause o encerramento do programa:

```

int pilha[50],*p1,*to;
main()
{
int valor;
p1 = pilha;
to = p1;
printf("Numero --> Pilha, 0 recupera e -1 finaliza \n");
do {
scanf("%d",&valor);
if (valor != 0)
push(valor);
else {
valor = pop();
printf("%d\n",valor);
}
} while (valor != -1);
}

push(i)
int i;
{
p1++;
if (p1 == (to + 50)) {
puts("Estouro da Pilha (superior) ");
exit(1);
}
*p1 = i;
}

pop()
{
if ((p1) == to) {
puts("Estouro da Pilha (inferior) ");
exit(1);
}
p1--;
return *(p1+1);
}

```

Imaginemos agora uma pilha onde não soubéssemos antecipadamente o total de elementos de uma pilha, neste caso precisaríamos de instruções que permitissem a manipulação da área de memória livre entre o programa e área de trabalho utilizada pelo programa conforme mostra o esquema abaixo:

## Memória do Sistema

|  |                        |
|--|------------------------|
| Usada para Variáveis Locais e endereços de Rotinas | Do “fim” para o início |
| Memória Livre para Alocação                        |                        |
| Variáveis Globais                                  | Do “início” para o fim |
| Programa   |                        |

As funções `malloc()` e `free()` permitem que utilizemos a área de memória livre para criar nossas variáveis, por exemplo matrizes com limites a serem definidos em tempo de execução.

### Exemplo:

```
char x, y, *px;
main()
{
px = malloc(1000);
if (!px) {
    puts("Memória Insuficiente!");
    exit(1);
}
x = 'a';
push(x);
x = 'b';
push(x);
y = pop();
printf("%c\n", y);
y = pop();
printf("%c", y);
}

push(i)
char i;
{
px++;
*px=i;
}

pop()
{
px--;
return *(px+1);
}
```

Exemplo: Aloque uma certa quantidade de memória e posteriormente escreva letras na área alocada, recupere estas letras e depois libere a memória previamente alocada.

```

main()
{
char c,*p,*q;
int i=0,j,r;
cls();
q = malloc(1000);
p = q;
puts("Digite Letras ... para parar digite <enter>");
for(;;) {
    printf("Digite a %d letra: ",i+1);
    scanf("%c",&c);
    if (c == '\n')
        break;
    *p = c;
    c = '\n';
    p++;
    i++
}
p = q;
printf("\n\nLetras Digitadas:\n");
for(j=1;j<=i;j++) {
    printf("%d Letra = %c\n",j,*p);
    p++;
}
r=free(q);
if (r==0)
    puts("\nLiberada Memória Utilizada!");
else
    puts("\nFalha na Liberação da Memória Alocada!");
}

```

Exemplo: Retome o exercício anterior, porém ao invés de armazenar letras, armazene números.

```
main()
{
char c,*q,*p;
int i = 0, j, r, m, n;
cls();
q = malloc(1000);
p = q;
puts("Digite Numeros ... para parar digite <enter> ");
for(;;) {
printf("Digite o %do. Numero: ", i+1);
scanf("%d",&m);
if (m==0)
break;
*p = m;
m = 0;
p=p+2;
i++;
}
p = q;
printf("\n\nLetras Digitadas: \n");
for(j=1;j<=i;j++) {
printf("%d Letra = %d\n",j,*p);
p=p+2;
}
r=free(q);
if (r==0)
puts("\nLiberada Memoria Usada!");
else
puts("\nFalha na Liberação de Memória Reservada!");
}
```

**Nota:** A característica típica da linguagem C, que facilita a mistura de tipos, permite o endereçamento de variáveis *através de sua posição na memória*, ou seja, podemos usar um ponteiro de caracteres para apontar um número inteiro e recuperá-lo sem qualquer tipo de inconveniente.

Devemos notar entretanto que esta característica pode acarretar grandes problemas, e caso seja utilizada de forma inadequada, pode gerar problemas se acessarmos (ou principalmente se escrevermos) alguma área de memória importante.

Outro ponto importante, é percebermos que *não temos números, caracteres ou qualquer outra coisa armazenadas na memória além de bits*, portanto tudo o que necessitamos saber é *onde* está uma informação procurada e *qual* é seu formato e simplesmente acessá-la (ou substituí-la) de acordo com nossa necessidade.

É desnecessário dizer, que estas são características dos assemblers próprios de cada equipamento, portanto a Linguagem C nos desobriga a aprender o Assembler de cada microprocessador, porém não elimina a necessidade de conhecermos sua arquitetura.

## ***Matriz de Ponteiros e Indexação Múltipla***

São válidas expressões como as que seguem abaixo:

```
int *x[10];
x[2] = &valor;
.
.
.
printf("%d", *x[2]);
```

**ou**

```
main()
{
int x, *p, **q;
x = 10;
p = &x;
q = &p;
printf("%d", **q); /* 10 */
}
```

## **Ponteiros - Prática - Conceitos Avançados (Aula 16L)**

Processar Exemplos de Pilha.

## **Ponteiros Conceitos Complementares (Aula 17T)**

### ***Ponteiros como Strings***

Em C podemos usar o fato do ponteiro nulo (0 binário) ser o terminador de strings combinado a possibilidade de representação de matriz como ponteiros, conforme mostra o exemplo a seguir:

```
char *p = "Frase a demonstrar \n";
main()
{
int i;
printf("%s", p);
for (i=0; p[i]; i++)
    printf("%c", p[i]);
}
```

### ***Problemas a serem evitados com ponteiros***

Os erros vistos a seguir podem passar não ser notados durante a fase de desenvolvimento do sistema e mesmo durante algum tempo em sua fase operacional e ser detectado apenas esporadicamente, tornando muito difícil sua localização pelo programador, pois só serão observados, no caso de serem apontados endereços vitais para o sistema.

Atribuição de posição de memória desconhecida:

```
main()
{
int x, *p;
x = 10;
*p = x;
printf("%d", *p); /* valor desconhecido */
}
```

Observe que 10 é atribuído a uma posição de memória desconhecida, pois o endereço de “p” não é conhecido. Caso “p” estiver apontando para algum endereço vital, o sistema será paralisado, causando suspeita de mal funcionamento (hardware) ou presença de vírus (software). Na verdade, esta ocorrência é o chamado “bug” (pequeno erro no programa), pois somente algumas vezes causará erro. Observe serem grandes as possibilidades deste programa funcionar perfeitamente, pois o ponteiro provavelmente jogará o valor num local não usado. Porém quanto maior for o programa, mais provável será a possibilidade de encontrarmos um erro conforme descrito anteriormente.

Atribuição de valor para o ponteiro:

```
main()
{
int x, *p;
x = 10;
p = x;
printf("%d", *p); /* valor desconhecido */
}
```

Observe que não será impresso 10, mas um valor desconhecido qualquer, pois 10 será atribuído ao ponteiro p, que supostamente contém um endereço e não um valor. Se tivéssemos “p = &x;” aí o programa funcionaria de forma correta.

### ***Passagem de Variáveis ou Valores através Funções***

Quando desejamos que uma função altere o valor de uma variável da função que a chamou, passamos para a função chamada o endereço desta variável (passagem de parâmetro por referência). Quando somente precisamos do valor da variável e não pretendemos alterá-lo na rotina (passagem de parâmetro por valor), passamos diretamente

a variável, conforme visto a seguir:

### Exemplo 1: Passagem por Valor (a mantém seu valor)

```
main()
{
int a,r,x;
printf("Digite um valor: ");
scanf("%d",&a);
x = 2 * a + 3;
r = soma(a);
printf("%d, %d e %d",a,x,r);
}

soma(z)
int z;
{
int x=5;
x = 2 * x + z;
z = 0;
return(x);
}
```

### Exemplo 2: Passagem por Referência (a muda seu valor)

```
main()
{
int a,r,x;
printf("Digite um valor: ");
scanf("%d",&a);
x = 2 * a + 3;
r = soma(&a);
printf("%d, %d e %d",a,x,r);
}

soma(z)
int *z;
{
int x=5;
x = 2 * x + *z;
*z = 0;
return(x);
}
```

### Exemplo 3- Uso de ponteiros em funções.

```
main()
{
int a,b;
a = 100;
b = 20;
swapg(&a, &b);
printf("Maior = %d ",a);
printf("Menor = %d ",b);
}

swapg(c,d)
int *c,*d;
{
int t;
if (*c <= *d)
return; /* nao troca */
t = *c;
*c = *d;
*d = t;
}
```

## Laboratório (Aula 17L)

- 1- Troque o valor de 2 variáveis usando ponteiros, sem usar funções. Compare com o exemplo visto na aula teórica.
- 2- Crie um programa de contagem com loop usando ponteiros.
- 3- Crie um programa com 2 variáveis, sendo que a primeira será do tipo caracter e a segunda do tipo inteiro. Atribua valores da primeira para a segunda e vice-versa através de ponteiros.

## Ponteiros x Matrizes e Entradas e Saídas (Aula 18T)

Exemplo: Retomando o exemplo da tabuada, resolvendo ao “estilo C de programar”

### *Estilo Pascal*

```
main()
{
int a[10],i;
for(i=0;i<10;i++)
a[i] = i*3;
for(i=0;i<10;i++)
printf("%d x 3 = %d \n",i,a[i]);
}
```

Apesar da solução apresentada acima ser correta logicamente, peca por desconsiderar a razão principal de um programa ser escrito em C, a velocidade. Intui-se facilmente que se endereçarmos uma variável por seu endereço, com certeza seremos mais velozes do que se a acessarmos pela tabela de variáveis (como feito no exemplo acima).

Antes de resolvermos novamente este problema, voltemos aos ponteiros neste pequeno programa, que servirá para entendermos a estratégia comumente usada pelos programadores C.

|                   | <i>Endereços</i> |      |      |      |
|-------------------|------------------|------|------|------|
|                   | Instruções       | 1000 | 1001 | 1002 |
| main()            |                  |      |      |      |
| {                 |                  |      |      |      |
| char c,*pc,x;     | declarações      | c    | pc   | x    |
| c = 'A';          | atribuição       | A    | ?    | ?    |
| pc = &c;          | atribuição       |      | 1000 | ?    |
| printf("%c",*pc); | exibir           |      |      | A    |
| x = *pc;          | atribuição       |      |      | A    |
| }                 |                  |      |      |      |

### ***Tabuada ao Estilo C***

```
main()
{
int a[10],i,*p;
p = &a;
for(i=1;i<10;i++)
*(ponteiro+i) = i*3;
for(i=0;i<10;i++)
printf("%d x 3 = %d \n",i,a[i]);
}
```

### ***Entradas e Saídas***

Além das funções “printf” e “scanf”, muito poderosas existem outras instruções de entrada e saída, porém de menores recursos e ocupando menos espaço em memória.

| <b>Instruções</b> | <b>Descrição</b>                             |
|-------------------|--|
| getchar()         | lê um caracter do teclado aguardando <Enter> |
| getche()          | lê um caracter do teclado e prossegue        |
| getch()           | lê um caracter sem eco na tela e prossegue   |
| putchar()         | escreve um caracter na tela                  |
| gets()            | lê uma string do teclado                     |
| puts()            | escreve uma string na tela                   |

Todas as instruções vistas anteriormente tem como argumento uma variável do tipo necessário (caracter ou seqüência de caracteres).

Exemplo 1: Escrever “A” de diversas formas.

```
main()
{
int i;
char c;
i = 65;
c = 'A';
putchar(65);
putchar('A');
putchar(i);
putchar('\n');
}
```

A linguagem C é pródiga em “confundir” dispositivos, como podemos constatar no exemplo a seguir.

**Exemplo 2: Teclado é Arquivo!**

```
main()
{
char s[30];
int c;
while ((c=getchar()) != EOF)
    putchar(c);
get(s);
puts(c);
}
```

Revisando o comando “scanf”, agora estamos em condições de justificar o porquê devemos endereçar as variáveis com &. O sinal % na frente dos especificadores de entrada serve para informar o tipo de dado a ser lido. Um caracter não branco na string faz com que “scanf” leia e desconsidere um caracter coincidente. Por exemplo, a string de controle “%d,%d” faz com que “scanf” primeiro leia um inteiro, depois leia e desconsidere um vírgula e, finalmente, leia outro inteiro. Se não houver ‘,’, “scanf” será encerrada.

O comando “scanf” deve receber valores passados por seus endereços. Devemos lembrar que C trabalha desta forma para chamadas por referência, permitindo que uma função altere o conteúdo de um argumento, como em “scanf(“%d”,&cont);”.

Para leitura de strings (matrizes de caracteres), o nome da matriz sem índice informa o “endereço do primeiro elemento” da matriz, ou seja um ponteiro, portanto não devemos usar &, como em “scanf(“%s”,matriz);”.

Os programadores BASIC devem ter em mente que soluções como as separações por vírgula no comando "INPUT" não funcionaram adequadamente em instruções do tipo "scanf("%d %d",&r,&c);".

No exemplo a seguir o 't' (se digitado!) será descartado, 10 ficará em x e 20 em y como em "scanf("%st%s",&x,&y);".

Outro cuidado a ser tomado pode ser constatado na expressão a seguir "scanf("%s %s",& dado);", somente após você digitar um caracter após um caracter branco. Isto ocorre pois "%s" instruiu "scanf()" a ler e desconsiderar espaços.

Os usuários BASIC (e alguns do Clipper) tem o seguinte costume:

```
INPUT"Digite um Número: ";A%
```

Analogamente pensa-se (de forma errada) que podemos fazer:

```
scanf("Digite um número %d ",&a);
```

algo totalmente sem sentido em C.

Teríamos a seguinte estrutura equivalente:

```
puts("Digite um número: ");
scanf("%d",&a);
```

### *Principais Códigos de Formatação de scanf e printf*

| <b>Código</b> | <b>Significado</b>   | <b>Observações</b> |
|---------------|--|--------------------|
| %c            | lê/escreve um único caracter   |                    |
| %d            | lê/escreve inteiro decimal   |                    |
| %i            | lê/escreve inteiro decimal   |                    |
| %e            | lê/escreve número com ponto flutuante                                      | Notação Científica |
| %f            | lê/escreve número com ponto flutuante                                      |                    |
| %h            | lê/escreve um inteiro curto  |                    |
| %o            | lê/escreve um número octal   |                    |
| %s            | lê/escreve uma string  |                    |
| %x            | lê/escreve um número hexadecimal   |                    |
| %p            | lê/escreve um ponteiro   |                    |
| %n            | receber um valor inteiro igual ao número de caracteres lidos até o momento |                    |
| número        | especificará o total de caracteres a serem lidos para um campo qualquer    |                    |
| %g            | Usa %e ou %f, aquele que for menor   | Somente printf     |
| %%            | Imprime %  | Somente printf     |
| %u            | Decimal sem Sinal  | Somente printf     |
| h             | Modificador short  |                    |
| l             | Modificador long   |                    |

Exemplificando:

%ld- especifica um inteiro longo.

Os modificadores “acompanham” o outro código de formato, modificando sua apresentação.

## **Ponteiros x Matrizes e Entradas e Saídas (Aula 18L)**

- 1- Processar Exemplos vistos em teoria.
- 2- Retome exercícios anteriores, utilizando as instruções de entrada e saída getchar(), getch(), getche(), putchar(), gets() e puts().

## **Operadores e Funções String (Aula 19T)**

### *Operadores*

char strcat(s1,s2)

char \*s1,\*s2;

copia a string s2 para o final de s1.

char strncat(s1,s2,n)

char \*s1,\*s2;

copia no caractere de s2 para o final de s1.

char strcmp(s1,s2)

char \*s1,\*s2;

compara duas strings retomando

0- se iguais

1- se a 1ª. for maior que 2ª.

2- se a 2ª. for maior que 1ª.

char strcpy(s1,s2)

char \*s1,\*s2;

copia a string s2 para a string s1

strlen(s)

char \*s;

retorna o total de caracteres de s, exceto o caracter nulo

atoi(s)

char \*s;

converte string em número inteiro

Podemos “truncar” variáveis tipo string, procedendo como se estas fossem matrizes de caracter.

Exemplo: Escreva o nome do mês correspondente ao número.

```
main()
{
int i,j,k;
char *mes = "JANFEVMARABRMAIJUNJULAGOSSETOUTNOVDEZ";
cls();
printf("Digite o Mes: ");
scanf("%d",&j);
k=j*3-3;
for(i=k;i<k+3;i++)
printf("%c",mes[i]);
}
```

No Exemplo a seguir, determinamos o exato tamanho da variável digitada, através da função “strlen”, e “misturamos” os tipos “tot” e “a” de forma conseqüente.

Exemplo: Determine o “número da sorte” de um nome.

```
main()
{
int tot=0,i,w;
char a,s[20];
cls();
printf("Digite seu Nome: "); scanf("%s",s);
for(i=0;i<=strlen(s)-1;i++) {
a = s[i];
tot = tot + a - 64;
}
printf("%s seu numero da sorte e' %d",s,tot);
}
```

Podemos também converter strings e números (inteiros/fracionários) conforme desejarmos:

### Exemplo: Conversão de String em Número Inteiro

```
main()
{
int i;
char s[10];
printf("Digite uma sequencia de numeros com letras: ");
gets(s);
i = atoi(s);
printf("Numero: %d ",i);
}
```

### Laboratório (Aula 19L)

- 1- Elabore um programa que armazene e exiba notas de alunos de classes de no máximo 4 alunos. Existem apenas 3 classes de alunos.
- 2- Dado um nome, inverta-o.
- 3- Dado um nome, ordene suas letras de forma crescente.

### Entradas e Saídas em Dispositivos - Arquivos (Aula 20T)

As funções e programas a seguir foram concebidos para serem compilados pelo Turbo C, portanto algumas adaptações poderão ser necessárias para sua utilização no Classic C. Entretanto a maior parte dos programas funcionará de forma adequada também no Classic C.

| <b>Nome</b>           | <b>Função</b>                                   |
|-----------------------|---|
| <code>fclose()</code> | Fecha uma fila                                  |
| <code>feof()</code>   | Devolve se fim de fila                          |
| <code>ferror()</code> | Devolve Verdadeiro se um erro tiver ocorrido    |
| <code>fopen()</code>  | Abre uma fila                                   |
| <code>fprint()</code> | Saída   |
| <code>fscanf()</code> | Entrada   |
| <code>fseek()</code>  | Procura um byte especificado na fila            |
| <code>getc()</code>   | Lê um caracter na fila                          |
| <code>putc()</code>   | Grava um caracter na fila                       |
| <code>remove()</code> | Apaga o arquivo                                 |
| <code>rewind()</code> | Reposiciona o ponteiro do Arquivo em seu início |

Para podermos utilizar estas instruções, temos que carregar a biblioteca “stdio.h”, que contém estas rotinas.

Sintaxes:

- fopen()

A função “fopen” tem duas finalidades, a saber:

- abrir uma fila de bytes
- ligar um arquivo em disco àquela fila

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

Exemplo: Abertura de arquivo para gravação:

```
if ((fp = fopen("teste", "w")) = NULL) {  
    puts("Não posso abrir o Arquivo teste.\n");  
    exit(1); /* força o término da execução da rotina */  
}
```

A tabela a seguir apresenta os modos de abertura de arquivo válidos:

| <b>Modo</b> | <b>Significado</b>                                  |
|-------------|---|
| “r”         | Abre Arquivo de Texto para Leitura                  |
| “w”         | Cria Arquivo de Texto para Gravação                 |
| “a”         | Anexa a um Arquivo de Texto                         |
| “rb”        | Abre Arquivo Binário para Leitura                   |
| “wb”        | Cria Arquivo Binário para Gravação                  |
| “ab”        | Anexa a um Arquivo Binário                          |
| “r+”        | Abre Arquivo de Texto para Leitura/Gravação         |
| “w+”        | Cria Arquivo de Texto para Leitura/Gravação         |
| “a+”        | Abre ou Cria Arquivo de Texto para Leitura/Gravação |
| “r+b”       | Abre Arquivo Binário para Leitura/Gravação          |
| “w+b”       | Cria Arquivo Binário para Leitura/Gravação          |
| “a+b”       | Abre ou Cria Arquivo Binário para Leitura/Gravação  |
| “rt”        | Idem a “r”  |
| “wt”        | Idem a “w”  |
| “at”        | Idem a “a”  |
| “r+t”       | Idem a “r+”   |
| “w+t”       | Idem a “w+”   |
| “a+t”       | Idem a “a+”   |

- putc()

Grava caracteres em fila previamente abertos

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

ch é o caracter a ser gravado

fp é o ponteiro devolvido por fopen

- getc()

Ler caracteres em uma fila aberta

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

Exemplo:

```
ch = getc(fp);  
while (ch != EOF)  
    ch = getc(fp);
```

- fclose()

Fechar as filas abertas. Caso o programa seja encerrado sem que as filas sejam fechadas, dados gravados nos buffers podem ser perdidos.

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

- ferror()

Determina se a operação de arquivo produziu um erro. Sua forma geral será:

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

- rewind()

Reinicia o arquivo, equivale ao Reset do Pascal, ou seja apenas movimenta o ponteiro do arquivo para seu início.

Exemplo (Funciona no Classic C) de montagem de um pequeno cadastro de nomes, endereços e salários de funcionários.

```
struct registro {  
    char nome[40];  
    char endereco[40];  
    float valor;  
} matriz[100];
```

```

main()
{
char escolha;
inicia_matriz();
for (;;) {
    escolha = menu();
    switch (escolha) {
        case 'i' :
            inserir();
            break;
        case 'e' :
            exibir();
            break;
        case 'c' :
            carga();
            break;
        case 's' :
            salvar();
            break;
        case 'f' :
            saida();
    }
    }
puts(escolha);
}

saida()
{
cls();
exit();
}

inicia_matriz()
{
int t;
for (t=0;t<100;t++)
    *matriz[t].nome = '\\0';
}

```

```

menu()
{
char s;
cls();
do {
puts("Inserir");
puts("Exibir");
puts("Carregar");
puts("Salvar");
puts("Finaliza");
printf("Digite a 1ª. Letra: ");
scanf("%c",&s);
} while(s != 'i' && s != 'e' && s != 'c' && s != 's' && s != 'f');
return(s);
}

inserir()
{
int i;
for (i=0; i < 100; i++)
if (!*matriz[i].nome) break;
if (i==100) {
puts("Arquivo Cheio!");
return;
}
printf("Nome: "); gets(matriz[i].nome);
printf("End.: "); gets(matriz[i].endereco);
printf("Val.: "); scanf("%f",&matriz[i].valor);
}

exibir()
{
char x;
int t;
cls();
for(t=0;t<100;t++) {
if (*matriz[t].nome) {
printf("%s \n", matriz[t].nome);
printf("%s \n", matriz[t].endereco);
printf("%f \n", matriz[t].valor);
puts(" "); puts("<Enter> para prosseguir!");
x = getchar();
}
else
break;
}
}
}

```

```

salvar()
{
FILE *fp;
int i;
if ((fp=fopen("LISTA.DAT","wb"))==NULL) {
puts("Falhou Abertura! ");
return;
}
for (i=0;i<100;i++)
if (*matriz[i].nome)
if (fwrite(&matriz[i],sizeof(struct registro), 1,fp) != 1)
puts("Falha na Gravacao! ");
fclose(fp);
}

carga()
{
FILE *fp;
int i;
if ((fp=fopen("LISTA.DAT","rb")) == NULL) {
puts("Falha na Abertura do Arquivo!");
return;
}
inicia_matriz();
for (i=0; i < 100; i++)
if (fread(&matriz[i], sizeof(struct registro), 1, fp) != 1) {
if (feof(fp)) {
fclose(fp);
return;
}
else {
puts("Erro de Leitura! ");
fclose(fp);
return;
}
}
}
}

```

## **Laboratório (Aula 20L)**

- 1- Escreva um programa que armazene números primos numa matriz.
- 2- Escreva um nome, num arquivo.
- 3- Retome o exercício 1, porém armazene-o num arquivo.

## Operações com Arquivo (Aula 21T)

Através dos exemplos que serão apresentados a seguir, constataremos a boa interação existente entre a linguagem C e o sistema operacional, antes porém apresentaremos uma nova forma de escrever “main()”.

Vamos supor que desejamos criar um programa que escreva num arquivo cujo nome será fornecido na chamada do programa (Exemplificando: KTOD TESTE <Enter>). Gostaríamos que o DOS criasse o arquivo TESTE guardando o conteúdo digitado durante a execução do programa.

Para isso temos a seguinte sintaxe para “main()”:

```
main(argv, argc)
```

onde

argc - tem o número de argumentos contidos nas linha de comando (necessariamente maior ou igual a um, pois o próprio programa já é considerado um argumento pelo D.O.S.). Argv é um ponteiro que acomodará os caracteres digitados.

Exemplo 1: Programa KTOD, que escreve caracteres num arquivo criado/aberto via D.O.S.

```
#include "stdio.h"
main(argc, argv)
int argc;
char *argv[];
{
FILE *fp;
char ch;
if (argc != 2) {
    printf("Digite o Nome do Arquivo\n");
    exit(1);
}
if ((fp=fopen(argv[1], "w")) == NULL) {
    printf("Arquivo não pode ser aberto\n");
    exit(1);
}
do {
    ch = getchar();
    putc(ch, fp);
} while( ch != '$');
fclose(fp);
}
```

**Exemplo 2: Programa DTOV, que apresenta em vídeo os caracteres digitados via KTOD.**

```
#include "stdio.h"
main(argc,argv)
int argc;
char *argv[];
{
FILE *fp;
char ch;
if (argc != 2) {
printf("Digite o Nome do Arquivo\n");
exit(1);
}
if ((fp=fopen(argv[1],"w")) == NULL) {
printf("Arquivo não pode ser aberto\n");
exit(1);
}
ch = getc(fp);
while (ch != EOF) {
putchar(ch);
ch=getc(fp);
} while( ch != '$');
fclose(fp);
}
```

## **Laboratório (Aula 21L)**

- 1- Uso do Compilador Turbo C. Testar exemplos vistos em teoria.
- 2- Elabore programa que exiba um programa em C armazenado em disco (a semelhança do comando type do DOS).
- 3- O que ocorre com o arquivo DADOS.DAT (que contém dados) após a execução do seguinte programa.

```
main()
{
FILE *fp;
if ((fp=fopen("DADOS.DAT","w"))==Null) {
exit(1);
}
fclose(fp);
}
```

## Operações com Arquivo (Continuação) (Aula 22T)

### Exemplo 1: Programa para copiar Arquivos.

```
#include "stdio.h"
main(argc,argv)
int argc;
char *argv[];
{
FILE *in, *out;
char ch;
if (arg != 3) {
    printf("Digite o Nome dos Arquivos\n");
    exit(1);
}
if ((in=fopen(argv[1],"rb")) == NULL) {
    printf("Arquivo origem não existe\n");
    exit(1);
}
if ((out=fopen(argv[2],"wb")) == NULL) {
    printf("Arquivo destino não existe\n");
    exit(1);
}
while (! feof(in))
    putc(getc(in),out); /* esta é a cópia propriamente dita */
fclose(in);
fclose(out);
}
```

### Outras funções de tratamento de Arquivos.

- getw() e putw()

Idênticas a getc() e putc(), porém trabalhando com inteiros.

- fgets() e fputs()

Sintaxes:

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

fputs() é análoga a puts(), porém escreve em disco.

fgets() lê uma "string" da fila especificada, incluindo caracteres como \n, porém a "string" lida sempre será finalizada com zero ASCII.

- fread() e fwrite()

Permitem que leiamos/gravemos blocos de dados, sua forma geral é a seguinte:

```
int fread(void *buffer, int num_bytes, int cont, FILE *fp);
int fwrite(void *buffer, int num_bytes, int cont, FILE *fp);
```

### Exemplo 2: Leitura de Arquivos contendo números.

```
main()
{
FILE *fp;
float f = 12.23;
if ((fp=fopen("teste","wb")) == NULL) {
    printf("Arquivo não pode ser criado\n");
    exit(1);
}
fwrite(&f,sizeof(float(),1,fp);
fclose(fp);
}
```

- fseek()

### Entrada e saída com acesso aleatório

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

fp - é o ponteiro de arquivo devolvido por fopen().

num\_bytes - é um inteiro longo que representa o número de bytes desde a origem até chegar a posição corrente.

Este comando é normalmente utilizado em arquivos binários.

### Exemplo 3: Leitura de um caracter em um arquivo binário.

```
main()
{
FILE *fp;
if ((fp=fopen("teste","rb")) == NULL) {
    printf("Arquivo não pode ser aberto\n");
    exit(1);
}
fseek(fp,234L,0);          /* L força que seja um inteiro longo */
return getc(fp);         /* lê o caracter 234 */
}
```

#### Exemplo 4: Programa EDL

```
include <stdio.h>
main()
{
int c;
while ((c = getchar()) != EOF)
    putchar(c);
}
```

### Laboratório (Aula 22L)

- 1- Compile e Estude o programa EDL.
- 2- Construa os Programas KTOD e DTOV usando o Turbo C.
- 3- Construa o comando COPIAR, visto na teoria.
- 4- Elabore programa para criação de um dump de arquivo.

### Operações com Arquivo (continuação) (Aulas 23T)

Caso o programador desejasse reproduzir entradas e saídas semelhantes as existentes em teclado e vídeo, ele poderia utilizar os comandos “fprintf” e “fscanf” conforme as sintaxes a seguir:

```
fprintf(fp, "string de controle", lista de argumentos);
fscanf(fp, "string de controle", lista de argumentos);
```

A linguagem C permite a criação de 5 tipos de estruturas particulares, a saber:

- Estrutura, um grupo de variáveis sobre o mesmo nome.
- Campo de Bit, que permite fácil acesso a bits dentro de uma palavra.
- União, que permite que definamos a mesma parte da memória contendo dois ou mais tipos de variáveis (obviamente não usadas concomitantemente).
- Enumeração, que é similar a uma lista de símbolos.
- Tipo Final, onde o programador cria um novo nome para um tipo já existente.

A seguir criaremos um registro que fará uso destes conceitos. Supondo que desejamos um registro contendo Nome, Endereço, etc., teríamos:

```
struct regs {
    char nome[30];
    char rua[40];
    char cidade[20];
    char estado[02];
    unsigned long int cep;
}; /* ponto e vírgula encerrando o registro */
```

### ***Dados***

Além dos tipos básicos de dados que costumamos utilizar em nossos exemplos (int, float, char, etc.), em C dispomos de modificadores que podem mudar a forma de armazenar e de acessar determinada variável.

### ***Modificadores de Acesso e de Armazenamento***

O modificador const, impede que uma variável mude de valor durante a execução do programam conforme segue:

```
const float ver=3.20;
```

neste caso *ver* não poderá ser alterada durante a execução do programa.

O modificador volatile, permite que determinada variável tenha seu valor alterado sem um comando do programa. Este tipo de situação ocorre por exemplo no caso de seu programa acessar o relógio da máquina. Neste caso sua variável de tempo será atualizada constantemente sem que seja necessária qualquer instrução de seu programa.

O modificador auto, raramente usado, serve para declarar variáveis locais. O modificador extern permite que variáveis globais declaradas em módulos compilados separadamente possam ser adequadamente utilizadas.

Exemplo:

#### **Prog. Um**

```
int x,y;
main()
{
    .
    .
    .
}
func1()
{
    x = 12;    y = 3;
}
```

#### **Prog. Dois**

```
extern x,y;
funcDois()
{
    .
    x = y++;
    .
}
```

O modificador `static` permite que uma variável seja permanente dentro de sua própria função. Diferentemente da variável global, esta variável somente é válida em sua própria rotina.

Exemplo:

```
serie()
{
    static int ser_num;
    ser_num = ser_num + 12;
    return(ser_num);
}
```

No exemplo anterior `ser_num` continua a existir entre as chamadas das funções ao invés de existir somente dentro da função *série*, como ocorreria com uma variável local normal.

O modificador `register` permite, caso seja possível, que determinada variável inteira ou caracter, não se localize na memória convencional, fique num registrador disponível (se houver um) da CPU, tornando a execução do programa muito mais veloz. Basta para isso declaramos:

```
register int i;
for(i=0;i<30000;i++)
```

Caso seja possível o contador `i` será processado muito mais rapidamente do que seria sem o modificador `register`.

### ***Conversões de Tipos***

O exemplo a seguir mostra a facilidade que temos para converter tipos em C.

```
main()
{
    int x=70,x1=7;
    char c='a',c1='A';
    float f=23.215;
    printf("x= %d, c= %c, f= %2.2f, x1= %d, c1= %c \n",x,c,f,x1,c1);
    c = x;
    printf("c <--x : c= %c\n",c);
    x = f;
    printf("x <--f : x= %d\n",x);
    f = x1;
    printf("f <--x1: f= %2.2f\n",f);
    f = c1;
    printf("f <--c1: f= %2.2f",f);
}
```

```
x= 70, c= a, f= 23.22, x1= 7, c1= A
c <--x : c= F
x <--f : x= 23
f <--x1: f= 7.00
f <--c1: f= 65.00
```

### ***Ponteiros de funções***

Da mesma forma como uma variável pode ser “apontada” por uma variável tipo ponteiro, uma função também poderá ser. Imaginemos um programa onde desejássemos comparar duas palavras numa função que as recebesse juntamente com um ponteiro que indicasse o sucesso ou não da comparação. No programa principal a função teria que ser declarada como um valor e depois poderia ser apontada por um ponteiro. Particularmente neste caso temos poucas vantagens em usar este raciocínio, porém em alguns casos poderemos usar este método de maneira a melhorarmos nossos programas.

### ***Estruturas***

Em C, uma estrutura é uma coleção de variáveis referenciadas através de um nome definido pelo programador, semelhante ao record da linguagem Pascal.

Exemplo de programa com um pequeno cadastro:

```
struct dados {
char nome[30];
char rua[40];
char cidade[20];
char estado[02];
char cep[09];
};
```

Não devemos confundir tipo com variável, de forma que é errado afirmar-se coisas como “a variável dados recebeu ...”, este tipo (dados) servirá para posteriormente declaramos uma variável como segue:

```
struct dados cliente;
ou ainda
struct dados {
char nome[30];
char rua[40];
char cidade[20];
char estado[02];
char cep[09];
} cliente;
```

## ***Matrizes de Estrutura***

Vamos supor termos definido um registro. Será que nos bastará podermos manipular um registro por vez, ou seria mais interessante termos como manipular uma série deles ao mesmo tempo? Para tornar isto possível devemos ter uma matriz que reflita a estrutura declarada.

Exemplo:

```
struct dados fornecedor[100];
```

Este trecho de código criaria uma matriz (vetor) com 100 elementos organizados pelo formato de dados.

## ***Construção de Programa com acesso a registros***

A seguir iremos contruir um exemplo completo de programa que acessa um tipo especial de arquivo, ou seja, aquele que contém apenas registros. Basicamente a teoria de Banco de Dados é fundamentada nos conceitos que estão sendo exemplificados a seguir.

```
struct registro {
int codigo;
long qtd;
double valor;
char descricao[20];
};

main()
{
struct registro r, *rp;
r.codigo = 999;
r.qtd = 113333L;
r.valor = 23.21e6;
strcpy(r.descricao, "MOTOR DE EXPLOSAO");
printf("Codigo: %d, Qtd: %ld, Valor: %.2f\n", r.codigo, r.qtd, r.valor);
printf("Descricao: %s\n", r.descricao);
rp = &r;
printf("Codigo: %d, Qtd: %ld, Valor: %.2f\n", rp->codigo, rp->qtd, rp->valor);
strcpy(rp->descricao, "MOTOR DE ARRANQUE");
printf("Descricao: %s\n", rp->descricao);
}
```

## Laboratório - Arquivos (Aula 23L)

1. No programa a seguir o carregaremos de um arquivo para uma matriz em memória um arquivo com dados de clientes, que nos permitirá inserir novos nomes, exibir os dados armazenados em memória e novamente armazená-lo em disco.

```
struct registro {
    char nome[40];
    char endereco[40];
    float valor;
} matriz[100];
```

```
main()
{
    char escolha;
    inicia_matriz();
    for (;;) {
        escolha = menu();
        switch (escolha) {
            case 'i' :
                inserir();
                break;
            case 'e' :
                exibir();
                break;
            case 'c' :
                carga();
                break;
            case 's' :
                salvar();
                break;
            case 'f' :
                saida();
        }
        puts(escolha);
    }
}
```

```
saida()
{
    cls();
    exit();
}
```

```

inicia_matriz()
{
int t;
for (t=0;t<100;t++)
    *matriz[t].nome = '\\0';
}

menu()
{
char s;
cls();
do {
    puts("Inserir");
    puts("Exibir");
    puts("Carregar");
    puts("Salvar");
    puts("Finaliza");
    printf("Digite a 1ª. Letra: ");
    scanf("%c",&s);
    } while(s != 'i' && s != 'e' && s != 'c' && s != 's' && s != 'f');
return(s);
}

inserir()
{
int i;
for (i=0; i < 100; i++)
    if (!*matriz[i].nome) break;
    if (i==100) {
        puts("Arquivo Cheio!");
        return;
    }
printf("Nome: "); gets(matriz[i].nome);
printf("End.: "); gets(matriz[i].endereco);
printf("Val.: "); scanf("%f",&matriz[i].valor);
}

```

```

exibir()
{
char x;
int t;
cls();
for(t=0;t<100;t++) {
    if (*matriz[t].nome) {
        printf("%s \n", matriz[t].nome);
        printf("%s \n", matriz[t].endereco);
        printf("%f \n", matriz[t].valor);
        puts(" "); puts("<Enter> para prosseguir!");
        x = getchar();
    }
    else
        break;
}
}

salvar()
{
FILE *fp;
int i;
if ((fp=fopen("LISTA.DAT","wb"))==NULL) {
    puts("Falhou Abertura! ");
    return;
}
for (i=0;i<100;i++)
    if (*matriz[i].nome)
        if (fwrite(&matriz[i],sizeof(struct registro), 1,fp) != 1)
            puts("Falha na Gravacao! ");
fclose(fp);
}

```

```

carga()
{
FILE *fp;
int i;
if ((fp=fopen("LISTA.DAT","rb")) == NULL) {
    puts("Falha na Abertura do Arquivo!");
    return;
}
inicia_matriz();
for (i=0; i < 100; i++)
    if (fread(&matriz[i], sizeof(struct registro), 1, fp) != 1)
        if (feof(fp)) {
            fclose(fp);
            return;
        }
        else {
            puts("Erro de Leitura! ");
            fclose(fp);
            return;
        }
}
}

```

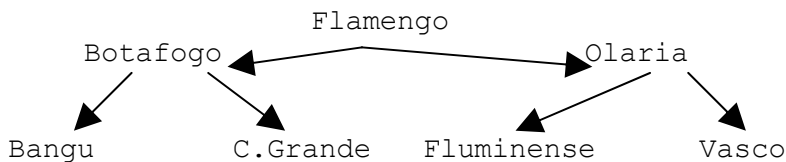
2. Monte Registro que exiba o mês escolhido pelo usuário.

## Arquivos - Conceito de Chaves (Aula 24T)

O programa anterior demonstra como devemos proceder para Carregar um arquivo do disco para a Memória. Permite listarmos e inserirmos dados neste arquivo.

Finalmente permite salvarmos os dados para o Arquivo. Ficou faltando uma rotina de pesquisa, para podermos consultar, excluir e alterar dados, além de uma rotina permitindo listar os dados organizados, por exemplo alfabeticamente.

Montaremos uma rotina que permita a ordenação destes dados usando ponteiros para isto. Uma rotina de pesquisa binária também poderia ser elaborada bastando para isto a criação de rotina que permitisse a navegação dentro do modelo abaixo demonstrado:



Observe que qualquer pesquisa acharia o time pesquisado no máximo em 3 tentativas. Isto ocorre porque a estrutura acima está devidamente balanceada. Suponha porém que as entradas não ocorram na sucessão acima mostrada (Flamengo, Botafogo, Olaria, Bangu, C.Grande, Fluminense e Vasco), mas ocorram na seqüência apresentada a seguir: América, Corinthians, Guarani, Palmeiras, Portuguesa, Santos e São Paulo. Teríamos:

América

--- Corinthians

--- Guarani

--- Palmeiras

--- Portuguesa

--- Santos

--- São Paulo

Desta forma não seria adequado desenvolvermos uma rotina de pesquisa sem antes escrevermos uma rotina de “balanceamento” da estrutura acima, que possibilitam obtermos estruturas no conceito de árvore binária balanceada. Felizmente todos os gerenciadores de arquivo disponíveis, providenciam este tipo de tratamento, desobrigando o programador de aplicação da elaboração deste tipo rotina.

A seguir dois exemplos completos de estruturas manipuladas por ponteiros.

Exemplo 1: Cadastro acessado por ponteiros, com acesso seqüencial (Registro 1,2,3,4,5,67,8,...,100; em 100 Registros).

```
struct addr
{
char nome[30];
char rua[40];
char cidade[18];
char estado[02];
char cep[09];
char lixao[10];
struct addr *next; /* ponteiro da prox entrada */
struct addr *prior; /* ponteiro p/ entrada anterior */
} lista;
struct addr *start; /* primeiro da lista */
struct addr *last; /* ultimo da lista */
struct addr *null;
```

```

main()
{
int choice;
null = (struct addr *)malloc(sizeof(lista));
do {
    choice = menu_select();
    switch(choice) {
        case 1 :
            enter();
            break;
        case 2 :
            delete();
            break;
        case 3 :
            list();
            break;
        case 4 :
            achar();
            break;
        case 5 :
            save();
            break;
        case 6 :
            load();
            break;
        case 7 :
            exit(0);
    }
} while(1);
}

menu_select()
{
char s[80];
int c;
printf("1. Inserir Nomes\n");
printf("2. Retirar Nomes\n");
printf("3. Listar Nomes\n");
printf("4. Procurar Nome\n");
printf("5. Salvar Arquivo\n");
printf("6. Carregar Arquivo\n");
printf("7. Finalizar\n");
do {
    printf("\nSua opcao: ");
    gets(s);
    c = atoi(s);
} while (c < 0 || c > 7);
return c;
}

```

```

enter()
{
struct addr *info;
char *malloc();
do {
    info = (struct addr *)malloc(sizeof(lista));
    if (info == 0){
        printf("\nMemoria Esgotada!");
        return;
    }
    inputs("Entre Nome: ",info->nome,30);
    if (!info->nome[0])
        break;
    inputs("Entre Rua: ",info->rua,40);
    inputs("Entre Cidade: ",info->cidade,18);
    inputs("Entre Estado: ",info->estado,02);
    inputs("Entre CEP: ",info->cep,10);
    if (start!=0) {
        last->next = info;
        info->prior = last;
        last = info;
        last->next = null;
    }
    else {
        start = info;
        start->next = null;
        last = start;
        start->prior = null;
    }
    } while(1);
}

inputs(prompt,s,count)
char *prompt;
char *s;
int count;
{
char p[255];
do {
    printf(prompt);
    gets(p);
    if (strlen(p) > count)
        puts("Muito Grande!");
    } while (strlen(p) > count);
strcpy(s,p);
}

```

```

delete()
{
struct addr *info, *find();
char s[255];
int volta;
inputs("Entre Nome: ",s,30);
info = find(s);
if (info) {
    if (start == info) {
        start = info->next;
        start->prior = null;
    }
    else {
        info->prior->next = info->next;
        if (info != last)
            info->next->prior = info->prior;
        else
            last = info->prior;
    }
    free(info);
}
}

struct addr *find(nome)
char *nome;
{
struct addr *info;
info = start;
while (info && info != null) {
    if (!strcmp(nome,info->nome))
        return info;
    info = info->next;
}
return null;
}

list()
{
int t;
char lixo;
struct addr *info;
info = start;
while (info && info != null) {
    display(info);
    scanf("%c",&lixo);
    info = info->next;
}
printf("\n\n");
}

```

```

achar()
{
struct addr *info,*find();
char s[255];
inputs("Entre Nome: ",s,30);
info = find(s);
if (info != null)
    display(info);
else
    printf("Nao Encontrado!\n");
}

display(info)
struct addr *info;
{
printf("%s\n",info->nome);
printf("%s\n",info->rua);
printf("%s\n",info->cidade);
printf("%s\n",info->estado);
printf("%s\n",info->cep);
printf("\n\n");
}

save()
{
int t,size;
struct addr *info;
char *p;
FILE *fp;
if ((fp = fopen("LISTA.DAT","w")) == 0) {
    puts("Falhou a Abertura!");
    exit(0);
}
printf("Salvando Arquivo\n");
size = sizeof(lista);
info = start;
while (info && info != null) {
    p = (char *)info; /* convesao p/ ponteiro de caracter */
    for (t=0;t<size-1;++t)
        putc(*p++,fp); /* salva um byte */
    info = info->next; /* proximo */
}
putc EOF,fp); /* marcando fim de arquivo ... opcional */
fclose(fp);
}

```

```

load()
{
int t,size;
struct addr *info,*temp;
char *p,*malloc();
FILE *fp;
if ((fp = fopen("LISTA.DAT","r")) == 0) {
    puts("Falhou Abertura");
    exit(0);
}
printf("Carregando Arquivo\n");
size = sizeof(lista);
start = (struct addr *)malloc(size);
if (!start) {
    puts("Acabou Memoria! ");
    return;
}
info = start;
p = (char *)info; /* ponteiro para caracter */
while ((*p++ = getc(fp)) != EOF) {
    for (t=0;t<size-2;++t)
        *p++ = getc(fp); /* carrega byte a byte */
    info->next = (struct addr *)malloc(size); /* aloca mais memoria */
    if (!info->next) {
        printf("Memoria Esgotada!\n");
        return;
    }
    info->prior = temp;
    temp = info;
    info = info->next;
    p = (char *)info;
    if (info == null) break;
}
free(temp->next);
temp->next = null;
last = temp;
start->prior = null;
fclose(fp);
}

```

**Exemplo 2: Registro ordenados alfabeticamente através de ponteiros:**

```
struct addr
{
char nome[30];
char rua[40];
char cidade[18];
char estado[02];
char cep[09];
char vago[10];
struct addr *next;      /* ponteiro da prox entrada */
struct addr *prior;    /* ponteiro p/ entrada anterior */
} lista;
struct addr *start;    /* primeiro da lista */
struct addr *last;    /* ultimo da lista */
struct addr *null;    /* endereco inicial */
struct addr *entra;    /* entrada atual */
char mostra = 'S';
```

```

main()
{
int choice;
cls;
printf("Exibir Situacao (Enter Exibe - n nao): "); scanf("%c",&mostra);
null = (struct addr *)malloc(sizeof(lista));
do {
choice = menu_select();
switch(choice) {
case 1 :
enter();
break;
case 2 :
delete();
break;
case 3 :
list();
break;
case 4 :
achar();
break;
case 5 :
save();
break;
case 6 :
load();
break;
case 7 :
exit(0);
}
} while(1);
}

menu_select()
{
char s[80];
int c;
printf("1. Inserir Nomes\n");
printf("2. Retirar Nomes\n");
printf("3. Listar Nomes\n");
printf("4. Procurar Nome\n");
printf("5. Salvar Arquivo\n");
printf("6. Carregar Arquivo\n");
printf("7. Finalizar\n");
do {
printf("\nSua opcao: ");
gets(s);
c = atoi(s);
} while (c < 0 || c > 7);
return c;
}

enter()
{
struct addr *info;
struct addr *old;
int situacao;
char lixo;

```

```

char *malloc();
do {
    last = (struct addr *)malloc(sizeof(lista));
    if(last == 0) {
        printf("\nMemoria Esgotada!");
        return;
    }
    inputs("Entre Nome: ",last->nome,30);
    if (!last->nome[0])
        break;
    inputs("Entre Rua: ",last->rua,40);
    inputs("Entre Cidade: ",last->cidade,18);
    inputs("Entre Estado: ",last->estado,02);
    inputs("Entre CEP: ",last->cep,10);
    if(start!=0) {
        info = start;
        old = start;
        situacao = 0;
        while (situacao==0) {
            if(mostra!='n') {
                printf("Posicionado em %s <Enter>\n",info->nome);
                scanf("%c",&lixo);
            }
            if(*info->nome>*last->nome) {
                if(mostra != 'n') {
                    printf("Posicionou (1) %s\n",info->nome);
                    situacao = 1;
                    break;
                }
            }
            if(info==null) {
                if (mostra != 'n')
                    printf("Fim de Arquivo (2) %s\n",last->nome);
                situacao = 2;
                break;
            }
        }
        old = info;
        info = info->next;
        if(info->next==null) {
            if(mostra != 'n') {
                printf("Aponta para Fim de Arquivo (3) %s\n",info->nome);
                situacao = 3;
                break;
            }
        }
    }
    if(situacao==1) {
        if(info==start) {
            last->next = info;
            info->prior = last;
            last->prior = null;
            start = last;
        }
    }
    else {
        last->next = info;
        info->prior = last;
        last->prior = old;
    }
}

```

```

    old->next = last;
    }
}
if(situacao==2) {
    last->next = null;
    old->next = last;
    last->prior = old;
}
if(situacao==3) {
    last->next = null;
    info->next = last;
    last->prior = info;
}
}
else {
    start = last;
    start->next = null;
    start->prior = null;
}
}while(1);
}

inputs(prompt,s,count)
char *prompt;
char *s;
int count;
{
char p[255];
do {
    printf(prompt);
    gets(p);
    if (strlen(p) > count)
        puts("Muito Grande!");
    } while (strlen(p) > count);
strcpy(s,p);
}

```

```

delete()
{
struct addr *info, *find();
char s[255];
int volta;
inputs("Entre Nome: ",s,30);
info = find(s);
if (info) {
    if (start == info) {
        start = info->next;
        start->prior = null;
    }
    else {
        info->prior->next = info->next;
        if (info != last)
            info->next->prior = info->prior;
        else
            last = info->prior;
    }
    free(info);
}
}

achar()
{
struct addr *info, *find();
char s[255];
int volta;
inputs("Entre Nome: ",s,30);
info = find(s);
if (info==null)
    printf("Nao Encontrado!\n");
else
    display(info);
}

struct addr *find(nome)
char *nome;
{
struct addr *info;
info = start;
while (info && info != null) {
    if (!strcmp(nome,info->nome))
        return info;
    info = info->next;
}
return null;
}

```

```

list()
{
int t;
char lixo;
struct addr *info;
info = start;
while (info && info != null) {
    display(info);
    scanf("%c",&lixo);
    info = info->next;
}
printf("\n\n");
}

display(info)
struct addr *info;
{
printf("\n\n*** E X I B I N D O   R E G I S T R O   ***\n");
printf("%s\n",info->nome);
printf("%s\n",info->rua);
printf("%s\n",info->cidade);
printf("%s\n",info->estado);
printf("%s\n",info->cep);
printf("\n\n");
}

save()
{
int t,size;
struct addr *info;
char *p;
FILE *fp;
if ((fp = fopen("ALFA.DAT","w")) == 0) {
    puts("Falhou a Abertura!");
    exit(0);
}
printf("Salvando Arquivo\n");
size = sizeof(lista);
info = start;
while (info && info != null) {
    p = (char *)info; /* convesao p/ ponteiro de caracter */
    for (t=0;t<size-1;++t)
        putchar(*p++,fp); /* salva um byte */
    if(info->next==null)
        break;
    info = info->next; /* proximo */
}
putchar(EOF,fp); /* marcando fim de arquivo ... opcional */
fclose(fp);
}

```

```

load()
{
int t,size;
struct addr *info,*temp;
char *p,*malloc();
FILE *fp;
if ((fp = fopen("ALFA.DAT","r")) == 0) {
puts("Falhou Abertura");
exit(0);
}
printf("Carregando Arquivo\n");
size = sizeof(lista);
start = (struct addr *)malloc(size);
if (!start) {
puts("Acabou Memoria! ");
return;
}
info = start;
p = (char *)info; /* ponteiro para caracter */
while ((*p++ = getc(fp)) != EOF) {
for (t=0;t<size-2;++t)
*p++ = getc(fp); /* carrega byte a byte */
info->next = (struct addr *)malloc(size); /* aloca mais memoria */
if (!info->next) {
printf("Memoria Esgotada!\n");
return;
}
info->prior = temp;
temp = info;
info = info->next;
if(info->next==null)
break;
p = (char *)info;
if (info == null)
break;
}
free(temp->next);
temp->next = null;
last = temp;
start->prior = null;
fclose(fp);
}

```

## Teste de Cadastro (Aula 24L)

1. Digitação do Exemplo Acima.  
 Critique a forma de recuperação dos dados previamente digitados.
2. Montar programa que escreva Dados num arquivo.

3. Montar programa a semelhança do Comando Type.
4. Montar programa que escreva um texto num arquivo.
5. Montar programa que leia texto previamente armazenado num arquivo.

## Montagem de Cadastro (Aula 25T)

Analise o programa Cadastro e monte uma pequena agenda com nomes e telefones.

Sugestões:

- Defina claramente as opções do programa (exemplo: Adicionar, Buscar, etc).
- Defina as funções a serem criadas (exemplo: add\_nome, find\_nome, etc).
- Crie o Algoritmo do Programa e de cada Rotina.
- Divida a Tarefa entre os componentes do grupo.
- Crie um pseudo código e depois (depois mesmo) digite o programa em C.

Lembramos que o Turbo C possui mais recursos que o Classic C, porém erros neste compilador são potencialmente muito mais complicados de serem encontrados e corrigidos do que falhas similares no Classic C.

### *O pré-processador*

O pré-processador é executado antes da compilação propriamente dita. Ele atua como se fosse um processador de textos “inteligente”, fazendo substituições de textos, inclusões de arquivos e até processamento condicional.

Todas as diretivas para o processador devem ser iniciadas com o sinal #, que deverá estar no início da linha.

### *#define*

Sintaxe:

```
#define <identificador> <definição>
```

Exemplo:

```
#define PI 3.14159
#define quadrado(a) (a) * (a)
```

Assim se no programa existir:

```
y = quadrado(x-2);
```

a expressão será interpretada como  $y = (x-2) * (x-2)$ ;

### ***#include***

Sintaxe:

```
#include <arquivo>
```

Exemplo:

```
#include "stdio.h"
```

Devemos observar que existem poucos comandos na linguagem C, e que a maioria das funções deve ser escrita em C. Isto realmente ocorre, porém os produtores dos compiladores C fornecem todas estas funções previamente escritas para seus

compiladores, necessitando apenas serem “incluídas” nos programas pelo programador conforme for necessário.

Nada impede que desenvolvamos novas bibliotecas específicas a nossas necessidades que poderão facilmente ser transportadas entre ambientes (D.O.S. e Unix por exemplo).

### ***#undef***

Sintaxe:

```
#undef <identificador>
```

Faz com que o #define atribuído a algum identificador seja cancelado.

### ***#if, #else e #endif***

A diretiva if <expressão> faz com que, caso a expressão seja diferente de zero, o código entre a linha do #if até o #else seja incluídos, senão será incluído o trecho entre #else e o #endif.

Exemplo:

```
#define teste 100
#if teste > 50

#else
#endif
```

## **#ifdef e #ifndef**

`ifdef <identificador>` permite verificar se certo identificador foi definido.

Se existir é retornado Verdadeiro.

`ifndef <identificador>` permite verificar se certo identificador foi definido.

Se não existir é retornado Verdadeiro.

```
#define pi 3.14159
main()

#ifndef eu
printf("'eu' não foi definido.\n");
#endif

#ifndef pi
printf("Nunca será executado, pois pi existe!\n");
#endif
```

## ***Unões***

Imaginemos um caso onde tenhamos um programa muito grande e que precisemos executá-lo sem que possamos fazer qualquer redução em suas estruturas. Caso pudermos usar o mesmo local de memória para armazenarmos duas variáveis distintas, resolveremos nosso problema.

Exemplo:

```
main()
{
union data
{
int dia;
int mes;
int ano;
};
union data d;
d.dia = 30;
printf("%d\n", d.dia);
d.mes = 3;
printf("%d\n", d.mes);
printf("%d\n", sizeof(d));
}
```

## **Montagem do Cadastro (Aula 25L)**

Digitação do Exemplo Acima.

Critique a forma de recuperação dos dados previamente digitados.

## **Teoria/Laboratório - Integrações (Aula 26T)**

### ***Integração C com Clipper***

Devido ao fato da linguagem Clipper ter sido escrita em C, a integração das duas linguagens é relativamente simples, porém devemos observar as seguintes condições:

1. As variáveis C tem uma tipologia mais rica (não suportada pelo Clipper), de forma que a passagem de variáveis entre Clipper e C tem que levar em consideração esta característica. Felizmente a Nantucket (produtora original do Clipper) providenciou os fontes em C, `extend.h` e `nandef.h`, que devem ser compilados juntamente com as rotinas em C a serem utilizadas pelo Clipper.
2. A Rotina deve ser declarada sem utilizar a palavra `main()` e a função dever ser precedida pela palavra `CLIPPER` (Exemplo: `CLIPPER apaga()`).
3. A biblioteca `LLIBCR.LIB` (da Microsoft), deve ser linkeditada juntamente com as bibliotecas `CLIPPER`, `EXTEND`.

4. Para rotinas profissionais, sugerimos a utilização do linkeditor BLINKER (da BLINK), ou o RTLINK (da Nantucket), que criam executáveis mais eficientes que àqueles criados pelo TLINK (Borland), LINK (Microsoft) ou PLINK (Phoenix), embora todos estes possam ser utilizados.

5. Devido ao fato do Clipper ter sido escrito em C, mais especificamente no Microsoft C, preferencialmente este deve ser o compilador a gerar o código objeto para ser posteriormente ligado ao código Clipper. Outro compilador poderá eventualmente gerar código que resulte em alguma incompatibilidade imprevista.

### Exemplo: Inverter Cores de Setor da Tela Programa em Clipper

```
TelaLs = 04
TelaCe = 14
TelaLi = 23
TelaCd = 67
Tela = SaveScreen(TelaLs,TelaCe,TelaLi,TelaCd)
Atributo = "RB/N"
C_CHCOLOR(@Tela,Atributo)
Return
```

### *Função em C*

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <bios.h>
#include <mem.h>
#include <string.h>
#include <nandef.h>
#include <extend.h>
/* SO ACEITA LETRAS MAIUSCULA */
#define B 0x01
#define G 0x02
#define R 0x04
#define W 0x07
#define BB 0x10
#define BG 0x20
#define BR 0x40
#define BW 0x70
#define I 0x08
#define BLI 0x80
#define MAXATRIB 20
#define VIDEO 0xB800
```

```

CLIPPER c_chcolor()
{
register int i,j;
unsigned int len_tel;
char      *tela,atr ;
if ( PCOUNT != 2 || !ISCHAR(1) || !ISCHAR(2) ) {
printf( "parametro incorreto em c_chcolor \n" ) ;
exit( 2 ) ;
}
tela      = _parc( 1 );
len_tel   = _parcsiz( 1 );
atr       = _atr_color( _parc(2) , _parclen(2) );
for ( i=1 ; i<len_tel ; i=i+2 )
tela[ i ] = atr ;
retni(0) ;
return   ;
}

```

```

/* str_atr - 'WNBGR*+/WNBGR*+' retorna o caracter atributo da cor */
int _atr_color( char *str_atr , int len_atr )
{
register int i ;
char atr=0x00 ;
int charpos ;

if ( ( charpos = _charpos( str_atr, '/' , len_atr ) ) != 0 ) {
for ( i=1+charpos ; i < len_atr && str_atr[ i ] != ',' ; i++ ) {
switch(str_atr[i]) {
case 'B' : atr = atr|BB ; break ;
case 'G' : atr = atr|BG ; break ;
case 'R' : atr = atr|BR ; break ;
case 'W' : atr = atr|BW ; break ;
case '*' : atr = atr|BLI ; break ;
case '+' : atr = atr|I ;
}
}
}

for ( i = 0; i < len_atr && str_atr[i] != '/' && str_atr[i] != ','; i++){
switch(str_atr[i]) {
case 'B' : atr = atr|B ; break ;
case 'G' : atr = atr|G ; break ;
case 'R' : atr = atr|R ; break ;
case 'W' : atr = atr|W ; break ;
case '*' : atr = atr|BLI ; break ;
case '+' : atr = atr|I ;
}
}
return( atr );
}

/* retorna a posicao do caracter na string */
int _charpos( char *string , char charac , int len_str )
{
register int charpos ;
for( charpos = 0 ; charpos < len_str ; charpos++ )
if( string[charpos] == charac )
return( charpos ) ;
return( 0 ) ;
}

```

### ***Integração C com Btrieve***

Gerenciadores de Arquivo como o Btrieve (Novell) ou Gerenciadores de Bancos de Dados como o Oracle (Oracle), podem ser acessados por programas escritos em C, desde que estes recebam/passem parâmetros da forma especificada pelo gerenciador de Arquivo/Banco de Dados. De forma a exemplificar a utilização de C integrado a gerenciadores, segue trecho de programa em C, abrindo um Arquivo de Dados (Btrieve) e escrevendo no mesmo um registro. Obviamente, é necessário profundo conhecimento em Gerenciadores, para a elaboração de programas utilizando-se deste recurso.

Exemplo: Busca de uma Chave (Função 9 do Btrieve)

```
strcpy(chave,"AMARELO"); /* string */
funcao = 9; /* inteiro */
buffer = 128; /* inteiro */
canal = fp; /* ponteiro de arquivo */
/* retorno é uma variável previamente criada para acomodar o tamanho de
um registro */
status = 0; /* inteiro */
call btrv(funcao,canal,buffer,chave,retorno,status);
```

Status deverá assumir valor 4 se a chave não existir, 0 se existir e um número qualquer entre 1 e 99 indicando que tipo de erro ocorreu.

### ***Biblioteca DBE e ODBC***

A recente tendência de mercado em voltar a utilizar linguagens de 3ª geração (principalmente C, BASIC, COBOL e Pascal), que não possuem intrinsecamente gerenciadores de Base de Dados como as linguagens xBase (Clipper, dBase, Fox, etc), criou novamente mercado para os gerenciadores de arquivo como os que existiam até meados da década de 80 (MicroB, KISSBasic, Btrieve).

De fato o Btrieve, por ser o mais utilizado e também por pertencer a uma empresa bastante poderosa (Novell, que adquiriu a SoftCraft produtora original deste programa), foi o único gerenciador que “sobreviveu” a era xBase. Com o retorno das linguagens acima citadas e com o desinteresse da Novell em adequar o Btrieve aos novos dialetos surgidos das linguagens de 3ª geração, criou-se um mercado promissor que Borland (DBE) e Microsoft (ODBC) estão agora disputando. Em favor da Microsoft temos o fato de seu pacote interagir muito bem com o Visual BASIC e o Visual C, pacotes de grande utilização no mercado americano (principalmente o primeiro). Os pontos positivos da Borland residem no suporte de seu gerenciador ao acesso por registro, além do acesso via SQL (Structured Query Language), que é a linguagem padrão (sic) de acesso aos Bancos de Dados mais utilizados (Oracle, SyBase, Informix e Ingres), além da perfeita interação de seu programa com o Borland C++, a versão de linguagem C mais utilizada para desenvolvimento nos Estados Unidos.

## **Projeto em C (Aulas 26L)**

1. Elabore sistema para Controle de Estoque de Peças, contendo arquivo de produtos e de movimentos.
2. Elabore Jogo Senha. O computador deverá permitir que dois humanos joguem entre si, atuando apenas como validador do resultado e indicando quantos dígitos estão no local correto/próximo (bom/ótimo).

## **APÊNDICE I - BIBLIOGRAFIA E PROGRAMA DO CURSO**

### **Bibliografia:**

#### *Básica*

- Diagramas de Blocos - Dirceu D. Salvetti  
*EDUSP*

- Turbo C - Herbert Schildt

Makron Books

#### *Complementares*

- Linguagem C - Thelmo J.M. Mesquita  
*Érica*
- Linguagem Algorítmica - Dirceu D. Salvetti

*EDUSP*

- Programação Sistemática - Niklaus Wirth

Campus

## **Programa do Curso:**

Apresentação (Aula 01T)

Linguagem C (Aula 02T) - HELLO.C

C- Uma Visão Geral e Instruções de Entrada e Saída (Aula 03T)

Instruções de Entrada e Saída (Aula 04T) - QUADR.C

Tomada de Decisão (Aula 05T) - MAIOR.C

Tipos de Dados (Aula 06T) - FAT1.C até FAT5.C

Variáveis e Operadores (Aula 07T) - DOSTIME.C, TIPOS.C, FAHRCELC.C, REPRES.C, INC1.C e INC2.C

Tomadas de Decisão - Parte II (Aula 08T) - SWITCH.C

Loops (Aula 09T) - PRIMO1.C até PRIMO3.C e FOR.C

Comandos Desestruturadores (Aula 10T) - ALO\_VEJA.C

Matrizes (Aula 11T)

Ordenação (Aula 12T) - ORDENA.C

Ponteiros - Apresentação (Aula 13T)

Ponteiros - Conceitos de Endereços (Aula 14T) - PONT\_7.C e PONT\_1.C

Ponteiros - Conceitos Avançados (Aula 15T) - PONT\_3.C, PONT\_4.C, PONT\_2.C, PONT\_5.C, INV1.C e INV2.C

Ponteiros - Pilhas (Aula 16T) - PILHA.C e PILHA2.C

Ponteiros - Conceitos Complementares (Aula 17T)

Ponteiros x Matrizes e Entradas e Saídas - Arquivos (Aula 18T) - PONT\_8.C, PONT\_9.C e PONT\_6.C

Operadores e Funções String (Aula 19T) - MAIUSC.C, MES.C e NUMEROLO.C

Entradas e Saídas em Dispositivos (Aulas 20T) - IMP.C e ATOI.C

Operações com Arquivo (Aula 21T) - REGS.C e REGS2.C

Operações com Arquivo - Continuação (Aula 22T) - CAD1.C, ESCARQ.C, ESCARQ2.C e LESEQ.C

Operações com Arquivo - Continuação (Aula 23T) - TYPE.C e TYPE2.C

Operações com Arquivo - Conceitos de Chaves (Aula 23T)

Teoria - Integrações Clipper e Btrieve (Aulas 24T)

Montagem de Cadastro (Aula 25T)

Integrações (Aula 26T)

***Marcas Registradas citadas nesta apostila***

Os produtos citados anteriormente, pertencem aos seus respectivos fabricantes,

a saber:

Turbo C, Turbo Pascal, Borland C++, DBE - Borland Internacional Co.

Visual BASIC, Visual C, ODBC - Microsoft Co.

Fox- Fox Software Inc. (Microsoft)

Clipper- Nantucket Co. (Computer Associates)

Ingress- ASK (Computer Associates)

Oracle- Oracle Inc.

Btrieve- SoftCraft System, Inc. (Novell)

SyBase- SyBase Co.

Informix- Informix Co.

dBase- Ashton Tate, Inc. (Borland)

## Apêndice II - Lista de Exercícios Bimestrais

### *Lista de Exercícios de TAPD - 1º. Bimestre*

- 1- Elaborar programa que construa os “n primeiros números da série de Fibonacci (0 1 1 2 3 5 8 ...).
- 2- Elaborar programa que imprima os números primos entre os números “a” e “b”, fornecidos pelo usuário.
- 3- Crie programa que calcule a somatória de uma seqüência de números dados.
- 4- Traduzir o algoritmo a seguir para C, simular e dizer qual problema foi resolvido.

```
declare i,lim,a como inteiras
inicio
iniciar i com 1, ler a, ler lim
enquanto i < lim faça
    i recebe i + 1
    a recebe a - 1
    imprima a
final
```

- 5- Certo usuário resolveu testar se um número inteiro qualquer “n” era divisível (resto = 0) pelos seguintes números: 3 e 5. Simule para 15 e 19.
- 6- Traduzir o algoritmo a seguir para C, simular e dizer qual problema foi resolvido.

```
declare i,num como inteiras
inicio
solicitar e ler num
iniciar i com 1
enquanto num > 0 faça
    i recebe num * i
    num recebe num - 1
final
```

- 7- Elabore um programa que totalize uma série de “n” números inteiros informados pelo teclado e imprima sua média.

8- Codifique o algoritmo abaixo em C.

```
declare i,j,k
leia k
se k > 100
  para i de 1 até 4 faça
    leia j
    se j > 7
      imprima j
    senão
      leia j
      se j = 10
        imprima k
      senão
        imprima j
```

Simule para: 200 3 5 6 8 e também para: 20 10

9- Codifique o algoritmo abaixo em C.

```
declare i,j,k,r
leia j,k
para i de j até k faça
  r = resto(i/2)
  if r = 0 então
    imprima i
```

*Simule para: 3 12*

10- Elabore programa que apresente a série abaixo apresentada:

0 4 7 10 13 16 ....

O usuário deve informar o total de número a serem apresentados.

Elabore Algoritmos, Programas em C e Simulações.

## Lista de Exercícios de TAPD- 2º. Bimestre

- 1- Ordene de forma crescente o conteúdo da matriz “m” com 10 elementos.
- 2- Dado o programa a seguir

```
main()
{
  int i,j,l;
  printf("Entre com 2 números: ");
  scanf("%d",&i); scanf("%d",&j);
  if (i > j)
    l = i + j;
  else
    l = i;
  printf("%d",l);
}
```

O que faz este programa

Simule para  $i = 2, j = 3$  e para  $j = 5$  e  $i = 2$

Caso trocássemos “scanf(“%d”,&i)” por “scanf(i)”, o que aconteceria.

- 3- Traduzir o algoritmo a seguir para C, simular e dizer se este programa imprime sempre o menor número. Justifique ou corrija o programa, caso este esteja errado.

```
Leia a, b, c
Se a < b então
  d <-- a
Se a < c então
  d <-- a
Se b < c então
  d <-- b
imprima d
```

- 4- Elabore um programa que imprima valores de acordo com a tabela abaixo, sem usar o comando if.

|                     |         |             |           |
|---------------------|---------|-------------|-----------|
| $i < 20$            | --> 3   | $i = 20$    | --> 4 5 6 |
| $i > 20$ e $i < 50$ | --> 5 6 | $i = 50$    | --> 6     |
| $i > 50$ e $i < 75$ | --> 7 8 | $i \geq 75$ | --> 9     |
| $i = 90$            | --> 8   |             |           |

Sugestão: Use switch

- 5- Dado o Programa

```
main()
{
  int x,y;
  for (x = 1, y = 2; x + y < 20; x = y++ + x + 1)
    printf("%d",x+y);
}
```

- a- Simule a execução deste programa, o que será impresso
- b- O que ocorreria caso fossem feitas as alterações abaixo,

```
for (x = 1, y = 2; x + y < 20; x = ++y + x + 1)
    printf("%d", x+y);
    y = 2;
```

**Simule.**

c- Rescreva este programa para que o usuário possa decidir, a cada impressão, se deseja ou não prosseguir a execução.

## 6- Traduza para a linguagem C

```
principal
declarar i=0,j=1,m,k como inteiras
    imprima i,j
para k de 1 até 20 faça
    m = i + j
    imprima m
    i = j
    j = m
```

**Simule**

## 7- Dado o Programa a seguir:

```
main()
{
int y,a;
printf("Digite um valor: ");
scanf("%d",&a);
for(y=a;y<=100;y++);
    printf("%d",y);
for(y=100;y>0;y--) {
    printf("%d\n",y);
a=y+5;
}
printf("%d",a);
}
```

**Simule, o que será impresso?**

O programador cometeu um erro lógico. Identifique-o, corrigindo o programa.

Rescreva o programa, sem usar o comando "for".

## 8- Traduza para a linguagem C

```
principal
declarar i,j,k,t como inteiras
a como matriz de inteiros com 11 elementos
leia i,j
para k de i até j faça
  imprima linha em branco
para t de 0 até 10 faça
  a[t] = k * t
imprima k,t,a[t]
```

O que faz este programa?

## 9- Dado o algoritmo a seguir

```
Programa X
i,n,j <-- Inteiro
A <-- Matriz de Inteiros
Limpar a Tela
Leia n (Limite da Série)
j <-- 1
Para i que varia de 1 até n faça
  Se resto (i/4) = 0 então
    A[j] <-- i
    j <-- j + 1
Para i que varia de 1 até j faça
  Imprima A[j]
```

- a- O que faz este programa .
- b- Traduzir para C.
- c- Simulação. Considerar n=17.

10- Calcule e apresente os “n” primeiros números primos solicitados pelo usuário.

Elabore Algoritmos, Programas em C e Simulações.

*Lista de Exercícios de TAPD - 3º. Bimestre*

- 1- Elabore programa que atribua um valor constante em uma variável “b” para uma variável “a”, ambas inteiras, obrigatoriamente através de ponteiros.
- 2- Crie uma função que acumule em uma variável “x”, passada como parâmetro pelo programa principal (seu endereço naturalmente) um valor digitado pelo operador na própria função.
- 3- Crie programa que acumule em uma variável “x” passada do programa principal, recursivamente, 5 vezes o número 5.
- 4- Elabore função que troque os valores das variáveis “a” e “b” passadas pelo programa principal.
- 5- Elabore programa que armazene números em uma pilha e recupere-os a ordem do usuário.

6- Considere o programa abaixo:

```
main()
char c, *pc, d;
{
c = 'A';
*pc = c;
d = &pc;
printf("%c", d);
}
```

O programador deseja transferir o valor da variável “c” para “d” usando ponteiros.

O que ocorre quando executado o programa acima, simule.

Programa correto, se necessário.

Comente a utilização do ponteiro neste programa, na versão acima.

7- Considere o programa abaixo:

```
main()
{
char a = '1', b = '9';
Troca(&a, &b);
printf("%c e %c", a, b);
}
```

Escrever “Troca()”, de forma que sejam impressos “9 e 1”

8- Considere o programa a seguir:

```
main()
int i, k, *pi, *pk;
{
i = 2; k = 0;
puts("Qual será o valor de k? ");
*pk = i;
pk = &k;
printf("%d", k);
}
```

Simule a execução deste programa

Qual é o valor de k após a execução. Justifique.

Se trocarmos “\*pk = i” por “pk = &i” o que ocorreria.

Caso seja necessário, corrija este programa para que no seu final “k” ter o valor de “i”.

9- Sabendo que o programa principal lê um número, escreva função que mostre na tela a tabuada deste número digitado e retorne ao programa principal o valor da somatória das parcelas da tabuada.

10- Monte programa que acesse números inteiros previamente armazenados em uma pilha, porém propositadamente cause um estouro inferior da pilha. Qual será o resultado apresentado?

Elabore Algoritmos, Programas em C e Simulações.

*Lista de Exercícios de TAPD - 4º. Bimestre*

- 1- Monte programa que transfira os dados do arquivo DADO.DAT, não necessariamente existente, para o arquivo DADO.BAK.
- 2- Elabore programa que crie um arquivo chamado DADO.DAT e escreva um registro neste arquivo.
- 3- Dado um Nome qualquer, inverta-o.
- 4- Crie programa que exiba os dados do arquivo DADO.DAT, caso este arquivo exista. Se não existir envie mensagem “Arquivo Inexistente” ao operador.
- 5- Crie programa que exiba os dados do arquivo DADO.DAT, caso este arquivo exista. Se não existir envie mensagem “Arquivo Inexistente” ao operador.
- 6- Elabore Programa que crie e armazene registros em arquivo a ser acessado randômicamente.
- 7- Crie Programa que altere registros criados previamente.
- 8- Elabore Programa que liste em tela registros criados previamente.
- 9- Crie Lista de Registros usando ponteiros como apontadores.
- 10- Crie Registros ordenados de forma alfabética com ponteiros.

Elabore Algoritmos, Programas em C e Simulações.

## Apêndice III - Respostas de Alguns Exercícios Propostos

### Aula 1L - Exercício 1

```
main()
{
printf("Hello, world!\n");
}
```

### Aula 7L - Exercício 6

```
main()
{
char ac[80];
int format, n;
char *acpoint;
for(n = 1; n < 8; n++){
    format = n;
    dostime(ac, format);
    printf("Acessando a funcao 'dostime' no formato,;
        %d que representa: ",format);
    puts(ac);
}
}
```

### Aula 8L - Exercício 6

```
main()
{
char c;
printf("Digite 1, 2 ou 3 ou outro caracter:");
c = getch();
switch © {
    case '1':
        puts("\nVoce Escolheu 1");
    case '2':
        puts("\nVoce Escolheu 2");
        break;
    case '3':
        puts("\nVoce Escolheu 3");
        break;
    default:
        puts("\nVoce Escolheu algo diferente de 1,2 ou 3");
}
puts("Fim!");
}
```

### Aula 9L - Exercício 3

```
main()
{
char ac[80];
int j,i,ini,fim,n,nao;
double r;
cls();
printf("Digite extremo inferior: "); scanf("%d",&ini);
printf("\nDigite extremo superior: ");scanf("%d",&fim);
dostime(ac, 2);
puts(ac);
for(i=ini;i<=fim;i++) {
    nao = 1;
    if (i % 2 == 0)
        nao = 0;
    else {
        j = 3;
        r = i;
        r = sqrt( r );
        while (j<=r) {
            if (i % j == 0) {
                nao = 0;
                break;
            }
            j =j + 2;
        }
    }
    if (nao || i == 2)
        printf("%d ",i);
}
printf("\n");
dostime(ac, 2);
puts(ac);
}
```

## Aula 23L - Exercício 2

```
struct date {
int day;
int month;
char mon_name[12];
};
main()
{
static struct date months[3] = { {5,1,"JANEIRO"}, {4,2,"FEVEREIRO"},
    {10,3,"MARCO"} };
int a;
struct date *p_months;
p_months=&months[1];    /* observe que o 1°. esta' armazenado    */
                        /* na posicao 0 */
                        /* da matriz, sempre usada em C. */
                        /* Portanto Fevereiro sera' o 2°. mes */
printf("Nome do Mes No. %d e': %s.",;
p_months->month, p_months->mon_name);
}
```

## Aula 24L - Exercício 2

```
main()
{
FILE *fo;
char c, dest[25];
int i;
char *cp;
double d;
printf("Arquivo a ser Criado: ");
gets(dest);
if((fo = fopen(dest,"w")) == NULL) {
    printf("Nao posso criar %s\n",dest);
    exit();
}
c = 'A';
i = 999;
cp = "String criada para exemplo.";
d = 123.456;
fprintf(fo,"%c %d %s %lf\n", c, i, cp, d);
fclose(fo);
puts("Resposta Correta (no arquivo):A 999 String criada ;
    para exemplo. 123.456000");
}
```

### Aula 24L - Exercício 3

```
main()
{
FILE *ler;
int i;
char x;
char name[50];
double d;
if((ler = fopen("TESTE.TST","r")) == NULL) {
puts("Nao Posso Abrir TESTE.TST");
exit();
}
fscanf(ler,"%c %d %s %lf", &x, &i, name, &d);
printf("%c %d %s %lf", x, i, name, d);
}
```

### Aula 24L - Exercício 4

```
main()
{
int ret;
FILE *fi, *fo;
char c, source[50], dest[50];
printf("Nome do Arquivo: ");
gets(source);
fi = fopen(source, "w+");
ret = fputs("abcdefghijklmnopqrstuvwxyz", fi);
fclose(fi);
puts("Feito");
}
```

## Aula 24L - Exercício 5

```
/* programa leitor de arquivo, semelhante ao type do DOS */
#define MAXLEN 80
main()
{
FILE *fi;
char source[25], line[MAXLEN], *p;
printf("Nome do Arquivo: ");
gets(source);
if((fi = fopen(source, "r")) == NULL) {
    printf("\tNao posso abrir %s\n", source);
    exit();
}
p = line;
while(p != NULL) {
    p = fgets(line, MAXLEN, fi);
    printf(line);
}
fclose(fi);
puts("\n{FEITO}");
}
```

## Lista 1 - Exerc. 1 (Fibonnaci)

```
main()
{
int n, i, j, l, p;
printf("Digite um numero: ");
scanf("%d", &n);
i=0;
j=1;
printf("%d %d ", i, j);
for(l=1; l<=n-2; l++) {
    p=i+j;
    printf("%d ", p);
    i=j;
    j=p;
}
}
```

## Lista 3 - Exerc. 2 (Passagem de Valores e Variáveis)

```
main()
{
int x;
soma(&x)
printf("\nO valor digitado foi %d\n", x);
}
```

```
soma(z)
int *z;
{
int y;
printf("Digite um Valor: ");
scanf("%d",&y);
*z=y;
}
```