Conexões

Tecnologia

Inovação

SOFTEX

PERNAMBUCO

Lógica de Programação

Professor: Danilo Farias

Algoritmo

- **Lógica de programação** é a técnica de encadear pensamentos para atingir determinado objetivo. O aprendizado desta técnica é necessário, para quem deseja trabalhar com desenvolvimento de sistemas e programas.

- **Algoritmo** é uma sequência de passos finitos com o objetivo de solucionar um problema.

Algoritmo

- **Algoritmo** não é a solução de um problema, pois, se assim fosse, cada problema teria um único algoritmo. Algoritmo é um conjunto de passos (ações) que levam à solução de um determinado problema.

1. Entrada de dados;
2. Processamento de dados;
3. Saída de dados;

Algoritmo

- Exemplo de Algoritmo do nosso cotidiano.

```
1 – Retirar o telefone do gancho
2 – Esperar o sinal
3 – Colocar o cartão
4 – Discar o número
5 – Falar no telefone
6 – Colocar o telefone no gancho
```

- O algoritmo é exatamente esse conjunto de passos que resolveu o problema de uma pessoa falar no telefone.

Algoritmo

- Outro exemplo clássico é um algoritmo para resolver o problema de fritar um ovo que poderia estar escrito em forma de uma receita.

Algoritmo

- Outro exemplo clássico é um algoritmo para resolver o problema de fritar um ovo que poderia estar escrito em forma de uma receita.

```
1 – pegar frigideira, ovo, óleo e sal
2 – colocar óleo na frigideira
3 – acender o fogo
4 – colocar a frigideira no fogo
5 – esperar o óleo esquentar
6 – colocar o ovo
7 – retirar quando pronto
```

- Instrução indica a um computador uma ação elementar a ser executada.

Algoritmo

- Como seria um algoritmo para trocar uma lâmpada.

Algoritmo

- Como seria um algoritmo para trocar uma lâmpada.

```
1 – se (lâmpada estiver fora de alcance)
      pegar a escada;
2 – pegar a lâmpada;
3 – se (lâmpada estiver quente)
      pegar pano;
4 – Tirar lâmpada queimada;
5 – Colocar lâmpada boa;
```

Algoritmo

- Como seria um algoritmo para descascar um saco de batatas?!

Algoritmo

- Como seria um algoritmo para descascar um saco de batatas?!

```
1 – pegar faca, bacia e batatas;
2 – colocar água na bacia;
3 – enquanto (houver batatas)
      descascar batatas;
```

Algoritmo

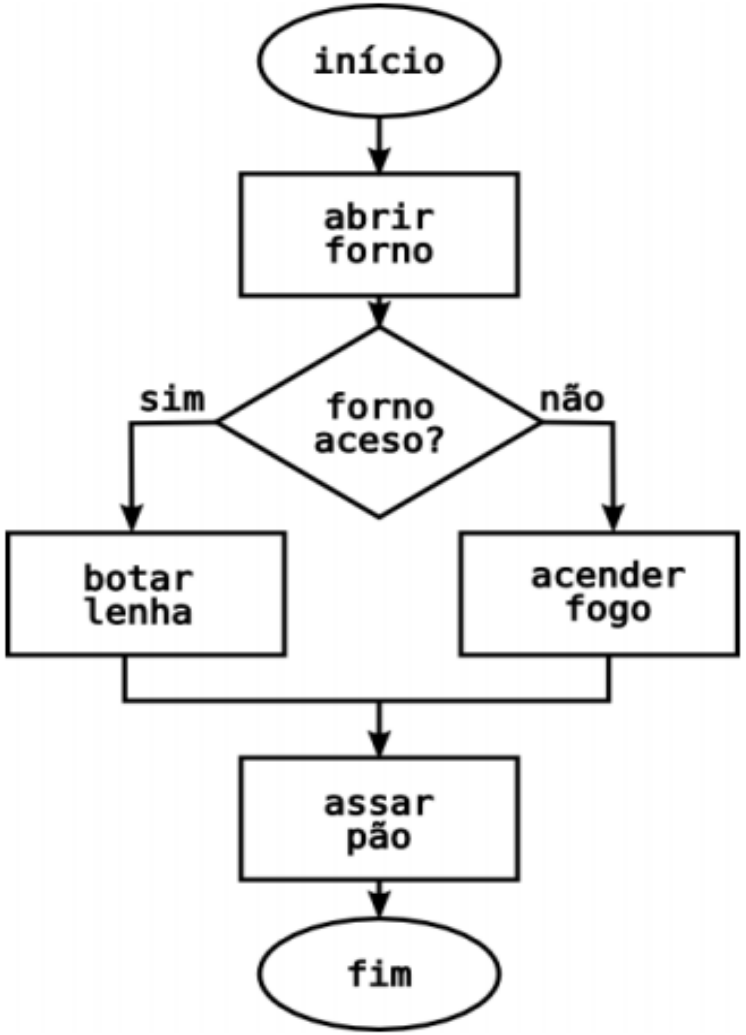
- Fazer um algoritmo para levar um leão, uma cabra e um pedaço de grama de um lado para outro de um rio, atravessando com um bote. Sabe-se que nunca o leão pode ficar sozinho com a cabra e nem a cabra sozinha com a grama.

```
1 – Levar a grama e o leão
2 – Voltar com o leão
3 – Deixar o leão
4 – Levar a cabra
5 – Deixar a cabra
6 – Voltar com a grama
7 – Levar o leão e a grama
```

Representação do Algoritmo

- **Fluxograma** é uma apresentação do algoritmo em formato gráfico. Cada ação ou situação é representada por uma caixa. Tomadas de decisões são indicadas por caixas especiais, possibilitando ao fluxo de ações tomar caminhos distintos.
- **Linguagem** que é qualquer tipo de informação que deva ser transferida, processada ou armazenada deve estar na forma de uma linguagem.

Algoritmo - Fluxograma



Algoritmo - Linguagem

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

int main(){
    setlocale(LC_ALL, "portuguese");
    float r;
    float area, comp;
    const float PI = 3.1415;

    r = 3;
    area = PI * r * r;
    comp = 2 * PI * r;

    printf("Area = %.3f", area);
    printf("\nComprimento = %.3f", comp);

    return 0;
}
```

Linguagem C

- Uma linguagem é considerada estruturada quando permite que o programador pegue trechos de maior uso do seu programa e transforme-os em pequenos módulos (procedimentos e funções) que serão reutilizados sempre que necessário.

- A **linguagem C** alia características de baixo nível (linguagem próxima a de máquina) com características de alto nível (linguagem próxima a do ser humano)

Linguagem C

- C é uma linguagem que possui uma **sintaxe enxuta** e que permite, em geral, um rápido entendimento pelo programador iniciante, desde que sejam destinados a ela alguns momentos de prática dos exercícios.

- É importante ressaltar um ponto muito importante sobre a **linguagem C**: o C é "**sensível a caixa alta ou baixa (*case sensitive*)**, ou seja, faz diferença entre o maiúsculo e o minúsculo".

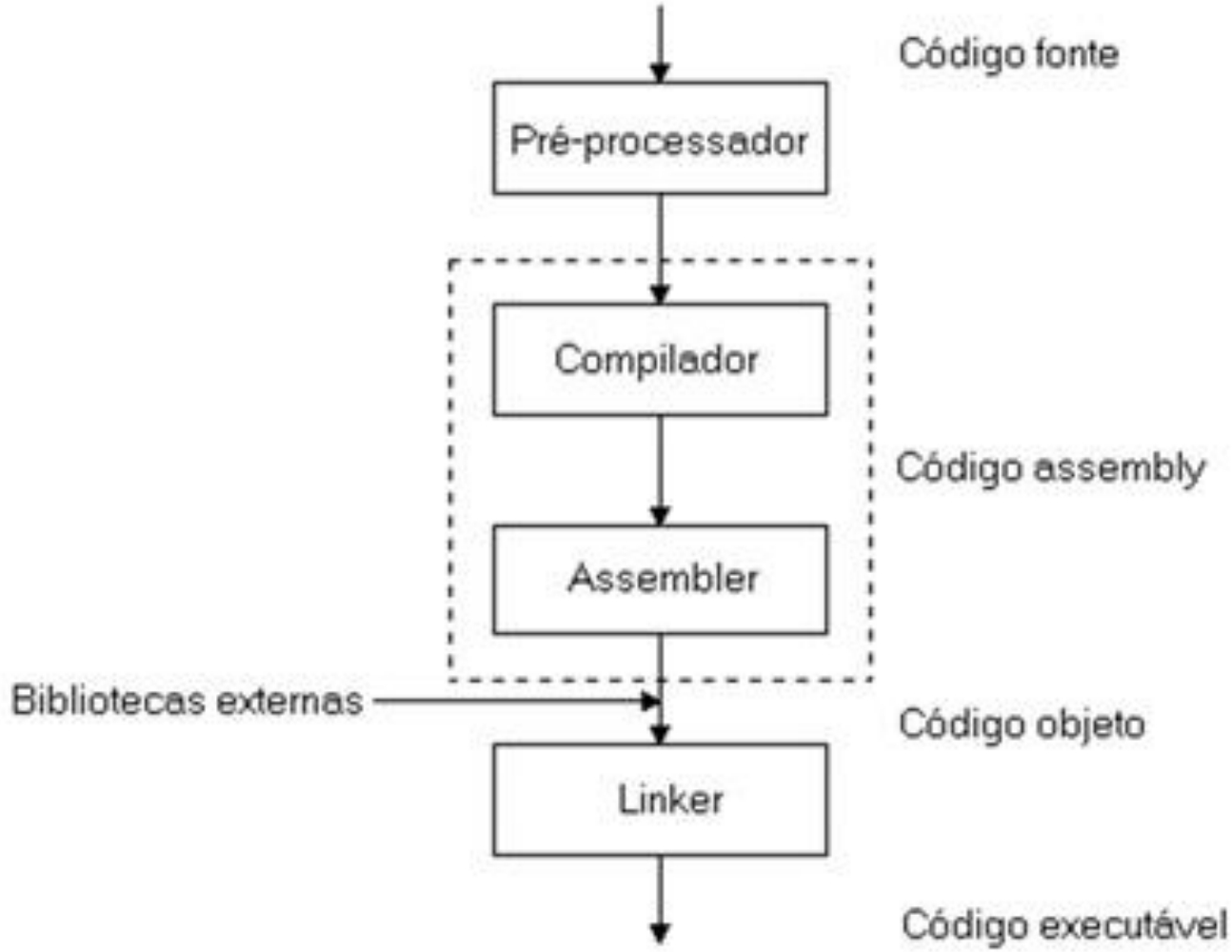
- Podemos dizer que um código escrito em **C é portável estruturadamente**, o que significa que é possível adaptá-lo para os mais diferentes tipos de computadores e sistemas operacionais (*Windows, Linux* etc.) de forma otimizada. (**Portabilidade**)

Linguagem C

- Inicialmente, iremos descrever os termos utilizados para facilitar o entendimento do processo. Chama-se **código-fonte** o texto do programa que os usuários escrevem.

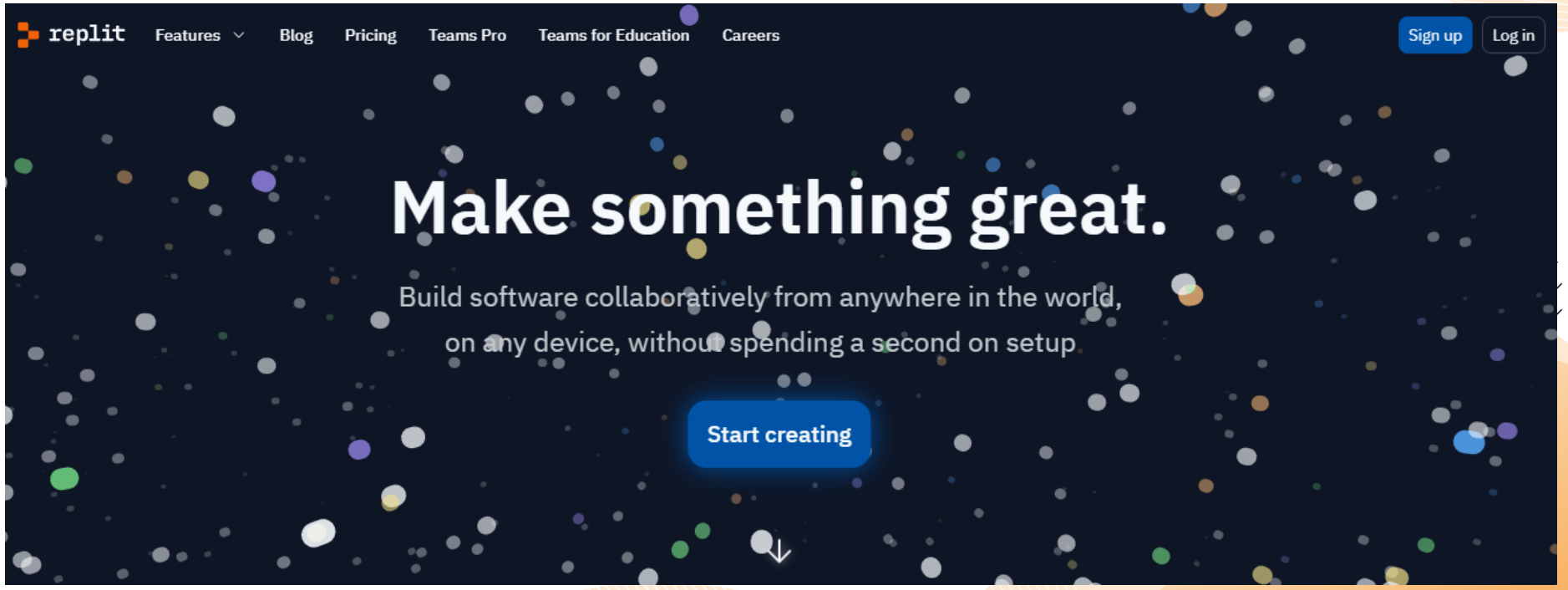
  - O **código-objeto** é a tradução do código-fonte do programa em um código de máquina para que o computador possa ler e executar.
  - O **linkeditor** é um programa que une funções compiladas separadamente em um programa, combinando as funções da biblioteca C com o programa escrito. A sua entrada é o código objeto e a sua saída é um programa executável.
  - **Tempo de compilação** são os eventos ocorridos no momento da compilação dos programas, nesse tempo é verificado se o código escrito está compatível com a sintaxe da linguagem.
  - **Tempo de execução** são os eventos ocorridos enquanto o programa é executado.

Linguagem C


Código fonte
Pré-processador
Compilador
Código assembly
Assembler
Bibliotecas externas
Código objeto
Linker
Código executável

Ambiente de programação em C

- Inicialmente, introduziremos um ambiente de fácil utilização e que apresenta uma interface bastante amigável, o **replit.** Plataforma web que permite escrever e compilar códigos em diversas linguagens.



Linguagem C
Função *main*()

- Função – A base para a linguagem C

```
tipo nomeFunc(declaração dos parâmetros)
{
    declaração de variáveis;
    instrução_2;
    ..............
    instrução_n;
    return var_tipo;
}
```

```
int main()
{

    return 0;
}
```

Estrutura de um programa em C

- A função **main()**

Em todo programa C deve existir uma única função chamada **main()**. Ela marca o ponto de partida do programa, que termina quando for encerrada a execução da função **main()**. Se um programa for constituído de uma única função, esta será **main()**.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   printf("Primeiro programa.");
   system("PAUSE");
   return 0;
}
```

Linguagem C
Função *main*()

- A função **main()**

A função **main()** particular do nosso primeiro programa é do tipo **int**. Isso significa que a função deverá retornar um número inteiro. A nossa instrução de retorno é a seguinte:

```
return 0;
```

```
int main()
{

    return 0;
}
```

Linguagem C
Diretivas

- **Bibliotecas (Diretivas)**

As bibliotecas são arquivos em linguagem de máquina que contêm funções desenvolvidas por outros programadores e podem ser usadas em C.

A biblioteca padrão, fornecida pelos compiladores C, contém funções que executam as operações básicas de I/O. A função **printf()** está associada à saída padrão do sistema operacional (geralmente o vídeo).

DIRETIVAS DO PRÉ-PROCESSADOR

As duas primeiras linhas do nosso programa não são instruções da linguagem C (observe que não há ponto-e-vírgula ao seu final), mas sim diretivas do pré-processador.

A DIRETIVA #include

A diretiva **#include** provoca a inclusão de outro arquivo em nosso programa fonte.

Linguagem C
Diretivas

- **Bibliotecas (Diretivas)**

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("Primeiro programa.");
    system("PAUSE");
    return 0;
}
```

```
#include "meuArq.h"
```

Quando usamos os sinais < e >, o arquivo é procurado somente na pasta **include**, criada na instalação do seu compilador. Quando usamos aspas duplas, o arquivo é procurado primeiramente na pasta atual e depois, se não for encontrado, na pasta **include**.

Linguagem C
printf( )

- **Funções Utilizadas**

A FUNÇÃO printf()

A instrução

```
printf("Primeiro programa.");
```

A FUNÇÃO system()

A função **system()** executa um comando interno do sistema operacional ou um programa (.EXE, .COM ou .BAT). Em nosso programa estamos executando o comando PAUSE.

```
system("PAUSE");
```

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   printf("Primeiro programa.");
   system("PAUSE");
   return 0;
}
```

Linguagem C
printf( )

- **Códigos Especiais – Função *printf()***

Códigos especiais	Significado
\n	Nova linha.
\t	Tabulação.
\b	Retrocesso (usado para impressora).
\f	Salto de página de formulário.
\a	Beep – Toque do auto-falante.
\r	CR – Retorno do cursor para o início da linha.
\\	\ – Barra invertida.
\0	Zero.
\'	Aspas simples (apóstrofo).
\"	Aspas dupla.
\xdd	Representação hexadecimal.
\ddd	Representação octal.

Linguagem C
printf( )

- Explorando o ***printf()***

```
#include <stdio.h> /* Para printf() */
#include <stdlib.h>/* Para system() */
int main()
{
   printf("%s está a %d milhões de milhas\ndo sol.\n", "Venus", 67 );
   system("PAUSE");
   return 0;
}
```

O programa imprimirá na tela:

```
Venus está a 67 milhões de milhas do sol.
```

Linguagem C
Comentário

- **Comentários**

COMENTÁRIOS DE PROGRAMA

Comentários de programa podem ser colocados em qualquer lugar e são tratados pelo compilador como espaços em branco. Eles ajudam quem desenvolve o programa e qualquer pessoa que lê o programa fonte. São utilizados para documentar o código.

```
/*Esse programa mostra o uso de comentários em várias linhas    *
 * e mostra também o uso de comentários em uma única linha      *
 *                                                              *
 *                                                              *
 *                  Primeiro programa                           *
 ****************************************************************
*/
/* Prog1.C */
#include <stdio.h> /* Para printf() */
#include <stdlib.h>/* Para system() */
int main()          /* Função main */
{ /* Início do corpo da função main */
   printf("Primeiro programa."); /* Chamada a função printf */
   system("PAUSE"); /* Chamada a função system */
   return 0;
}/* Fim do corpo da função main */
```

Linguagem C
Variáveis

- **Variáveis**

As variáveis são o aspecto fundamental de qualquer linguagem de computador. Uma variável em C é um espaço de memória reservado para armazenar um certo tipo de dado e tendo um nome para referenciar o seu conteúdo.

Podemos declarar várias variáveis de um mesmo tipo numa única instrução. Elas deverão ser separadas por vírgulas.

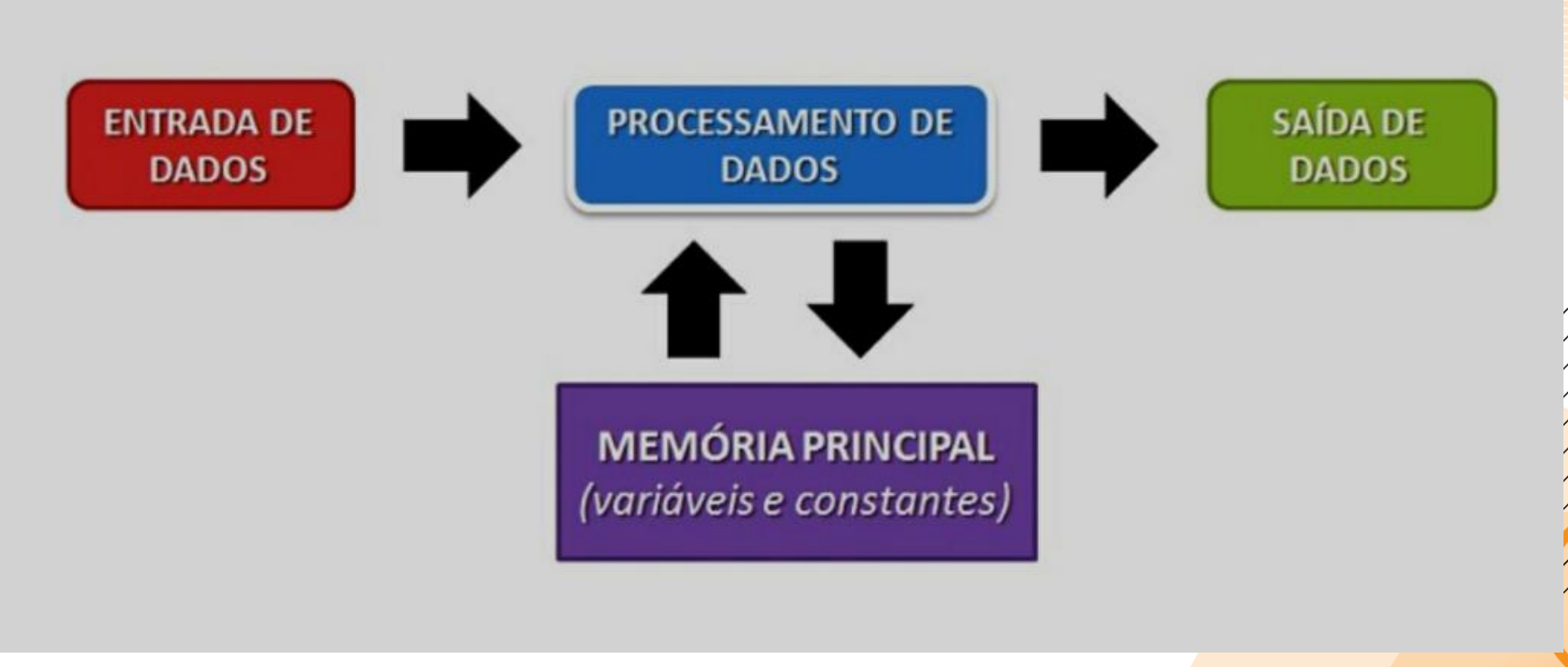
```
int num1, num2;
int aviao, foguete, helicoptero;
```

Toda variável em C deve ser declarada antes de ser usada.

Toda variável em C deve ser declarada no início do bloco de uma função, logo após a abertura da chave e antes de qualquer outra instrução.

Linguagem C
Variáveis

- **Variáveis**



Linguagem C
Variáveis

- **Variáveis**

POR QUE DECLARAR VARIÁVEIS

- Reunir variáveis em um mesmo lugar, dando a elas nomes significativos, facilita ao leitor entender o que o programa faz.
- Uma seção de declarações de variáveis encoraja o planejamento do programa antes de começar a escrevê-lo. Isto é, planejar as informações que devem ser dadas ao programa e quais as que o programa deverá nos fornecer.
- Declarar variáveis ajuda a prevenir erros. Por exemplo, se escrevermos o (letra o) em vez de 0 (zero):

Linguagem C
Variáveis

- **Variáveis**

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

int main(){
    setlocale(LC_ALL, "portuguese");
    int numA, numB;
    int result;

    numA = 44;
    numB = numA + 5;
    result = numB * 1.3;

    printf("\nO primeiro número é o %d", numA);
    printf("\nO segundo número é o %i", numB);
    printf("\nO resultado de 30%% a mais do valor de %d é %d", numB, result);

    return 0;
}
```

Linguagem C
Tipos de Variáveis

- **Tipos de variáveis**

  Os dados primitivos podem ser divididos em quatro categorias:
  - Texto
  - Inteiro
  - Real
  - Lógico

Tipo	Bits	Bytes	Escala
char	8	1	128 a 127
int	32	4	−2.147.483.648 a 2.147.483.647(ambientes de 32 bits)
short	16	2	−32.765 a 32.767
long	32	4	−2.147.483.648 a 2.147.483.647
unsigned char	8	1	0 a 255
unsigned	32	4	0 a 4.294.967.295 (ambientes de 32 bits)
unsigned long	32	4	0 a 4.294.967.295
unsigned short	16	2	0 a 65.535
float	32	4	$3,4 \times 10^{-38}$ a $3,4 \times 10^{38}$
double	64	8	$1,7 \times 10^{-308}$ a $1,7 \times 10^{308}$
long double	80	10	$3,4 \times 10^{-4932}$ a $3,4 \times 10^{4932}$
void	0	0	nenhum valor

Linguagem C
Palavras-Chaves

- **Palavras-chave**

Categoria	Palavras-chave
Tipos de dados	char, int, float, double, void
Modificadores de tipo	long, short, signed, unsigned
Modificadores de tipo de acesso	const, volatile
Comandos condicionais	if, else, switch, case, default
Comandos de laços	while, for, do
Comandos de desvio	break, goto, return, continue
Operador	sizeof

Programa em C

- Explorando o ***printf()***

```
/* Tamanho de campo com inteiros */
#include <stdio.h> /* Para printf() */
#include <stdlib.h>/* Para system() */
int main()
{
    int lapis=45, borrachas=2345, canetas=420,
    cadernos=8,fitas=13050;
    printf("\nLapis          %12d",lapis);
    printf("\nBorrachas      %12d",borrachas);
    printf("\nCanetas        %12d",canetas);
    printf("\nCadernos       %12d",cadernos);
    printf("\nFitas          %12d",fitas);
    system("PAUSE");
    return 0;
}
```

Programa em C

- Explorando o ***printf()***

```
/* Tamanho de campo com ponto flutuante */
#include <stdio.h> /* Para printf() */
#include <stdlib.h>/* Para system() */
int main()
{
   float lapis=4.875, borrachas=234.542, canetas=42.036,
         cadernos=8.0,fitas=13.05;
   printf("\nLapis          %12.2f",lapis);
   printf("\nBorrachas      %12.2f",borrachas);
   printf("\nCanetas        %12.2f",canetas);
   printf("\nCadernos       %12.2f",cadernos);
   printf("\nFitas          %12.2f",fitas);
   system("PAUSE");
   return 0;
}
```

Programa em C

- Explorando o ***printf()***

%d	Imprime o campo em decimal.
%x	Imprime o campo em hexadecimal.
%o	Imprime o campo em octal.
%c	Imprime o campo no formato caractere.

```
/* Definindo a base numérica */
#include <stdio.h> /* Para printf() */
#include <stdlib.h>/* Para system() */
int main()
{
    printf("\n%d",65);
    printf("\n%x",65);
    printf("\n%o",65);
    printf("\n%c",65);
    system("PAUSE");
    return 0;
}
```

A saída será:

```
Decimal: 65
Hexadecimal: 41
Octal: 101
Caractere: A
```

Linguagem C
Operadores

Uma linguagem C é rica em operadores, em torno de 50. Alguns são mais usados que outros, como é o caso do operador de atribuição.

Operador de atribuição

Em C, o sinal de igual não tem a interpretação dada em matemática. Representa a atribuição da expressão à sua direita à variável à sua esquerda. Por exemplo:

```
x = 2000;
```

Linguagem C
Operadores

- Operador de Atribuição

Esse operador é utilizado para armazenar um valor em uma dada variável. Assim, o operador de atribuição nos possibilita armazenar um dado em um espaço de memória, previamente declarado. É importante que o dado que será armazenado seja compatível com o tipo da variável que receberá a atribuição. Por exemplo, as variáveis reais podem receber valores reais e inteiros.

```
x = 2000;
```

atribui o valor 2000 à variável de nome **x**. A ação é executada da direita para a esquerda. Toda vez que utilizamos um operador, criamos uma expressão. Toda expressão tem um valor numérico. É bem simples entender que

```
5 + 2
```

tem o valor 7.

Linguagem C
Operadores

- Operador de Atribuição

Talvez não seja tão simples você compreender que

$$x = 3$$

tem o valor 3. Uma expressão de atribuição tem o valor atribuído. Por esse motivo, podemos escrever

$$y = x = 3$$

e, lembrando que atribuições são executadas da direita para a esquerda, a expressão anterior pode ser escrita

$$y = (x = 3)$$

e y terá valor 3.

Linguagem C
Operadores

- Operador de Aritméticos

Em C temos cinco operadores aritméticos binários (que operam sobre dois operandos) e um operador aritmético unário (que opera com apenas um operando).

- Binários:
  - \+ soma
  - \- subtração
  - * multiplicação
  - / divisão
  - % resto da divisão

Por exemplo,

```
17%5 /*Resulta 2*/
```

tem o valor 2, pois, quando dividimos 17 por 5, restam 2.

- Unário:

O OPERADOR MENOS UNÁRIO: –

O operador menos unário é usado somente para indicar a troca do sinal algébrico do valor associado. Pode também ser pensado como o operador que multiplica seu operando por −1. Por exemplo:

```
x = -8;
x = -x;
```

Depois destas duas instruções, o conteúdo de **x** será 8.
Não existe em C o operador + unário, portanto escrever "x = +8" está errado.

Linguagem C
Operadores

- Operador de Aritméticos - Precedência

A **Precedência** e operadores indica qual operador deverá ser executado primeiro. Nos tempos de escola, aprendemos que a **multiplicação** e a **divisão** têm precedência sobre a **soma** e a **subtração**.

Parêntesis podem ser utilizados para forçar o cálculo de uma expressão numa determinada ordem. Exemplos:

1. a + (b / 2)
2. (a * 2) + 1
3. ((((a*2) + 1)*2)+1)*2+1)

```
n = (z + y) * x;
```

Observe agora a seguinte expressão

```
x - y + 5
```

Os operadores + e − têm a mesma precedência, e a regra de associatividade é da "esquerda para a direita", o que indica que a operação é executada da esquerda para a direita. Na expressão anterior, y será subtraído de x antes da execução da operação de soma.

Linguagem C
Operadores

- Operador de Endereços (&)

A linguagem C oferece um operador que opera sobre o nome da variável e resulta o seu endereço na memória do computador. O operador de endereços é representado pelo símbolo **&.**

Toda variável ocupa uma certa localização na memória, e o seu endereço é o do primeiro byte ocupado por ela. Se você declarou uma variável inteira, nomeou-a **n** e atribuiu a ela o valor 2, quando **n** for referida obteremos 2. Entretanto, se você usar &n, o resultado será o endereço do primeiro byte (byte menos significativo) ocupado por ***n***.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int n;
   n=2;
   printf("Valor=%d, endereço=%p\n",n,&n);
   system("PAUSE");
   return 0;
}
```

Endereços de memória são impressos em hexadecimal (base 16) e o formato usado é %p.
A saída deste programa varia conforme a máquina e a memória do equipamento. Veja um exemplo:

```
Valor=2, endereço=0012FED4
```

Linguagem C
Operadores

- A função ***scanf()***

A função **scanf()** é outra função de I/O (entrada e saíde) presente na biblioteca padrão do C**.** Ela é o complemento da função **printf()** e nos permite ler dados formatados da entrada padrão (teclado). Ela está vinculada a biblioteca **stdio.h**

SINTAXE

```
scanf("expressão de controle", lista de argumentos)
```

O comando scanf é utilizado para a entrada de dados pelo usuário. Seu uso é semelhante ao uso do printf: o primeiro argumento é uma cadeia de caracteres na qual o par "%d" indica um valor inteiro decimal. Os valores lidos são associados às variáveis que aparecem a seguir. Os nomes das variáveis aparecem precedidos de '&'.

```
01      #include <stdio.h>
02      int main()
03      {
04          int a,b,c;
05          scanf("%d %d %d", &a,&b,&c);
06          printf("a:%d b:%d c:%d \n",a,b,c);
07      }
```

Linguagem C
Operadores

- A função ***scanf()***

Códigos de formatação para scanf()	Significado
%c	Caractere simples.
%d	Inteiro decimal com sinal.
%i	Inteiro decimal, hexadecimal ou octal.
%e	Notação científica.
%f	Ponto flutuante em decimal.
%g	Usa %e ou %f, o que for menor.
%o	Inteiro octal.
%s	String de caracteres.
%u	Inteiro decimal sem sinal.
%x	Inteiro hexadecimal.
%ld	Inteiro decimal longo.
%lf	Ponto flutuante longo (double).
%Lf	Double longo.

Linguagem C
Operadores

- A função ***scanf()***

Veja um exemplo:

```
/* Calcula a sua idade em dias */
#include <stdio.h>
#include <stdlib.h>
int main()
{
      float anos,dias;
      printf("Digite a sua idade em anos: ");
      scanf("%f",&anos);
      dias = anos*365;
      printf("A sua idade em dias é %.0f.\n",dias);
      system("PAUSE");
      return 0;
}
```

MÚLTIPLAS ENTRADAS COM scanf()

Podemos ler vários valores com uma única chamada à **scanf()**.

```
/* Mostra o uso de scanf() com várias entradas */
/* Calcula a média de 4 notas */
#include <stdio.h>
#include <stdlib.h>
int main()
{
   float p1, p2, p3, p4;
   double media;
   printf("\nDigite as notas das 4 provas: ");
   scanf("%f%f%f%f",&p1, &p2, &p3, &p4);
   media = (p1 + p2 + p3 + p4)/4.0;
   printf("\nMÉDIA: %.2f\n",media);
   system("PAUSE");
   return 0;
}
```

Eis a saída:

```
Digite as notas das 4 provas: 5.5 7.5 3.0 6.0
MÉDIA: 5.50
```

A função **scanf()** entende um espaço em branco como o término de uma entrada. Múltiplas entradas são digitadas separadas por um espaço em branco. Digitamos [ENTER] como finalizador geral.

Linguagem C
Operadores

- Exemplo da função ***scanf()***

O PROGRAMA QUE CONVERTE TEMPERATURAS

A seguir está um programa que usa vários operadores aritméticos e converte temperaturas de graus Celsius para seus correspondentes graus Fahrenheit.

```
/* Converte temperaturas de graus Celsius para Fahrenheit */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   float ftemp,ctemp;
   printf("Digite temperatura em graus Celsius: ");
   scanf("%f",&ctemp);
   ftemp = ctemp * 9/5 + 32;
   printf("\nTemperatura em graus Fahrenheit é %.2f\n",ftemp);

   system("PAUSE");
   return 0;
}
```

Eis um exemplo:

```
Digite temperatura em graus Celsius: 0
Temperatura em graus Fahrenheit é 32.00

Digite temperatura em graus Celsius: 21
Temperatura em graus Fahrenheit é 69.80
```

Linguagem C
Operadores

- Exemplo da função ***scanf()***

```
/* Converte temperaturas de graus Celsius para Fahrenheit */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   float ctemp;
   printf("Digite temperatura em graus Celsius: ");
   scanf("%f",&ctemp);
   printf("\nTemperatura em graus Fahrenheit é %.2f\n", ctemp *
   9/5 + 32);

   system("PAUSE");
   return 0;
}
```

Linguagem C - Operadores

- Exemplo da função

scanf() com Constante ***const***

O QUALIFICADOR const

A palavra-chave **const** assegura que a variável associada não será alterada em todo o programa. Esse qualificador é indicado para declarar valores constantes. Veja um exemplo de seu uso:

```
/* Mostra o uso de const para declarar constantes*/
#include <stdio.h>
#include <stdlib.h>
int main()
{
   const char Bip = '\a'; /*Declaração de constante*/
   const double Pi = 3.141592;/*Declaração de constante*/
   double raio, area;

   printf("\nDigite o raio da esfera: ") ;
   scanf("%lf",&raio);

   area= 4.0 * Pi * raio * raio;

   printf("%c%c", Bip, Bip);
   printf("\nArea da esfera = %.2lf\n", area);
   system("PAUSE");
   return 0;
}
```

Obrigatoriamente, as variáveis associadas ao qualificador **const** devem ser inicializadas.

Linguagem C
Operadores

- Operadores de Incremento (++) e de Decremendo (--)

```
    x = x + 1; /* Adiciona 1 a x */
```

é equivalente a

```
    ++x; /* Adiciona 1 a x */
```

que é equivalente a

```
    x++; /* Adiciona 1 a x */
```

```
    x = x - 1; /* Decrementa 1 de x */
```

é equivalente a

```
    --x; /* Decrementa 1 de x */
```

que é equivalente a

```
    x--;  /* Decrementa 1 de x */
```

Linguagem C
Operadores

- Operadores de Incremento (++) e de Decremendo (--)

```
n = 5;
x = ++n;
printf("\nN=%d    X=%d", n , x);
```

A saída será

```
N=6    X=6
```

```
n = 5;
x = n++;
printf("\nN=%d    X=%d", n , x);
```

A saída será

```
N=6    X=5
```

```
int n = 5;
printf("%d\t%d\t%d\n", n, n+1 , n++);
```

Isso parece razoável, e você pode pensar que a impressão será:

```
5    6    7
```

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int i=3,n;
   n = i * (i+1) + (++i);
   printf("\nn = %d\n", n);
   system("PAUSE");
   return 0;
}
```

Linguagem C
Operadores

- Operadores Aritméticos de Atribuição

$+= \quad -= \quad *= \quad /= \quad \%=$

x op = exp	equivale a	x = x op (exp).

Exemplos:

```
i  += 2;      equivale a    i = i + 2;
x  *= y+1;    equivale a    x = x * (y+1);
t  /= 2.5;    equivale a    t = t / 2.5;
p  %= 5;      equivale a    p = p % 5;
d  -= 3;      equivale a    d = d - 3;
```

SOFTEX
PERNAMBUCO

- Operadores Aritméticos de Atribuição

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    float nota , media = 0.0;

    printf("\nDigite a primeira nota: ");
    scanf("%f", &nota);
    media += nota;

    printf("\nDigite a segunda nota: ");
    scanf("%f", &nota);
    media += nota;

    printf("\nDigite a terceira nota: ");
    scanf("%f", &nota);
    media += nota;

    printf("\nDigite a quarta nota: ");
    scanf("%f", &nota);
    media += nota;

    media /= 4.0;
    printf("\nMÉDIA: %.2f\n", media);
    system("PAUSE");
    return 0;
}
```

Linguagem C
Operadores

- Operadores Relacionais

Os operadores relacionais fazem comparações. São eles:

>	maior
>=	maior ou igual
<	menor
<=	menor ou igual
= =	igual
!=	diferente

```
/* Mostra operadores relacionais */
#include <stdio.h>
#include <stdlib.h>
int main()
{
      int Verdadeiro, Falso;

      Verdadeiro = (15 < 20);
      Falso      = (15 == 20);
      printf("Verdadeiro %d\n", Verdadeiro);
      printf("Falso      %d\n", Falso);
      system("PAUSE");
      return 0;

}
```

Linguagem C
Operadores

- Operadores Lógicos

C oferece três operadores lógicos. São eles:

&&	lógico E
||	lógico OU
!	lógico NÃO

Alguns exemplos:

```
(dia==7 && mes==6)     /* Verdadeiro se dia for 7 e mês for 6 */
(op=='S' || op=='s')   /* Verdadeiro se op for S ou s */
( !masculino )         /* Verdadeiro se masculino for 0 */
```

Outros exemplos:

```
!5                     /* Lê-se "não verdadeiro" ou 0. */
1 || 2
x && y
a < b || a==c
a >= 5 && x <= 6*y
!x
!'j'
!(a+3)
```

Operadores	Operandos	Resultado
Aritméticos	Numéricos	Numérico
Relacionais	Numéricos	Lógico
Lógicos	Lógicos	Lógico

Linguagem C
Estruturas Condicionais

- Comandos de Decisão – ***if ; if-else ; switch***

Uma das tarefas fundamentais de qualquer programa é decidir o que deve ser executado a seguir. Os comandos de decisão permitem determinar qual é a ação a ser tomada com base no resultado de uma expressão condicional. Isso significa que podemos selecionar entre ações alternativas, dependendo de critérios desenvolvidos no decorrer da execução do programa.

A linguagem C oferece três comandos de decisão:

if
if-else
switch

Linguagem C
Estruturas Condicionais

- Comandos de Decisão – ***if***

O COMANDO if

O comando **if** instrui o computador a tomar uma decisão simples.

```
/* ifdemo.c */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int anos;
   printf("Quantos anos você tem? \n");

   scanf("%d", &anos);

   if(anos < 30) /* Toma uma decisão caso anos < 30 */
         printf("Você é muito jovem!\n");

   system("PAUSE");
   return 0;
}
```

SINTAXE DO COMANDO if

```
if (Expressão de teste)
{
        instrução;
        instrução;
}
```

Linguagem C
Estruturas Condicionais

- Comandos de Decisão – ***if-else***

O COMANDO if-else

O comando **if-else** é a expansão de um simples **if**. O comando **if** permite que executemos algo somente se a sua expressão de teste for verdadeira, caso contrário nada é executado.

Suponhamos que você queira executar alguma coisa se a expressão de teste for verdadeira e outra coisa se a expressão de teste for falsa. Neste caso, você deve usar o comando **if-else**.

SINTAXE DO COMANDO if-else

If-else consiste no comando **if** acompanhado de uma instrução ou de um bloco de instruções, seguido da palavra-chave **else**, também acompanhada de uma instrução ou de um bloco de instruções.

Uma única instrução não necessita de chaves:

```
if (Expressão de teste)
      instrução;
else
      instrução;
```

Várias instruções necessitam estar entre chaves:

```
if (Expressão de teste)
{
      instrução;
      instrução;
}
else
{
      instrução;
      instrução;
}
```

Linguagem C
Estruturas Condicionais

- Comandos de Decisão – ***if-else***

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

int main(){
    setlocale(LC_ALL, "portuguese");
    int idade;
    printf("Qual a sua idade: ");
    scanf("%d", &idade);
    if(idade<16){
        printf("\nVocê não pode votar ainda!\n");
    }else if(idade>=16 && idade<18){
        printf("\nTire seu título e vá votar!\n");
    }else if(idade>=18 && idade<65){
        printf("\nVocê tem a obrigação de ir votar!\n");
    }else{
        printf("\nVocê pode votar, mas não é mais obrigatório!\n");
    }

    return 0;
}
```

Linguagem C
Estruturas Condicionais

- Comandos de Decisão – ***if-else***

```c
int main(){
    setlocale(LC_ALL, "portuguese");
    float nota1, nota2, nota3, media;
    printf("Digite a primeira nota: ");
    scanf("%f", &nota1);
    printf("\nDigite a segunda nota: ");
    scanf("%f", &nota2);
    printf("\nDigite a terceira nota: ");
    scanf("%f", &nota3);
    media = (nota1+nota2+nota3)/3.0;
    if(media<=10.0 && media>=9.5){
        printf("\n\nSua média foi %.2f, seu conceito final é Excelente!", media);
    }else if(media<9.5 && media>=8.0){
        printf("\n\nSua média foi %.2f, seu conceito final é Ótimo!", media);
    }else if(media<8.0  && media>=7.0){
        printf("\n\nSua média foi %.2f, seu conceito final é Bom!", media);
    }else if(media<7.0 && media>=5.0){
        printf("\n\nSua média foi %.2f, seu conceito final é ANS!", media);
    }else if(media<5.0 && media>=0.0){
        printf("\n\nSua média foi %.2f, seu conceito final é Ruim!", media);
    }else{
        printf("\n\nSua média foi %.2f, notas erradas!!!", media);
    }
    return 0;
}
```

Linguagem C – Estruturas Condicionais

- Comandos de Decisão – ***if-else***

```
/* calculadora.c */
/* Simula uma calculadora de 4 operações*/
#include <stdio.h>
#include <stdlib.h>

int main()
{
   const int TRUE=1;
   while(TRUE)   /* Sempre verdadeiro */
   {
         float n1,n2;
         char op;

         printf("\nDigite número operador número: ");
         scanf("%f%c%f", &n1, &op, &n2);

         if(op == '+')
               printf("\n%f", n1 + n2);
         else
               if(op == '-')
                     printf("\n%f", n1 - n2);
               else
                     if(op == '*')
                           printf("\n%f", n1 * n2);
                     else
                           if(op == '/')
                                 printf("\n%f", n1 / n2);
                           else
                                 printf("Op. desconhecido.");
   }
   printf("\n");
   system("PAUSE");
   return 0;
}
```

Linguagem C – Estruturas Condicionais

- Comandos de Decisão – ***switch***

O COMANDO switch

O comando **switch** permite selecionar uma entre várias ações alternativas. Embora construções **if-else** possam executar testes para escolha de uma entre várias alternativas, muitas vezes são deselegantes. O comando **switch** tem um formato limpo e claro.

SINTAXE DO COMANDO switch

```
switch (variável ou constante)  <-- Sem ponto-e-vírgula
{
   case constante1:             <-- Dois pontos
                instrução;
                instrução;
                break;
   case constante2:
                instrução;
                instrução;
                break;
   case constante3:

                instrução;
                instrução;
                break;
   default:
                instrução;
                instrução;
}
```

Linguagem C – Estruturas Condicionais

- Comandos de Decisão – ***switch***

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

int main(){
    setlocale(LC_ALL, "portuguese");
    int num1, num2;
    char op;
    printf("Calculadora Digital - Informe uma expressão (Exp 6 * 7): ");
    scanf("%d %c %d", &num1, &op, &num2);
    switch(op){
        case '+':
            printf("\n%d + %d = %d\n", num1, num2, num1 + num2);
            break;
        case '-':
            printf("\n%d - %d = %d\n", num1, num2, num1 - num2);
            break;
        case '*':
            printf("\n%d * %d = %d\n", num1, num2, num1 * num2);
            break;
        case '\\':
            printf("\n%d \ %d = %d\n", num1, num2, num1 / num2);
            break;
        case '/':
            printf("\n%d / %d = %d\n", num1, num2, num1 / num2);
            break;
        default:
            printf("\nOperador invalido!\n");
    }
    return 0;
}
```

Linguagem C
Laços de Repetição

- Laços – ***for ; while ; do-while***

Laços são comandos usados sempre que uma ou mais instruções tiverem de ser repetidas enquanto uma certa condição estiver sendo satisfeita. Em C existem três comandos de laços:

for
while
do-while

Linguagem C
Laços de Repetição

- O Laço ***for***

O laço **for** é geralmente usado quando queremos repetir algo por um número fixo de vezes. Isso significa que utilizamos um laço **for** quando sabemos de antemão o número de vezes a repetir.

O exemplo seguinte imprime uma linha com vinte asteriscos (*) utilizando um laço **for** na sua forma mais simples.

```
/* ForDemo.c */
/* Mostra um uso simples do laço for */
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int i;
  for(i=0 ; i < 20 ; i++) /* Imprime 20 *  */
        printf("%c" , '*');

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

Linguagem C
Laços de Repetição

- O Laço ***for*** - Sintaxe

Linguagem C
Laços de Repetição

- O Laço ***for*** - Exemplo

Veja outro exemplo:

```
/* Tabuada6.c */
/* Imprime a tabuada do 6 */
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int i;
   for(i=1 ; i < 10 ; i++)
         printf("\n%4d x 6 = %4d" , i ,i*6);

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

Linguagem C
Laços de Repetição

- O Laço ***for*** - Exemplo

```
/* Tabuada61.c */
/* Imprime a tabuada do 6 invertida */
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i;
    for(i=9 ; i > 0 ; i--)
          printf("\n%4d x 6 = %4d" , i ,i*6);

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

Linguagem C
Laços de Repetição

- O Laço ***for*** - Exemplo

```
/* Multipl3.c */
/* Imprime os múltiplos de 3 entre 3 e 100 */
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int i;
   for( i=3 ; i <= 100 ; i += 3)
         printf("%d\t" , i);

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

Linguagem C
Laços de Repetição

- O Laço ***for*** - Exemplo

O OPERADOR VÍRGULA

Qualquer uma das expressões de um laço **for** pode conter várias instruções separadas por vírgulas. A vírgula, nesse caso, é um operador C que significa "faça isso e depois isso". Um par de expressões separadas por vírgula é avaliado da esquerda para a direita.

```
/* Mostra o uso do operador virgula no laço for */
/* Imprime os números de 0 a 98 de 2 em 2 */
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int i,j;
   for(i=0, j=i; (i+j) < 100 ; i++, j++)
         printf("%d " , i +j);

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

Linguagem C
Laços de Repetição

- O Laço ***for*** - Exemplo

USANDO CARACTERES

A variável do laço pode ser do tipo **char**. Veja um exemplo:

```
/* Mostra o uso de uma variável do tipo char para controle do laço */
 * Imprime as letras minúsculas e seus correspondentes valores
 * em decimal na tabela ASCII */
#include <stdio.h>
#include <stdlib.h>
int main()
{
   char ch;
   for(ch='a'; ch <= 'z'; ch++)
         printf("\nO valor ASCII de %c é %d", ch , ch);

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

Linguagem C
Laços de Repetição

- Usando Chamadas a Funções
  - É possível chamar funções de dentro de expressões do laço FOR

```
/* Codifica a entrada digitada */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h> /* Para getch() */
int main()
{
   unsigned char ch;
   for(ch=getch(); ch != 'X' ; ch = getch())
         printf("%c", ch +1);
   printf("\n");
   system("PAUSE");
   return 0;
}
```

Linguagem C
Laços de Repetição

- O Laço ***for*** - Exemplo

Omitindo expressões do laço for

Qualquer uma das três expressões de um laço **for** pode ser omitida, embora os ponto-e-vírgulas devam permanecer. Se a expressão de inicialização ou a de incremento for omitida, será simplesmente desconsiderada. Se a condição de teste não estiver presente, será considerada permanentemente verdadeira.

```
/* Codifica a entrada digitada */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h> /* Para getch() */
int main()
{
   unsigned char ch;
   for(; (ch=getch()) != 'X' ;)
         printf("%c", ch +1);

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

Laço infinito

Um laço infinito é aquele que é executado sempre, sem parar. Ele sempre tem a expressão de teste verdadeira, e um modo de parar sua execução é desligando o computador.

```
for (;;)
   printf("Laço Infinito");
```

Linguagem C
Laços de Repetição

- O Laço ***for*** - Exemplo

MÚLTIPLAS INSTRUÇÕES NO CORPO DE UM LAÇO for

Se um laço **for** deve executar várias instruções a cada iteração, elas precisam estar entre chaves.

Sintaxe:

```
for(i=0; i<10; i++)
{
    instrução ;
    instrução ;
    instrução ;
}
```

```
/* MEDIA.C
 * Imprime a média aritmética de 10 notas */
#include <stdio.h>
#include <stdlib.h>
int main()
{
   float soma = 0.0;
   const int max = 10;
   int i;

   for(i=0; i < max ; i++)
   {
         float nota;
         printf("\nDigite a nota %d : " , i+1);
         scanf("%f", &nota);
         soma += nota;
   }
   printf("\nMédia = %.2f\n", soma/max);

   system("PAUSE");
   return 0;
}
```

Linguagem C
Laços de Repetição

- O Laço ***for*** - Exemplo

Laços for aninhados

Quando um laço **for** faz parte do corpo de outro laço **for**, dizemos que o laço interno está *aninhado*.

Para mostrar esse uso, preparamos um programa que imprime as tabuadas do 2 ao 9.

```
int main()
{
   int i,j,k;
   system("cls");/*Limpa a tela */

   for(k=0; k<=1 ; k++)
   {
         printf("\n");
         for(i=1 ; i <= 4 ; i++)
               printf("TABUADA DO %3d    ", i+4*k+1);
         printf("\n");

         for(i = 1; i <= 9 ; i++)
         {
               for(j=2+4*k; j <= 5+4*k; j++)
                     printf("%3d x%3d = %3d    ", j,i,j*i);
               printf("\n");
         }
   }
   system("PAUSE");
   return 0;
}
```

Linguagem C
Laços de Repetição

- O Laço ***while***

O comando **while** consiste na palavra-chave **while** seguida de uma expressão de teste entre parênteses. Se a expressão de teste for verdadeira, o corpo do laço é executado uma vez e a expressão de teste é avaliada novamente. Esse ciclo de teste e execução é repetido até que a expressão de teste se torne falsa (igual a zero), então o laço termina e o controle do programa passa para a linha seguinte ao laço.

```
Inicialização ;

while (Teste)
{
                .
                .
          Incremento;
                .
}
```

SOFTEX
PERNAMBUCO

- O Laço ***while*** **- Exemplo**

```
#include <stdio.h>

int main(){
        float soma;
        const int numNotas = 4;
        int i = 0;

        printf("\n");

        while(i<numNotas){
                float nota;
                printf("\n Digite a nota %d: ", i++);
                scanf("%f",&nota);
                soma += nota;
        }

        /*for(i=0; i < numNotas ; i++){
                float nota;
                printf("\n Digite a nota %d: ", i++);
                scanf("%f",&nota);
                soma += nota;
        }*/

        printf("\n Média = %.2f\n", soma/numNotas);

        return 0;
}
```

Linguagem C
Laços de Repetição

- O Laço ***do while***

O terceiro e último comando de laço em C é o laço **do-while**. Esse laço é bastante similar ao laço **while**. Ele é utilizado em situações em que é necessário executar o corpo do laço uma primeira vez e, depois, avaliar a expressão de teste e criar um ciclo repetido.

SINTAXE DO LAÇO do-while

```
do
{
    instrução;
    instrução;

} while (teste);
```

Ponto-e-vírgula aqui

SOFTEX
PERNAMBUCO

- O Laço ***do while –*** Exemplo

```c
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <locale.h>

int main(){
    setlocale(LC_ALL, "portuguese");
    char c;
    do{
        //printf("\n\n");
        system("CLS");
        for(int i=0; i<6 ; i++){
            int numSorteado = rand();
            numSorteado = (numSorteado%60)+1;
            printf("%d ", numSorteado);
        }
        printf("\n\nVocê deseja repetir o sorteio? (s-sim, n-não): ");
        c = getch();
    }while(c=='s');
    return 0;
}
```

Linguagem C
Funções

- Funções

> Uma função é um conjunto de instruções desenhadas para cumprir uma tarefa particular e agrupadas numa unidade com um nome para referenciá-la.
>
> Funções dividem grandes tarefas de computação em tarefas menores, e permitem às pessoas trabalharem sobre o que outras já fizeram, em vez de partir do nada. Uma das principais razões para escrever funções é permitir que todos os outros programadores C a utilizem em seus programas.

Linguagem C
Funções

- Chamando uma Função

Você já escreveu programas qúe chamam funções. Como exemplo, considere o seguinte programa:

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

int main()
{
   int n;
   printf("Digite um número: ");
   scanf("%d", &n);
   printf("O quadrado do número é %d.\n", (n*n));
   system("PAUSE");
   return 0;
}
```

Linguagem C - Funções

- Criando uma Função

```
/* Mostra a escrita da função celsius() */
#include <stdio.h>
#include <stdlib.h>

float celsius(float);  /* Protótipo ou declaração da função */

int main()
{
   float c, f;
   printf("Digite a temperatura em graus Fahrenheit: ");
   scanf("%f", &f);

   c = celsius(f);     /* Chamada à função */

   printf("Celsius = %.2f\n", c);

   system("PAUSE");
   return 0;
}
/* celsius() */
/* Definição da função */
float celsius(float fahr)
{
   float c;
   c = (fahr - 32.0) * 5.0/9.0;
   return c;
}
```

Como você pode ver, a estrutura de uma função C é semelhante à da função **main()**. A diferença é que **main()** possui um nome especial.

Linguagem C
Funções

- O Protótipo de uma Função

Uma função não pode ser chamada sem antes ter sido declarada. A declaração de uma função é dita *protótipo da função*, é uma instrução geralmente colocada no início do programa que estabelece o tipo da função e os argumentos que ela recebe. O *protótipo da função* permite que o compilador verifique a sintaxe de chamada à função.

Quando chamamos a função **getche()** devemos incluir o arquivo **conio.h**, pois é lá que está escrito o protótipo dela.

O protótipo de uma função deve preceder a sua definição e a sua chamada.

```
float celsius(float);  /* Protótipo */
```

Essa declaração informa que a função de nome **celsius()** é do tipo **float** e recebe como argumento um valor **float**.

Linguagem C
Funções

- Tipo de uma Função

O tipo de uma função é definido pelo tipo de valor que ela retorna por meio do comando **return**. Uma função é do tipo **float** quando retorna um valor do tipo **float**.

Os tipos de funções C são os mesmos tipos que o das variáveis, exceto quando a função não retorna nada. Nesse caso, ela é do tipo **void**.

> O tipo de uma função é determinado pelo valor que ela retorna via comando **return**, e não pelo tipo de argumentos que ela recebe.

```
/* celsius() */
/* Definição da função */
float celsius(float fahr)
{
   float c;
   c = (fahr - 32.0) * 5.0/9.0;
   return c;
}
```

SOFTEX PERNAMBUCO

Linguagem C
Funções

- O comando ***return***

O comando **return** termina a execução da função e retorna o controle para a instrução seguinte do código de chamada.

```
return;
return expressão;
return (expressão);
```

Podemos eliminar a variável declarada no corpo da função **celsius()** e colocar a expressão de cálculo diretamente no comando **return**. Veja a mudança:

```
/* celsius() */
/* Definição da função */
float celsius(float fahr)
{
   return (fahr - 32.0) * 5.0/9.0;
}
```

Funções do tipo **void** podem ter um comando **return** desacompanhado de expressão. Nesse caso, o comando **return** serve para terminar a execução da função. Em funções do tipo **void**, o comando **return** não é obrigatório. Uma função sem comando **return** termina quando encontra a chave de fechamento (}).

Linguagem C
Funções

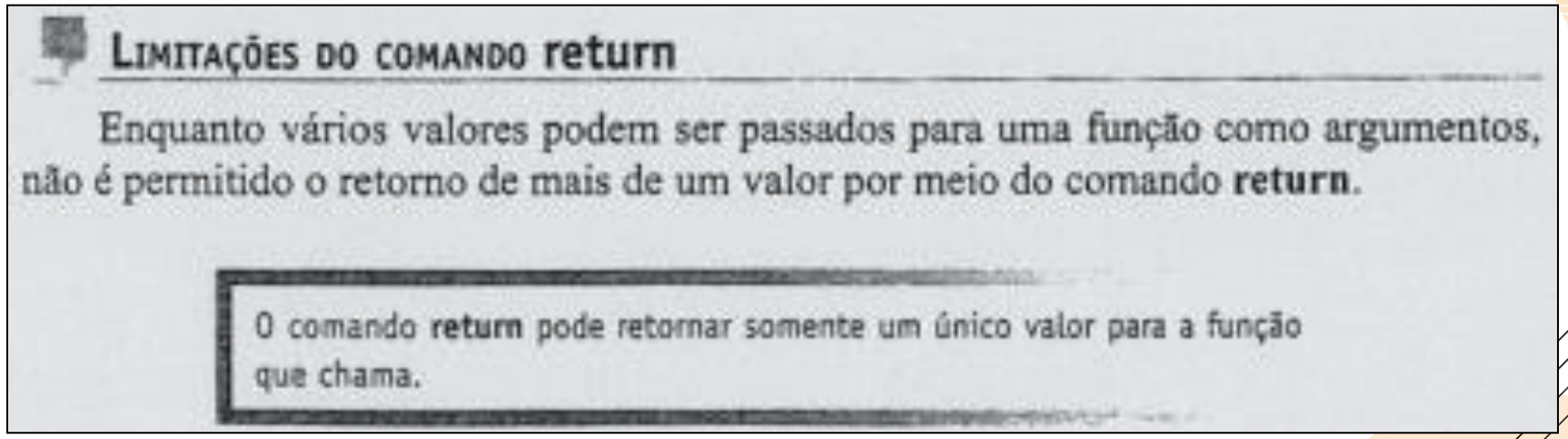
- Definição de um Função

```
tipo nome (declaração dos parâmetros)
{
   instruções;
}
```

```
/* celsius() */
/* Definição da função */
float celsius(float fahr)
{
   float c;
   c = (fahr - 32.0) * 5.0/9.0;
   return c;
}
```

Linguagem C
Funções

- O comando ***return***



LIMITAÇÕES DO COMANDO return

Enquanto vários valores podem ser passados para uma função como argumentos, não é permitido o retorno de mais de um valor por meio do comando **return**.

O comando **return** pode retornar somente um único valor para a função que chama.

Linguagem C – Funções

- Diretiva ***#define***

A diretiva **#define**, na sua forma mais simples, é usada para definir constantes com nomes apropriados. Veja o exemplo:

```
/* Mostra o uso da diretiva #define */
#include <stdio.h>
#include <stdlib.h>

#define PI 3.14

float area(float); /* Protótipo */

int main()
{
    float raio;
    printf("Digite o raio da esfera: ");(raio));
    scanf("%f",&raio);
  return 0;
}
/* area() */
/* Retorna a área da esfera */
float area(float r) /* Definição da função */
{
  return(4 * PI * r * r);
}
```

Por convenção, o identificador é sempre escrito em letras maiúsculas. Observe que não há ponto-e-vírgula após nenhuma diretiva do pré-processador. Cada diretiva deve ser escrita em uma linha nova. Em outras palavras, não podemos escrever mais de uma diretiva numa mesma linha.

Linguagem C
Funções

- Exemplo de um Função

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

#define PI 3.14159

float esfera(float r, char op){
    if(op == 'A'){
        return 4.0*r*r*PI;
    }else if(op == 'V'){
        return (4.0*r*r*r*PI)/3.0;
    }else if(op == 'P'){
        return 2*r*PI;
    }
}

int main(){
    setlocale(LC_ALL, "portuguese");
    float r;
    printf("Informe o raio da esfera: ");
    scanf("%f", &r);
    printf("\nP = %.2f | A = %.2f | V = %.2f", esfera(r,'P'), esfera(r,'A'), esfera(r,'V')) ;
    return 0;
}
```

Linguagem C
Funções

- Função Recurssiva

Uma função é dita *recursiva* se é definida em seus próprios termos, isto é, quando dentro dela há uma instrução de chamada para ela mesma.

$$n! = n * (n-1)!$$

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

int fatorial(int n){
    if(n==1){
        return n;
    }else {
        return n*fatorial(n-1);
    }
}

int main(){
    setlocale(LC_ALL, "portuguese");
    int num;
    printf("Informe um número inteiro positivo: ");
    scanf("%d", &num);
    if(num <= 0){
        printf("\n\nValor invalido");
    }else{
        printf("\n\n %d! = %d", num, fatorial(num));
    }
    return 0;
}
```

Linguagem C – Funções

- Diretiva ***#include***

A diretiva **#include** causa a inclusão de outro arquivo em nosso programa-fonte. Na verdade, o compilador substitui a diretiva **#include** de nosso programa pelo conteúdo do arquivo indicado, antes de compilar o programa.

A linha

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

solicita ao compilador que inclua o arquivo **stdio.h** em nosso programa antes de compilá-lo.

Além do uso dos sinais de < e >, a diretiva **#include** accita uma segunda sintaxe:

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

Linguagem C – Funções

- Diretiva ***#include***

Suponha que você tenha escrito várias fórmulas matemáticas para calcular as áreas de diversas figuras:

```
#define PI                          3.14159
#define A_CIRC(raio)                (PI*(raio)*(raio))
#define A_RET(base,altura)          ((base)*(altura))
#define A_TRI(base,altura)          ((base)*(altura)/2)
#define A_ELIP(raio1,raio2)         (PI*(raio1)*(raio2))
#define A_TRAP(alt,lad1,lad2)       ((alt)*((lad1)+(lad2))/2)
```

O texto acima pode ser gravado com o nome **areas.h** e todo programa que fizer uso destas macros deve simplesmente conter a diretiva:

```
#include "areas.h"
```

As possibilidades do uso de **#define** e **#include** podem ser bastante criativas. Analise o programa a seguir:

Linguagem C – Matrizes

- Matrizes

Matriz é uma coleção de variáveis de mesmo tipo que compartilham um mesmo nome.

```c
/* notas.c */
/* Calcula a média de cinco notas (não usa matriz) */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int nota0, nota1, nota2, nota3, nota4;
   float media;

   printf("Digite a nota do aluno 1 "); scanf("%d",&nota0);
   printf("Digite a nota do aluno 2 "); scanf("%d",&nota1);
   printf("Digite a nota do aluno 3 "); scanf("%d",&nota2);
   printf("Digite a nota do aluno 4 "); scanf("%d",&nota3);
   printf("Digite a nota do aluno 5 "); scanf("%d",&nota4);

   media = (nota0 + nota1 + nota2 + nota3 + nota4) / 5.0;
   printf("Média das notas: %.2f\n", media);

   system("PAUSE");
   return 0;
}
```

Linguagem C – Matrizes

- Matrizes

```
/* notas.c */
/* Calcula a média de cinco notas (usa matriz) */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int notas[5];
   int i;
   float media = 0.0;

      for(i=0 ; i < 5 ; i++)
   {
         printf("Digite a nota do aluno %d ", i+1);
         scanf("%d",&notas[i]);
         media += notas[i];
      }
   media /= 5.0;
   printf("Média das notas: %.2f\n", media);

   system("PAUSE");
   return 0;
}
```

Linguagem C – Matrizes

- Declarando uma Matrizes

```
/* notas.c */
/* Calcula a média de cinco notas (usa matriz) */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int notas[5];
   int i;
   float media = 0.0;

      for(i=0 ; i < 5 ; i++)
   {
         printf("Digite a nota do aluno %d ", i+1);
         scanf("%d",&notas[i]);
         media += notas[i];
      }
   media /= 5.0;
   printf("Média das notas: %.2f\n", media);

   system("PAUSE");
   return 0;
}
```

Linguagem C – Matrizes

- Referenciando um elemento da Matrizes

Os elementos são sempre numerados por índices iniciados por zero. Por exemplo, a instrução

```
notas[2] = 90;
```

O índice utilizado para referenciar elementos de uma matriz pode ser o valor de uma variável inteira ou uma constante. Em nosso programa, utilizamos a variável **i** como índice.

Observe a instrução

```
scanf("%d",&notas[i]);
```

Quando escrevemos **notas[i]**, estamos escrevendo o nome de uma variável do tipo **int** como outra qualquer. Assim, em todo lugar em que podemos utilizar o nome de uma variável **int**, podemos usar **notas[i]**.

Linguagem C – Matrizes

- Exemplo de Matrizes com ***#define***

```
/* notasf.c */
/* Calcula a média das notas de 5 alunos usando matriz float */
#include <stdio.h>
#include <stdlib.h>

#define TAMANHO 5 /* Não podemos usar const */

int main()
{

   float notas[TAMANHO] , media=0.0;
   int i;

   for(i=0 ; i < TAMANHO ; i++)
   {
         printf("Digite a nota do aluno %d ", i+1);
         scanf("%f",&notas[i]);
         media += notas[i];
   }
   media /= 5.0;
   printf("Média das notas: %.2f\n", media);

   system("PAUSE");
   return 0;
}
```

Linguagem C – Matrizes

- Iniciando os valores da Matriz

```
/* numdias.c */
/* Imprime o número de dias do ano até a data especificada */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int dmes[12]={ 31,28,31,30,31,30,31,31,30,31,30,31};

   int dia, mes, ano, total, i;

   printf("Digite a data no formato DD/MM/AAAA: ");

   /* %*c lê caractere do teclado e não atribui a nada */
   scanf("%d%*c%d%*c%d", &dia, &mes, &ano);

   if((((!(ano%4) && ano%100)) || !(ano%400))
         dmes[1]=29;  /* Ano bissexto */

   total=dia;

   for(i=0; i < mes-1; i++)
         total+=dmes[i];

   printf("Total de dias transcorridos desde o início do ano: %d\n",
         total);

   system("PAUSE");
   return 0;
}
```

Linguagem C – Matrizes

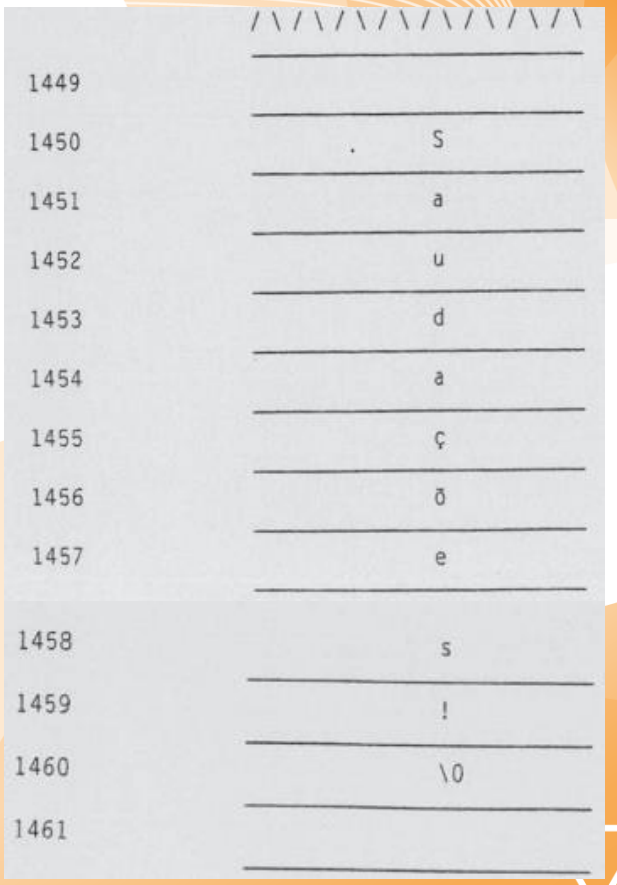
- String

O uso mais importante de matrizes é aplicado à criação de tipos de dados para armazenar e manipular textos como palavras, nomes e sentenças.

String é uma matriz do tipo **char**, que armazena um texto formado de caracteres e sempre terminado pelo caractere zero ('0'). Em outras palavras, *string* é uma série de caracteres, em que cada um ocupa um byte de memória, armazenado em seqüência e terminado por um byte de valor zero ('0'). Cada caractere é um elemento independente da matriz e pode ser acessado por meio de um índice.

```
printf("%s", "Saudações!"); /* String constante */
```

Na memória, a cadeia de caracteres "Saudações!" é armazenada da seguinte forma:



Linguagem C – Matrizes

- String - Exemplo

```
/* str1.c */
/* Mostra o uso de strings */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   char nome[80];
   printf("Digite o seu nome: ");
   scanf("%s", nome);
   printf("Saudações, %s.\n", nome);
   system("PAUSE");
   return 0;
}
```

Linguagem C – Matrizes

- Funções para manipular String - Exemplo

A FUNÇÃO scanf()

A instrução

```
scanf("%s", nome);
```

A FUNÇÃO gets()

```
/* str2.c */
/* Mostra leitura de string com gets() */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   char nome[80];
   printf("Digite o seu nome: ");
   gets(nome); /* Lê texto do teclado */
   printf("Saudações, %s:\n", nome);
   system("PAUSE");
   return 0;
}
```

A função **gets()**, como protótipo no arquivo **stdio.h**, é mais conveniente para a leitura de textos. O seu propósito é unicamente ler uma cadeia de caracteres do teclado enquanto não for pressionada a tecla [ENTER]. Todos os caracteres são armazenados na string e é incluído o caractere NULL no final.

Dúvidas

Bibliografia

- **Básica:**
  - Victorine Viviane Mizrahi - **Treinamento em Linguagem C** - 2ª Edição, 2008
  - UFMG - **Curso de. Linguagem C** , 1998

- **Complementar:**
  - Herbert Schildt - **C**, **completo e total** , Makron Books, 1996
  - TENENBAUM, A. M. et al., - **Estruturas de Dados Usando C**, São Paulo, Pearson Makron Books, 1995.


SOFTEX
PERNAMBUCO