

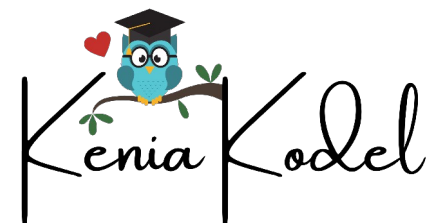


UNIVERSIDADE  
FEDERAL DE  
SERGIPE



# Tipos de Dados e Strings

## PROGRAMAÇÃO IMPERATIVA



# Tipos de Dados e Strings

## INTRODUÇÃO

De forma simplificada, são elementos de composição de um programa computacional:

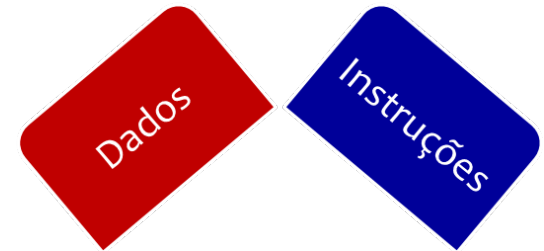


1. Instruções (provocam o processamento dos dados)
2. Dados (a serem processados)

# Tipos de Dados e Strings

## INTRODUÇÃO

```
int Cont, Num, Resultado;  
Resultado=0;  
for (Cont=1; Cont<=50; Cont=Cont+1)  
{  
printf("Digite um valor: ");  
scanf("%d", &Num);  
if (Num>Resultado)  
Resultado=Num;  
}  
printf("Resultado: %d.", Resultado);
```



Neste são trabalhados, por exemplo, os **dados** mantidos em Cont, Num e Resultado. Sobre estes são efetuadas **instruções** de leitura, escrita, de formatação, atribuição, operações aritméticas e relacionais, e de repetição.

# Tipos de Dados e Strings

## INTRODUÇÃO

Os tipos de dados definem que valores uma variável pode armazenar e as operações que podem ser efetuadas sobre estas.

Simplificadamente, podem ser:

1. Numéricos

Definem os valores que uma variável pode armazenar?!



2. Literais

3. Lógicos



Definem as operações que podem ser efetuadas sobre as variáveis?!

# Tipos de Dados e Strings

## TIPOS NUMÉRICOS

Os dados do tipo numérico são compostos por números. Podem ser:

- 1. Inteiros** – positivos ou negativos, e não apresentam componentes decimais. Exemplos de aplicação: número de pessoas, número de dias.
- 2. Reais** – positivos ou negativos, e podem ter componentes fracionários. Exemplos de aplicação: preço, comprimento, altura.

# Tipos de Dados e Strings

## TIPOS NUMÉRICOS INTEIROS

Em C, são tipos numéricos inteiros:

- **int**, valores e ocupação dependentes da arquitetura da máquina, de 2 a 4 bytes
- **short int**, valores de -32.768 a 32.767, e ocupa 2 bytes
- **long int**, valores de -2.147.483.648 a 2.147.483.647, e ocupa 4 bytes

Para saber quantos bytes o `int` ocupa, usar:  
**`printf("bytes ocupados por um inteiro: %d", sizeof(int));`**

# Tipos de Dados e Strings

## TIPOS NUMÉRICOS INTEIROS

Em C, são tipos numéricos inteiros:

- **unsigned int**, valores e ocupação dependentes da arquitetura
- **unsigned short int**, valores de 0 a 65.535, e ocupa 2 bytes
- **unsigned long int**, valores de 0 a 4.294.967.295 ocupa 4 bytes

Tem-se o tipo `int` e os modificadores `long`, `short`, `unsigned` (valores positivos e zero).

# Tipos de Dados e Strings

## COMPATIBILIDADE DE TIPOS

Uma vez compreendidos Tipos de Dados, é preciso ficar atento às atribuições:

```
int A=2.5;
```

```
char Letra='B';
```

```
float Y=Letra;
```

```
printf("%f", (float) 5/2);
```



# Tipos de Dados e Strings

## TIPOS NUMÉRICOS REAIS

São tipos numéricos com parte fracionária, também denominadas reais ou de ponto flutuante. Em C, nestes a parte inteira é separada da decimal por um ponto.

São tipos reais em C:

- **float**, de  $3.4e-38$  a  $3.4e38$ , e ocupa 4 bytes, e tem 6 casas decimais
- **double**, de  $1.7e-308$  a  $1.7e308$ , e ocupa 8 bytes, e tem 15 casas decimais
- **long double**, de  $3.4e-4932$  a  $3.4e4932$ , e ocupa 10 bytes

# Tipos de Dados e Strings

## COMPATIBILIDADE DE TIPOS

```
#include <stdio.h>
int main()
{
    int N1;
    float F1;
    printf("Digite um valor inteiro: ");
    scanf("%d", &N1);
    F1=N1;
    printf("O valor digitado em real eh: %.2f.", F1);
    return 0;
}
```



Se digitado 12?

```
Digite um valor inteiro: 12
O valor digitado em real eh: 12.00.
-----
```

# Tipos de Dados e Strings

## COMPATIBILIDADE DE TIPOS

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

```
int main()
```

```
{
```

```
    int N1;
```

```
    float F1;
```

```
    printf("Digite um valor real: ");
```

```
    scanf("%f",&F1);
```

```
    N1=F1;
```

```
    printf("A parte inteira do valor digitado eh:
```

```
    %d.",N1);
```

```
    return 0;
```

```
}
```



Se digitado 12.85?

```
Digite um valor real: 12.85
```

```
A parte inteira do valor digitado eh: 12.
```

# Tipos de Dados e Strings

## TYPE CASTING – CONVERSÃO DE TIPOS

```
#include <stdio.h>
int main() {
printf("%f", (float) 5/2);
return 0;
}
```

Type casting é uma maneira de converter uma variável de um tipo de dados para outro tipo de dados.

É possível converter os valores de um tipo para outro usando explicitamente o operador de conversão, como segue:

`(tipo) expressão`

Ou de forma implícita , como no código antes dado.

# Tipos de Dados e Strings

## TYPE CASTING – CONVERSÃO DE TIPOS

No código ao lado é usado explicitamente o operador de conversão, como segue:

(tipo) expressão

```
#include <stdio.h>
int main(){
    float F1;
    F1=(float) 7/2;
    printf("Valor: %f.",F1);
    return 0;
}
```

Com o casting, é exibido 3.500000. Sem casting, o valor 3.0000000 é atribuído à variável F1.

# Tipos de Dados e Strings

## ARREDONDAMENTO

```
#include <stdio.h>
#include <math.h>
int main(){
double I1,F1;
printf("Digite um valor real: ");
scanf("%lf",&I1);
F1=ceil(I1);
printf("O teto do valor eh: %.2lf.",F1);
return 0;
}
```

Função **ceil** – pertencente à math.h, retorna o menor inteiro (do tipo double) maior que seu argumento. Corresponde ao **teto** no arredondamento. Dado 12.85, retorna 13; dado 12.01, retorna 13.

# Tipos de Dados e Strings

## ARREDONDAMENTO

```
#include <stdio.h>
#include <math.h>
int main()
{
    double I1,F1;
    printf("Digite um valor real: ");
    scanf("%lf",&I1);
    F1=ceil(I1);
    printf("O teto do valor eh: %.2lf.",F1);
    return 0;
}
```

```
Digite um valor real: 9.01
O teto do valor eh: 10.00.
```



Pesquise sobre a  
função floor.

# Tipos de Dados e Strings

## LITERAIS

- Compostos por letras, dígitos numéricos, e/ou símbolos especiais. Exemplo: nome, endereço, matrícula, endereço de e-mail.
- São também denominados **alfanuméricos**.
- Seus valores são assinalados por aspas simples, quando char; e duplas, quando cadeias de caracteres.

Exemplos:

```
char Nome[31]= "Rick Xavier";
```

```
char Tecla= ' ';
```

```
char Matricula[10]="92332456";
```



# Tipos de Dados e Strings

## LITERAIS

Em C são implementados por meio de:

- **Char**, tipo primitivo, com capacidade para manter um único caractere;

**char** <identificador>;

- **Cadeia de caracteres** ou `string`,

**char** <identificador> [N] ;

Implementado por meio de array; com capacidade para manter N-1 caracteres alfanuméricos.

# Tipos de Dados e Strings

LITERAL: char

Para declarar uma variável do tipo char:

```
char <identificador>;
```

Exemplos:

- `char Letra;`
- `char Letra, Tecla, Opcao;`
- `char Op1, Op2, Op3;`

# Tipos de Dados e Strings

LITERAL: `char`

Para efetuar a **leitura** de dados do tipo `char`:

```
char Letra;  
scanf ("%c", &Letra) ;
```

Para efetuar a **exibição** do tipo `char`:

```
printf ("O caractere lido eh %c.", Letra) ;
```

E para efetuar a **atribuição** do tipo `char`:

```
Letra= 'a' ;
```

# Tipos de Dados e Strings

LITERAL: char

```
char Letra1, Letra2;  
printf("Digite um letra: ");  
scanf("%c",&Letra1);  
printf("Digita outra letra: ");  
scanf("%c",&Letra2);  
printf("As letras digitadas sao: %c e  
%c.",Letra1,Letra2);
```



Se digitados  
k e m?

**ERRO**


C:\Users\WM545B-T\Desktop\Estuda

```
Digite um letra: k  
Digita outra letra: As letras digitadas sao: k e  
.
```

# Tipos de Dados e Strings

LITERAL: char

```
char Letra1, Letra2;  
printf("Digite um letra: ");  
scanf("%c", &Letra1);  
printf("Digite outra letra: ");  
scanf(" %c", &Letra2);  
printf("As letras digitadas sao: %c e  
%c.", Letra1, Letra2);
```



Na leitura do primeiro caractere, o ENTER fica armazenado no buffer do teclado. A inclusão do espaço em branco antes da formatação “%c” na segunda leitura faz com que o scanf ignore (o ENTER) e retire-o do buffer. **Esta é uma possível solução!!!**

# Tipos de Dados e Strings

LITERAL: char

```
char Letra1, Letra2;  
printf("Digite um letra: ");  
scanf("%c",&Letra1);  
fflush(stdin);  
printf("Digita outra letra: ");  
scanf("%c",&Letra2);  
printf("As letras digitadas sao: %c e  
%c.",Letra1,Letra2);
```

```
Digite um letra: q  
Digita outra letra: r  
As letras digitadas sao: q e r.
```

Outra solução é usar a função `fflush` com o argumento `stdin`, a qual limpa todos os caracteres que existem no buffer do teclado.

# Tipos de Dados e Strings

LITERAL: `string`

- ✓ Uma `string` é um conjunto de caracteres.
- ✓ Em C, este (conjunto) é armazenado num vetor.

C	A	S	A	C	O	\0								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

- ✓ Para declarar uma cadeia de caracteres ou `string`:

**`char <identificador> [N];`**

# Tipos de Dados e Strings

LITERAL: `string`

Quando se trabalha com `string`, fala-se em comprimento. O comprimento N de um literal L consiste no número de caracteres apresentados por L.

Exemplos:

- ✓ `char Cidade[21]="Aracaju";` (07)
- ✓ `char Nome[31]="Rick Xavier";` (11)
- ✓ `"Manaus"` (06)
- ✓ `"M"` (01)
- ✓ `" "` (01)
- ✓ `"567"` (03)



# Tipos de Dados e Strings

LITERAL: `string`

Em C, o programador, ao definir uma variável do tipo `string`, deve sempre reservar um espaço a mais para o dígito de controle que delimita o fim da cadeia de caracteres: `'\0'`.

Assim, se se precisa de uma cadeia de caracteres para armazenar, por exemplo, um nome com 20 caracteres, deve-se declarar:

```
char Nome[21];
```

# Tipos de Dados e Strings

LITERAL: `string`

```
char Cidade[21]="Aracaju";      07
```

Neste exemplo tem-se uma variável chamada `Cidade`, do tipo `string`, com capacidade (comprimento máximo) para manter cadeia de caracteres com até 20 caracteres, que contém `"Aracaju"` e comprimento atual 07.

# Tipos de Dados e Strings

LITERAL: `string`

Se definido, por exemplo, o trecho de código:

```
char Nome[4]="1234";  
printf("A palavra eh:  
%s.",Nome);
```

É obtido como mensagem de erro:



```
[Error] Initializer-string for  
array of chars is too long
```

# Tipos de Dados e Strings

LITERAL: `string`

```
char Nome[4]="123";
```

```
printf("A palavra eh: %s.",Nome);
```



```
A palavra eh: 123.  
-----
```

# Tipos de Dados e Strings

LITERAL: `string`

Para **leitura** de variável literal do tipo `string` pode ser usado:

```
char Palavra[16];
```

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

A variável do tipo `string`, no `scanf`, nunca deve ser precedida por `&`.

A `string` lida por meio de `scanf` lê todos os caracteres até encontrar espaço em branco.

# Tipos de Dados e Strings

LITERAL: `string`

```
char Palavra[16];  
printf("Digite uma palavra: ");  
scanf("%s", Palavra);  
printf("A palavra digitada eh:  
%s", Palavra);
```

```
Digite uma palavra: Rio de Janeiro  
A palavra digitada eh: Rio
```

# Tipos de Dados e Strings

LITERAL: `string`

Para **leitura** de variável literal do tipo `string` pode ser usado:

```
char Palavra[16];
```

```
gets(Palavra);
```

Ao contrário da `scanf`, a função `gets` não está limitada à leitura de uma única palavra.

# Tipos de Dados e Strings

LITERAL: `string`

```
char Palavra[16];  
printf("Digite uma palavra: ");  
gets(Palavra);  
printf("A palavra digitada eh: %s"  
    ,Palavra);
```

```
Digite uma palavra: Rio de Janeiro  
A palavra digitada eh: Rio de Janeiro
```



# Tipos de Dados e Strings

LITERAL: `string`

Máscara para `scanf` funcionar na leitura de mais de uma palavra:

```
char Palavra[16];  
printf("Digite uma palavra: ");  
scanf("%[^\n]s", Palavra);  
printf("A palavra digitada eh:  
%s", Palavra);
```

```
Digite uma palavra: Rio de Janeiro  
A palavra digitada eh: Rio de Janeiro
```

# Tipos de Dados e Strings

LITERAL: `string`

Para **exibição** de variável literal do tipo `string` pode ser usado:

```
char Palavra[16];  
printf("%s", Palavra);
```

E também:

```
puts(Palavra);
```

# Tipos de Dados e Strings

LITERAL: `string`

Para **inicialização** de variável literal do tipo

`string` pode ser usado:

**`char`**

**`Palavra[16]={ 'J' , 'o' , 'a' , 'o' } ;`**

E também:

**`char Palavra[16]="Joao" ;`**

# Tipos de Dados e Strings

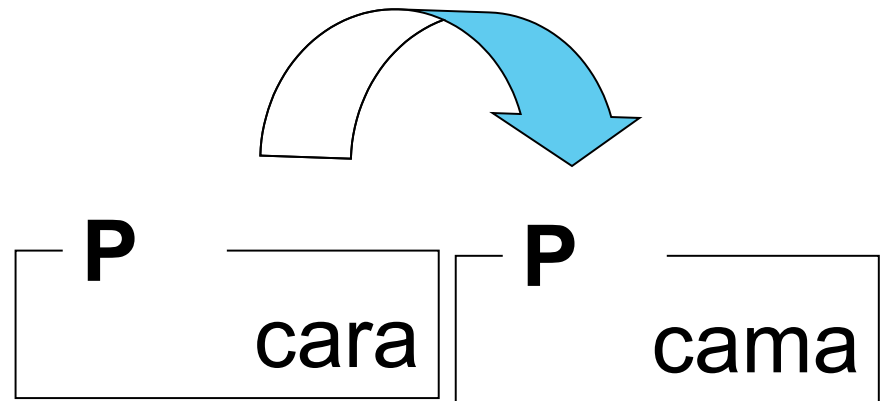
LITERAL: `string`

É possível **acessar** e **manipular** cada caractere componente de uma variável literal do tipo `string`. Para tanto deve-se usar o nome do literal seguido pela posição (inicia de 0) a ser manipulada entre colchetes:

```
char P[16]="cara";
```

```
P[2]='m' ;
```

```
puts (P) ;
```



Qual o novo valor de `P`?

# Tipos de Dados e Strings

LITERAL: `string`

Há funções de manipulação de `strings`.

**`strlen(String)`** – retorna o número de caracteres existentes (comprimento) na `string` passada como parâmetro. Faz parte da biblioteca `string.h`.

# Tipos de Dados e Strings

LITERAL: string

```
#include <stdio.h>
#include <string.h>
int main() {
    char Palavra[26];
    printf("Digite uma palavra: ");
    gets(Palavra);
    printf("Tamanho da palavra:
    %d" , strlen(Palavra));
    return 0;}
```



Se digitado Aula de  
Plimperativa?

```
Digite uma palavra: Aula de Plimperativa
Tamanho da palavra: 19
-----
```

# Tipos de Dados e Strings

## EXERCÍCIOS

1. Escrever programa para ler uma `string` `S` com até 20 caracteres e escrever cada letra de composição de `S` numa linha distinta da tela.
2. Escrever programa para ler uma `string` `S` com até 15 caracteres e um caractere `C`, e exibir a posição da primeira ocorrência de `C` em `S`. Exemplo, se dado `ROSANA` e `A`, exibir 4. **IMPORTANTE:** *Para implementação desta não pode ser usada função pré-definida de definição de primeira ocorrência de um caractere numa string.*

**EXERCÍCIO:** Escrever programa para ler uma string S com até 20 caracteres e escrever cada letra de composição de S numa linha distinta da tela.

```
#include <stdio.h>
#include <string.h>
int main(){
    //string
    char S[21];
    printf("Digite uma palavra: ");
    //leitura
    scanf("%s", S);
    //processamento
    int cont;
    for (cont=1; cont<=strlen(S); cont++)
        printf("%c \n", S[cont-1]);
    return 0;}
```



# EXERCÍCIO:

Escrever programa para ler uma string S com até 15 caracteres e um caractere C, e exibir a posição da primeira ocorrência de C em S.

```
...
//string S e caractere C
char S[16], C;
//leituras
printf("Digite uma palavra: "); scanf("%s",S);
printf("Digite um caractere: "); scanf(" %c",&C);
//processamento
int cont, pos=-1;
for (cont=0; cont<=strlen(S)-1; cont++)
    if (S[cont]==C){
        pos=cont;
        break;}
//saída
if (pos==-1)
    printf("Nao ha ocorrencia de %c em %s.",C,S);
else
    printf("A 1a ocorrencia de %c em %s eh: %d.",C,S,pos+1);
...
```

# Tipos de Dados e Strings

LITERAL: `string`

Para **atribuição** de variáveis literais do tipo `string` não pode ser usado a instrução `=`, deve ser usado:

**`strcpy (SDestino, SOrigem) ;`**

Assim, atribui-se o valor da variável `SOrigem` à variável `SDestino`. Esta também faz parte da biblioteca `string`.

```
char P1[16], P2[16];
```

```
gets(P1);
```

```
strcpy(P2, P1);
```

```
puts(P2);
```

**P1** uai



Qual o valor exibido?

# Tipos de Dados e Strings

LITERAL: `string`

Para **concatenação** (junção) de variáveis literais do tipo `string` deve ser usado:

**`strcat (SDestino, SOrigem) ;`**

Assim, une-se ao valor da variável `SDestino`, o valor da variável `SOrigem`.

```
char P1[16]="data", P2[16]=" do teste4?";
```

```
strcat (P1, P2) ;
```

```
puts (P1) ;
```



Qual o valor exibido?

# Tipos de Dados e Strings

## EXERCÍCIO

(3) Escrever programa para ler nome e sobrenome de um autor, compor e exibir uma string composta por: SOBRENOME, Nome. Exemplo, dado: pedro cabral, dever ser composta a string e exibida: CABRAL, Pedro.

# EXERCÍCIO

(3)

Escrever

programa para ler nome e sobrenome de um autor, compor e exibir uma string composta por: SOBRENOME, Nome. Exemplo, dado: pedro cabral, dever ser composta a string e exibida: CABRAL, Pedro.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
int main(){
    char Nome[21], Sobrenome[21]; //string
    printf("Nome: ");
    gets(Nome);
    printf("Sobrenome: ");
    gets(Sobrenome); //leituras
    int L;
    for (L=1; L<strlen(Nome); L++)
        Nome[L]=tolower(Nome[L]);
    Nome[0]=toupper(Nome[0]); //nome com inicial maiúscula
    for (L=0; L<strlen(Sobrenome); L++)
        Sobrenome[L]=toupper(Sobrenome[L]);
    //sobrenome em maiúscula
    char Resultado[41]="";
    strcat(Resultado,Sobrenome);
    strcat(Resultado,", ");
    strcat(Resultado,Nome); //composição da saída
    printf("Resultado: %s.\n",Resultado); //saída
    return 0;}
```

# Tipos de Dados e Strings

LITERAL: `string`

A função **`strcmp(S1, S2)`** é utilizada para comparar duas strings alfabeticamente:

- Retorna  $<0$  se S1 for menor que S2
- Retorna 0 se S1 e S2 forem iguais
- Retorna  $>0$  se S1 for maior que S2

Obs: Maiúsculas são tratadas como letras distintas.

algo  
amarelo  
Retorna -1

azul  
azul  
Retorna 0

boi  
berço  
Retorna 1

# Tipos de Dados e Strings

## EXERCÍCIO

(4) Escrever programa para ler três palavras distintas entre si com até 20 caracteres e exibí-las ordenadas alfabeticamente.

# EXERCÍCIO (4) Escrever

programa para ler três palavras distintas entre si com até 20 caracteres e exibi-las ordenadas alfabeticamente.

```
#include <stdio.h>
#include <string.h>
int main(){
    char P1[21], P2[21], P3[21];
    printf("Digite 3 palavras: ");
    scanf("%s%s%s",P1,P2,P3);
    if (strcmp(P1,P2)<0 && strcmp(P1,P3)<0)
        {if (strcmp(P2,P3)<0)
            printf("A ordem eh %s, %s e %s.",P1,P2,P3);
        else
            printf("A ordem eh %s, %s e %s.",P1,P3,P2);}
    if (strcmp(P2,P1)<0 && strcmp(P2,P3)<0)
        {if (strcmp(P1,P3)<0)
            printf("A ordem eh %s, %s e %s.",P2,P1,P3);
        else
            printf("A ordem eh %s, %s e %s.",P2,P3,P1);}
    if (strcmp(P3,P2)<0 && strcmp(P3,P1)<0)
        {if (strcmp(P2,P1)<0)
            printf("A ordem eh %s, %s e %s.",P3,P2,P1);
        else
            printf("A ordem eh %s, %s e %s.",P3,P1,P2);}
    return 0;}
```



# EXERCÍCIO (4) Escrever

programa para ler três palavras distintas entre si com até 20 caracteres e exibi-las ordenadas alfabeticamente.

## OUTRA SOLUÇÃO

```
#include <stdio.h>
#include <string.h>
int main(){
    char P1[21], P2[21], P3[21], Aux[21];
    printf("Digite 3 palavras: ");
    scanf("%s%s%s",P1,P2,P3);
    //ordenacao com deslocamento
    if (strcmp(P2,P1)<0){
        strcpy(Aux,P1);
        strcpy(P1,P2);
        strcpy(P2,Aux);}
    if (strcmp(P3,P2)<0){
        strcpy(Aux,P2);
        strcpy(P2,P3);
        strcpy(P3,Aux);}
    if (strcmp(P2,P1)<0){
        strcpy(Aux,P1);
        strcpy(P1,P2);
        strcpy(P2,Aux);}
    printf("A ordem eh %s, %s e %s.\n",P1,P2,P3);
    return 0;}
```

# Tipos de Dados e Strings

## EXERCÍCIO

Sabendo-se que `Y=toupper (X)` corresponde a instrução em C (biblioteca `ctype.h`) que atribui à variável `Y` do tipo `char` o valor da variável `X` (também do tipo `char`) em maiúsculo.

(5) Escrever programa para ler uma palavra com até 20 caracteres e exibi-la com letras maiúsculas, sem fazer uso de função pré-definida que efetua tal conversão (das letras de uma palavra).

# Tipos de Dados e Strings

## EXERCÍCIO

(6) Escrever programa para ler um nome e, independente de como este foi escrito, exibir a inicial maiúscula e o restante minúscula. Sendo:

- `Cadeia[P] = toupper(Cadeia[P]);`  
Converte a letra da posição P da string Cadeia para maiúscula, da biblioteca `ctype.h`.
- `Cadeia[P] = tolower(Cadeia[P]);`  
Converte a letra da posição P da string Cadeia para minúscula, da biblioteca `ctype.h`.

# Tipos de Dados e Strings

## TIPO LÓGICO

As variáveis do tipo lógico podem assumir dois valores: verdadeiro ou falso.

São também denominadas **booleanas**.

São usadas, por exemplo, para controle de fluxo.

# Tipos de Dados e Strings

## TIPO LÓGICO

Em C não existe nenhum tipo de dado específico para armazenar valores lógicos.

Em C o valor lógico FALSO é representado por zero; e o VERDADEIRO, por tudo que é diferente de zero.

Assim, as expressões relacionais em C são avaliadas em 0 (falsas) ou 1 (verdadeiras).

# Tipos de Dados e Strings

## TIPO LÓGICO

As expressões relacionais em C são avaliadas em 0 (falsas) ou 1 (verdadeiras).

```
int A=8, B=45, C=8;
```

```
printf("Valor lógico de A==B? %d", A==B); 0
```

```
printf("Valor lógico de A==C? %d", A==C); 1
```

```
printf("Valor lógico de A>=B? %d", A>=B); 0
```

```
printf("Valor lógico de A>=C? %d", A>=C); 1
```

```
printf("Valor lógico de A<B? %d", A<B); 1
```

```
printf("Valor lógico de A!=C? %d", A!=C); 0
```

# Tipos de Dados e Strings

## EXERCÍCIO

(7) [Adaptado do The Huxley] Nos parques de diversão, para que um usuário possa fazer uso de alguns brinquedos deve atender idade e altura mínimas. O parque Ambrolândia possui 5 brinquedos que possuem essas limitações:

- Barca Viking: 1,5m de altura e 12 anos.
- Elevator of Death: 1,4m de altura e 14 anos.
- Final Killer: 1,7m de altura ou 16 anos.
- Trem Fantasma: 1,35m de altura e 8 anos.
- Montanha Russa: 1,6m ou 12 anos

Dada a altura e a idade de uma pessoa P, faça um programa que identifique em quantos brinquedos P pode brincar.

# Tipos de Dados e Strings

## EXERCÍCIO

- Barca Viking: 1,5m de altura e 12 anos.
- Elevator of Death: 1,4m de altura e 14 anos.
- Final Killer: 1,7m de altura ou 16 anos.
- Trem Fantasma: 1,35m de altura e 8 anos.
- Montanha Russa: 1,6m ou 12 anos

Dada a altura e a idade de uma pessoa P, faça um programa que identifique em quantos brinquedos P pode brincar.

```
#include <stdio.h>
/* Barca Viking: 1,5m de altura e 12 anos.
Elevator of Death: 1,4m de altura e 14 anos.
Final Killer: 1,7m de altura ou 16 anos.
Trem Fantasma: 1,35m de altura e 8 anos.
Montanha Russa: 1,6m ou 12 anos*/
int main(){
    int Altura, Idade, Quant=0;
    printf("Digite altura e idade: ");
    scanf("%d%d",&Altura,&Idade);
    Quant=Quant + (Altura>=150 && Idade>=12);
    Quant=Quant + (Altura>=140 && Idade>=14);
    Quant=Quant + (Altura>=170 || Idade>=16);
    Quant=Quant + (Altura>=135 && Idade>=8);
    Quant=Quant + (Altura>=160 || Idade>=12);
    printf("Pode brincar em %d brinquedos.\n",Quant);
    return 0;}
```



# Tipos de Dados e Strings

## OUTROS TIPOS

Há ainda outros tipos de dados:

- Ponteiros (pointer)
- Conjuntos (set)
- Registros (record)
- Vetores (arrays)

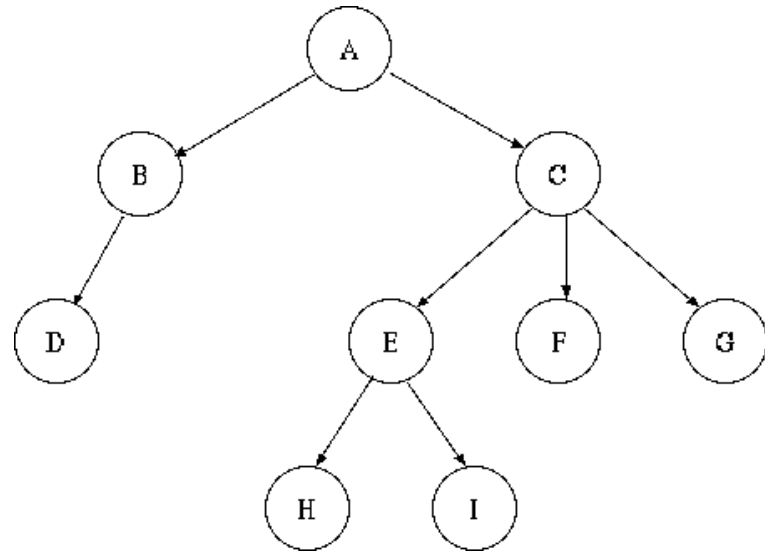
Índices	0	1	2	3	4	5	6
x	10	20	9	7	8	30	5

# Tipos de Dados e Strings

## OUTROS TIPOS

Há ainda outros tipos de dados:

- **Pilhas**
- **Filas**
- **Árvores**



# Tipos de Dados e Strings

## EXERCÍCIO

(8) Escrever programa para ler uma palavra P qualquer com até 20 caracteres e identificar quantas vogais P apresenta, por meio de uma função F (a identificação da quantidade de vogais de uma dada palavra e vogal).

(9) Escrever programa para ler uma palavra P qualquer com até 20 caracteres e identificar quantas não-vogais P apresenta.

# Tipos de Dados e Strings

## EXERCÍCIO

(10) Escrever programa para ler o nome de uma pessoa, composto por nome e sobrenome (entre nome e sobrenome há um espaço em branco). E exibir, sobrenome em maiúsculo, seguido de vírgula e do nome da pessoa.

# Tipos de Dados e Strings

## EXERCÍCIO

(11) Na universidade “DuSabiTudu” os alunos são identificados por matrículas compostas por 10 dígitos. Os 4 primeiros dígitos correspondem ao ano em que o aluno entrou na instituição. Assim, todos os alunos cujas matrículas iniciam por 2012 ingressaram no curso superior no ano de 2012. Os dígitos 5 e 6 da matrícula correspondem ao curso; sendo: 97 para agroecologia e 99 para gerontologia. Escrever programa para ler a matrícula de 100 alunos inscritos no programa “CienciasAlémDasFronteiras” e identificar quantos alunos ingressaram num dado ano A no curso de agroecologia. Aplicar função.

# Programação Imperativa

## COMPLEMENTAR AULA...

### Fundamentos da Programação de Computadores

*Ana Fernanda Gomes Ascencio*

*Edilene Aparecida Veneruchi de Campos*

#### Capítulos

Conceitos Básicos

Manipulação de Cadeia de Caracteres



# Programação Imperativa

COMPLEMENTAR AULA...

Curso de Linguagem C  
UFMG

*linux.ime.usp.br/  
~lucasmmg/livecd/  
documentacao/  
documentos/  
curso\_de\_c/  
www.ppgia.pucpr.br/  
\_maziero/ensino/so/  
projetos/curso-c/c.html*

## Aula 5

Matrizes e Strings

[Aula 1: Introdução e Sumário](#)

[Aula 2 - Primeiros Passos](#)

[Aula 3 - Variáveis, Constantes, Operadores e Expressões](#)

[Aula 4 - Estruturas de Controle de Fluxo](#)

[Aula 5 - Matrizes e Strings](#)

[Aula 6 - Ponteiros](#)

[Aula 7 - Funções](#)



# Programação Imperativa

## PRÓXIMO PASSO



Vetores