Aula 02 - Revisão C: Tipos de Dados

Prof. Me. Claudiney R. Tinoco

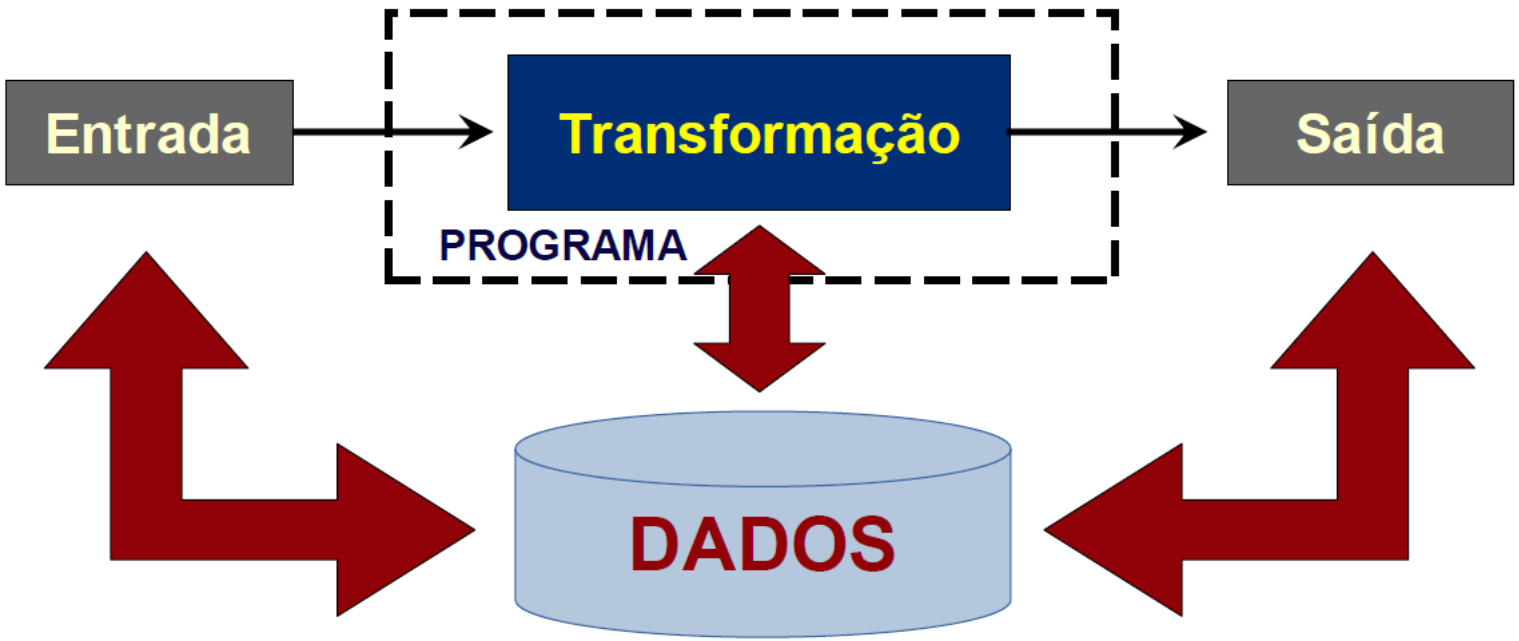
profclaudineytinoco@gmail.com

Faculdade de Computação (FACOM)
Bacharelado em Ciência da Computação (BCC)
Bacharelado em Sistemas de Informação (BSI)

Algoritmos e Estruturas de Dados 1 (AED1)
GBC024 - GSI006

Introdução

Processamento de Dados



- O computador é uma **ferramenta** que permite o processamento de dados

Importância dos Dados

- Variáveis são usadas em várias partes do algoritmo:

TIPO	EXEMPLO
Entrada de dados	`scanf("%d", &idade);`
Saída de dados	`printf("Acertos=%d", qtde);`
Armazenar resultado/valor (atribuição)	`area = base * altura;`

Acumuladores/Contadores	`cont++; //Conta`
	`soma = soma + num; //Acumula`

Tomada de decisão (seleção/repetição)	`if ( media >= 60 )`
Sinalizadores/Flags	`Atualizado = 1; // True`

Processamento de Dados

- Programas só reconhecem dados **armazenados na memória principal**
  - Dados estão fixos no código (**constantes**) ou armazenados em alguma posição da memória (**variáveis**)
  - Instruções geralmente envolvem **movimentação e/ou transformação desses dados**

- Ao conceber um algoritmo, podemos considerar a abstração:

Memória = coleção de caixas

  - Cada caixa possui as seguintes características:
    - Tem um identificador (**nome**)
    - Sempre armazena valor
    - Possui uma **posição exclusiva** na memória

Abstração da Memória

Abstração da Memória

IDENTIFICADOR	IDADE	NOME	X1
VALOR	18	"João"	2.5

- As "caixas" de memória (variáveis) seguem as premissas:
  - Ao atribuir um novo valor a uma posição da memória, o seu valor atual será substituído (**perdido**)
  - Mesmo sem qualquer atribuição explícita, uma posição de memória **sempre possui algum conteúdo** (chamado de **"lixo"**)

Tipos de Dados

- **Tipos primitivos:**
  - São as representações mais elementares (**simples**) dos dados disponibilizados em uma linguagem de programação
  - Armazena um único valor por variável
  - **Ex:** int, char e float

- **Tipos estruturados:**
  - Possibilita **estruturar dados complexos**
  - **Coleção de dados** relacionados a um único objeto
    - Composta por tipos primitivos e/ou outros tipos estruturados
    - **Ex:** ponto, alunos, notas, faturamento mensal, etc.
  - Podem ser **homogêneas** ou **heterogêneas**

Tipos Estruturados (**Estruturas Homogêneas**)

- Estruturas de dados compostas por **elementos do mesmo tipo de dado** (arranjos ou *arrays*)

- **Vetores:** estrutura linear que suporta *N* posições distribuídas sequencialmente
  - **Ex:** `char nome[100];`

- **Matrizes:** estrutura espacial que suporta *N x M* posições
  - **Ex:** `int mat_adj[5][5];`

Vetores/Arrays

Vetores

- Array ou Vetor é a forma mais comum de dados estruturados.
- Um vetor (ou *array*) é um conjunto de componentes do mesmo tipo.
  - Ex: 12 números inteiros, cada um representando um mês do ano; 120 booleanos para indicar o estado de ocupação de quartos de um hotel; 2 números reais para o grau de miopia de uma pessoa; etc..
- Um vetor é um **tipo de dado** estruturado, isto é, existe uma relação estrutural entre seus valores. Os tipos de dados simples são tipos **elementares** (caracter, real, inteiro, logico).
  - Um vetor é formado pela composição por tipos elementares ou de outros tipos estruturados

Array – Declaração em C

- Arrays são agrupamentos de dados adjacentes na memória. Declaração:
  - tipo_dado nome_array[tamanho];
- O comando acima define um array de nome **nome_array,** capaz de armazenar **tamanho** elementos adjacentes na memória do tipo **tipo_dado**

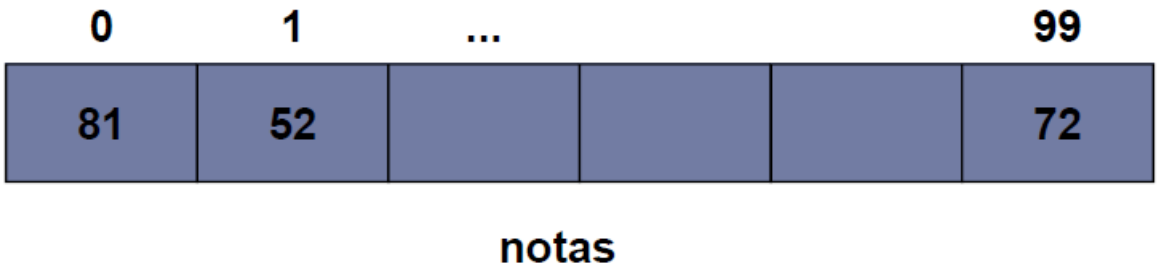
  - Ex: `double notas[5];`

Array - Definição

- As variáveis têm relação entre si
  - todas armazenam notas de alunos
- Podemos declará-las usando um ÚNICO nome para todos os 100 alunos
  - notas = conjunto de 100 números acessados por um índice = array.

Array - Definição

- Na linguagem C a numeração do array começa sempre do zero.
- Isto significa que, no exemplo anterior, os dados serão indexados de 0 a 99.
  - notas[0], notas[1], ..., notas[99]



Vetores – Índice Inválido

- Observação
  - Se o usuário digitar mais de 100 elementos em um array de 100 elementos, o programa tentará ler normalmente.
  - Porém, o programa os armazenará em uma parte não alocada de memória, pois o espaço alocado foi para somente 100 elementos.
  - Isto pode resultar nos mais variados erros no instante da execução do programa.

Array = variável

- Cada elemento do array tem todas as características de uma variável e pode aparecer em expressões e atribuições.
  - notas[2] = notas[3] + notas [20]
- Ex: somar todos os elementos de notas:

```
int soma = 0;
for(i=0;i < 100; i++)
   soma = soma + notas[i];
```

Faça um programa para ler 5 números e mostrar o resultado da soma desses números

```c
int main()
{
    double val, soma;
    int contador;

    soma = 0; // inicializando o valor de soma

    contador = 1; // inicializando o contador

    while (contador <= 5){
        printf("\nDigite o %do. numero: ", contador);
        scanf("%lf", &val);
        soma = soma + val;
        contador = contador + 1;
    }

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

Mudando o problema

- Faça um programa para ler 5 números e mostrar, após a leitura de todos os números, os números lidos juntamente com resultado da soma desses números
  - Na solução anterior, a cada passo do loop o valor lido era sobrescrito pelo próximo passo

```
while (contador <= 5){
        printf("\nDigite o %do. numero: ", contador);
        scanf("%lf", &val); // sobrescreve
        soma = soma + val;
        contador = contador + 1;
    }
```

  - Solução: Utilize loop e vetor, indexando o vetor com o contador do loop

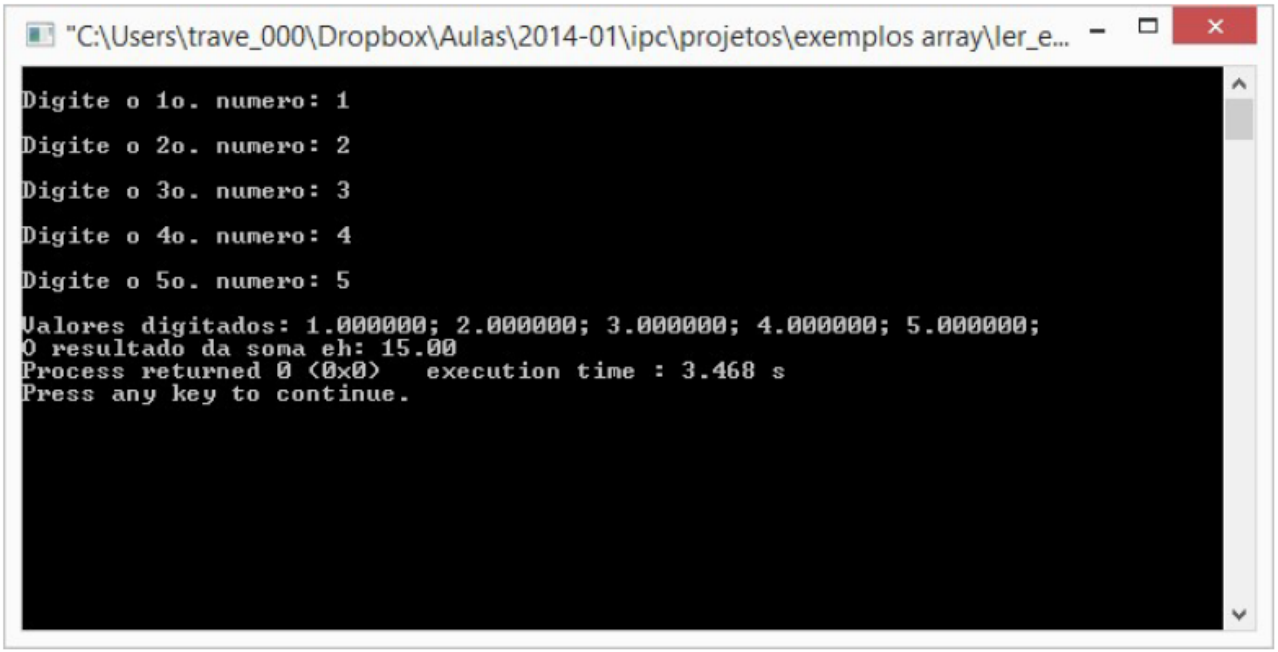
Solução I

```c
double val[5], soma;
int contador;


soma = 0; // inicializando o valor de soma

contador = 0; // iniciando o contador

while (contador < 5){
    printf("\nDigite o %do. numero: ", contador + 1);
    scanf("%lf", &val[contador]);
    soma = soma + val[contador];
    contador = contador + 1;
}

contador = 0;
printf("\nValores digitados: ");
while (contador < 5){
    printf("%f; ", val[contador]);
    contador = contador + 1;
}
printf("\nO resultado da soma eh: %.2f", soma);
return 0;
```

Crio o vetor `val[5]` para leitura de 5 double

Leio o valor em cada posição do vetor `val[contador]`

Resultado


"C:\Users\trave_000\Dropbox\Aulas\2014-01\ipc\projetos\exemplos array\ler_e...
Digite o 1o. numero: 1
Digite o 2o. numero: 2
Digite o 3o. numero: 3
Digite o 4o. numero: 4
Digite o 5o. numero: 5
Valores digitados: 1.000000; 2.000000; 3.000000; 4.000000; 5.000000;
O resultado da soma eh: 15.00
Process returned 0 (0x0)   execution time : 3.468 s
Press any key to continue.

Solução 2

- É melhor utilizar loop **for**, pois o número de iterações é conhecido

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

int main()
{
    double val[5], soma;
    int i;

    // inicializando o valor de soma
    soma = 0;
    for (i = 0; i < 5; i++) {
        printf("\nDigite o %do. numero: ", i+1);
        scanf("%lf", &val[i]);
        soma = soma + val[i];
    }

    printf("\nValores digitados: ");
    for (i = 0; i < 5; i++)
        printf("%f; ", val[i]);

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

Solução 2

- E se for para somar 100 números?

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

int main()
{
    double val[100], soma;
    int i;

    // inicializando o valor de soma
    soma = 0;
    for (i = 0; i < 100; i++) {
        printf("\nDigite o %do. numero: ", i+1);
        scanf("%lf", &val[i]);
        soma = soma + val[i];
    }

    printf("\nValores digitados: ");
    for (i = 0; i < 100; i++)
        printf("%f; ", val[i]);

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

Exercício 1

- Exercício
  - Para um array A com 5 números inteiros, formular um algoritmo que determine o maior elemento deste array.

Exercício 1

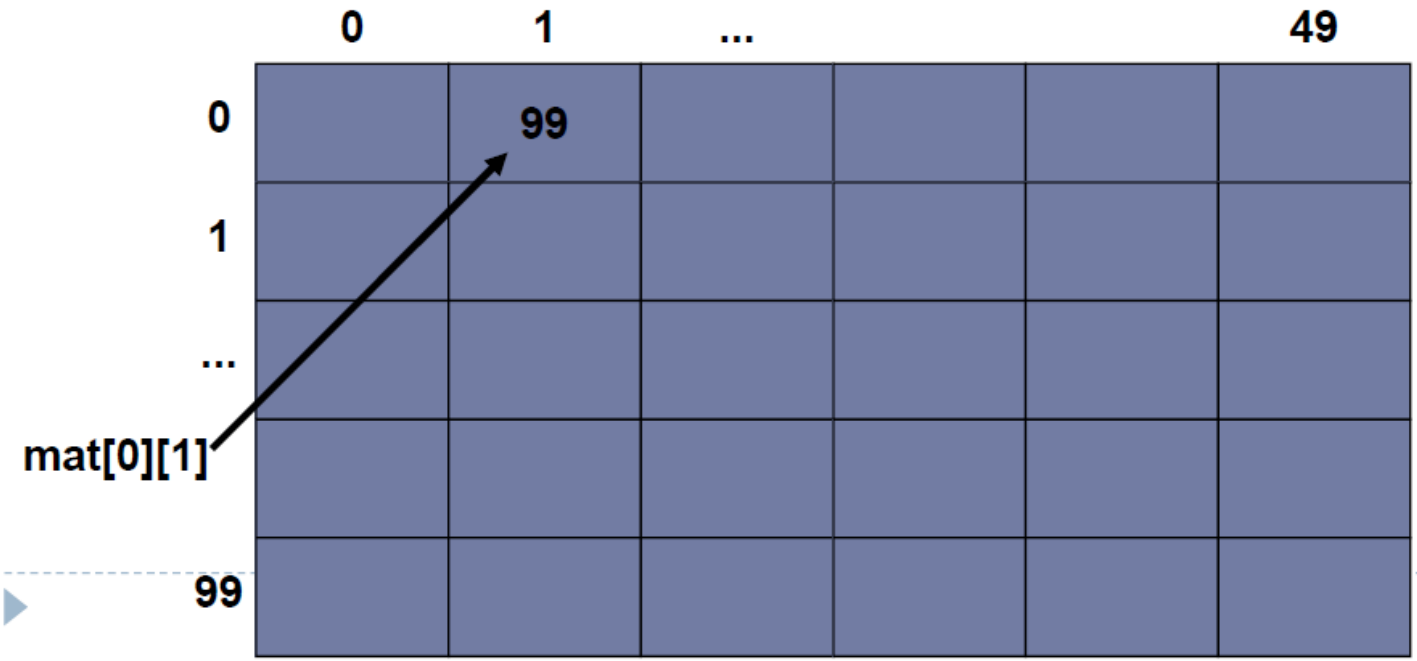
```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
  // declarando o vetor e inicializando seus elementos
  int A[5] = {3,18,2,51,45};
  int N = 5,i;

  int Maior = A[0];

  for(i = 1; i < N; i++){
    if (Maior < A[i])
        Maior = A[i];
  }
  printf("O maior valor eh: %d", Maior);
  return 0;
}
```

Vetores bidimensionais/Matrizes

- Também chamados de "matrizes", contém:
  - arranjados na forma de uma tabela de 2 dimensões;
  - necessita de dois índices para acessar uma posição: um para a linha e outro para a coluna
  - Índices começam sempre na posição ZERO.
- Declaração
  - tipo_variável nome_variável[linhas][colunas];

Arrays bidimensionais - matrizes

- Ex.: um array que tenha 100 linhas por 50 colunas
  - int mat[100][50];
  - mat[0][1] = 99;



Arrays bidimensionais - matrizes

- Como uma matriz possui dois índices, precisamos de dois comandos de repetição para percorrer todos os seus elementos.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    int mat[100][50];
05    int i,j;
06    for (i = 0; i < 100; i++){
07      for (j = 0; j < 50; j++){
08        printf("Digite o valor de mat[%d][%d]: ",i,j);
09        scanf("%d",&mat[i][j]);
10      }
11    }
12    system("pause");
13    return 0;
14  }
```

Arrays Multidimensionais

- Arrays podem ter diversas dimensões, cada uma identificada por um par de colchetes na declaração
  - int vet[5]; // 1 dimensão
  - float mat[5][5]; // 2 dimensões
  - double cub[5][5][5]; // 3 dimensões
  - int X[5][5][5][5]; // 4 dimensões

- Um array N-dimensional funciona basicamente como outros tipos de array. Basta lembrar que o índice que varia mais rapidamente (contíguo na memória) é o índice mais à direita.
  - int vet[5]; // 1 dimensão
  - float mat[5][5]; // 2 dimensões
  - double cub[5][5][5]; // 3 dimensões
  - int X[5][5][5][5]; // 4 dimensões

Exercício 1

- Dado um array A de 3x5 elementos inteiros, calcular a soma dos seus elementos.
- Exemplo

1	5	0	0	3
2	3	7	0	0
0	0	2	1	2

- Soma = 1 + 5 + 0 + 0 + 3 + 2 + 3 + 7 + 0 + 0 + 0 + 0 + 2 + 1 + 2 = 26

Exercício 1

```
int soma = 0;
int A[3][5];
int i,j;

for(i=0;i<3;i++){
  for(j=0;j<5;j++){
   soma = soma + A[i][j];
  }
}
printf("%d", soma);
```

Exercício 2

- Dado duas matrizes reais de dimensão 2x3, fazer um programa para calcular a soma delas.
  - Exemplo de como é a soma de duas matrizes

$$\begin{bmatrix} 1 & 3 & 2 \\ 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 & 2+5 \\ 1+7 & 0+5 & 0+0 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 7 \\ 8 & 5 & 0 \end{bmatrix}$$

Exercício 2

```
float A[2][3], B[2][3], Soma[2][3];
int i,j;

// << suponha comandos de leitura de A e B aqui >>

for(i=0;i<2;i++){
  for(j=0;j<3;j++){
    Soma[i][j] = A[i][j] + B[i][j];
  }
}
```

Inicialização

- Arrays podem ser inicializados com certos valores durante sua declaração. A forma geral de um array com inicialização é:
  - tipo_variável nome_variável [tam1][tam2]...[tamN]={lista_valores};

- A lista de valores é composta por valores (do mesmo tipo da variável) separados por vírgula. Os valores devem ser dados na ordem em que serão colocados na matriz.

```
float vet[6] = { 1.3, 4.5, 2.7, 4.1, 0.0, 100.1 };
int mat[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
int mat[3][4] = { {1, 2, 3, 4},{5, 6, 7, 8}, {9, 10, 11, 12}};
char str[10] = { 'J', 'o', 'a', 'o', '\0' };
char str[10] = "Joao";
char nomes[3][10] = { "Joao", "Maria", "Jose" };
```

- int mat[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };

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

int main()
{
   int mat[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
   int i,j;

   for (i = 0; i < 3; i++){
     for (j = 0; j < 4; j++){
        printf("%d\t",mat[i][j]);
     }
     printf("\n");
   }

   return 0;
}
```

```
"D:\Dropbox\Aulas\2014-01\ipc\projetos\exemplos array\ordem_matriz\bi
1       2       3       4
5       6       7       8
9       10      11      12

Process returned 0 (0x0)   execution time : 0.359 s
Press any key to continue.
```

- Observe os endereços das variáveis

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

int main()
{
   int mat[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
   int i,j;

   for (i = 0; i < 3; i++){
     for (j = 0; j < 4; j++){
        printf("%d\t",&mat[i][j]);
     }
     printf("\n");
   }

   return 0;
}
```

```
"D:\Dropbox\Aulas\2014-01\ipc\projetos\exemplos array\ordem_matriz\bin\D
2686696 2686700 2686704 2686708
2686712 2686716 2686720 2686724
2686728 2686732 2686736 2686740

Process returned 0 (0x0)   execution time : 1.520 s
Press any key to continue.
```

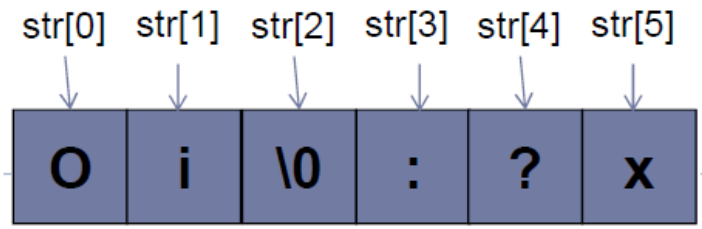
Strings

String

- Seqüência de caracteres adjacentes na memória.
- Em outras palavras, strings são arrays do tipo **char**.

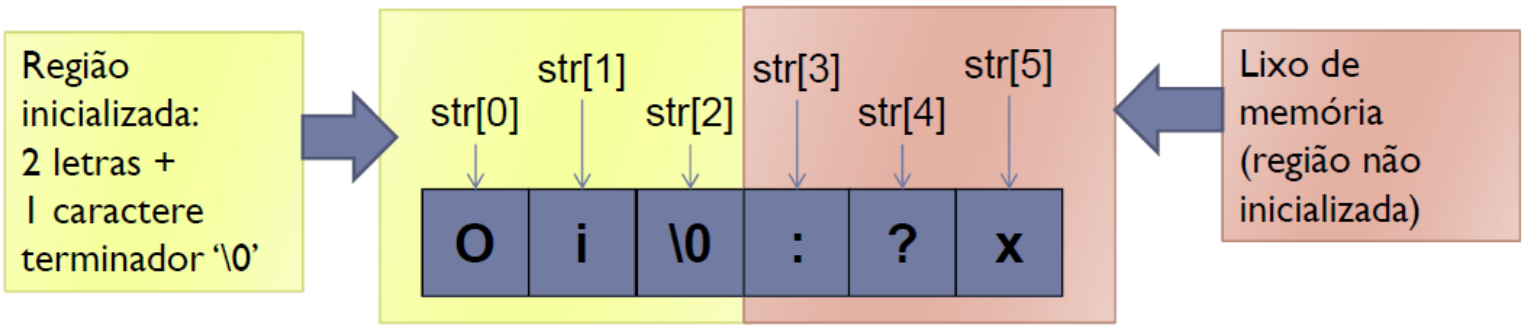
- Ex:
  - char str[6];

  - Devemos ficar atentos para o fato de que as strings têm no elemento seguinte a última letra da palavra/frase armazenada, um caractere '**\0**' (barra invertida + zero).
  - O caractere '**\0**' indica o fim da sequência de caracteres.
- Ex: char str[6] = "Oi";



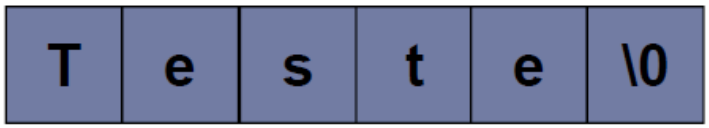
String

- Ex: `char str[6] = "Oi";`



String

- **Importante**
  - Ao definir o tamanho de uma string, devemos considerar o caractere '\0'.
  - Isso significa que a string **str** comporta uma palavra de no máximo 5 caracteres.
- Ex: char str[6] = "Teste";



- Por se tratar de um array, cada caractere podem ser acessados individualmente por indexação

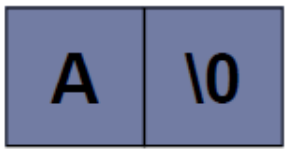
String

- IMPORTANTE:
  - Na inicialização de palavras, usa-se **"aspas duplas"**.
    - Ex: char str[6] = "Teste";

  - Na atribuição de caracteres, usa-se **'aspas simples'**
    - str[0] = 'L';

String

- IMPORTANTE:
  - "A" é muito diferente de 'A'
  - "A"



  - 'A'

Manipulando strings

- Strings são arrays. Portanto, **não** se pode atribuir uma string para outra!

```
char str1[10]
char str2[10] = "Ola";
str1 = str2; //Erro!!!
```

- O correto é copiar a string elemento por elemento.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    char str1[20] = "Hello World";
05    char str2[20];
06
07    str1 = str2; //ERRADO!
08
09    system("pause");
10    return 0;
11  }
```

Copiando uma string

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    int i;
05    char str1[20] = "Hello World";
06    char str2[20];
07    for (i = 0; str1[i]!='\0'; i++)
08       str2[i] = str1[i];
09    str2[i] = '\0';
10    system("pause");
11    return 0;
12  }
```

Manipulando strings

- Felizmente, a biblioteca padrão C possui funções especialmente desenvolvidas para esse tipo de tarefa
  - #include <string.h>

  - gets(str): lê uma string do teclado e coloca em str.
  - strlen(str): retorna o tamanho da string str.
  - strcpy(dest, fonte):copia a string contida na variável **fonte** para **dest**.
  - strcat(dest, fonte): concatena duas strings. Nesse caso, a string contida em **fonte** permanecerá inalterada e será anexada ao fim da string de **dest.**
  - strcmp(str1, str2): compara duas strings. Nesse caso, se as strings forem iguais, a função retorna ZERO.

Tipos Estruturados (Estruturas Heterogêneas)

- Estruturas de dados formadas por *K* **elementos de diferentes tipos de dados**
- Os elementos são denominados **campos da estrutura**
  - Cada campo tem um tipo de dado próprio

- **Exemplo:**

```
struct funcionario {
    char nome[100];
    int idade;
    float salario;
};
```

Estruturas de Dados

- Podem ser vistas como um **novo tipo de dado** formado pela composição de outros tipos (**tipo heterogêneo - struct**)
  - **Representação lógica** de um objeto/elemento do problema
- Pode ser declarada em qualquer escopo (local ou global)
  - Declaração = definição do novo tipo
  - Declaração da struct ≠ declaração da variável (não aloca memória)
- **Sintaxe da declaração:**

```
struct nomestruct {
    tipo1 campo1;
    tipo2 campo2;
    ...
    tipoN campoN;
};
```

Define os dados que compõem a estrutura

Estrutura de Dados

- Exemplo de agrupamento de dados:
  - Cadastro de pacientes

```
int  idade;
char nome[10] = "Maria";
double peso, altura;
int estado_civil;
float grau_miopia[2];
```

Todas essas informações são do mesmo paciente, portanto, podemos agrupá-las. Isso facilita também lidar com dados de outros pacientes no mesmo programa (**organização na memória**)

Estrutura de Dados

- Exemplo de agrupamento de dados:
  - Cadastro de pacientes

```
int  idade;
char nome[10] = "Maria";
double peso, altura;
int estado_civil;
float grau_miopia[2];
```

```
struct dados_pacientes {
    int  idade;
    char nome[10];
    double peso;
    double altura;
    int estado_civil;
    float grau_miopia[2];
};
```

Declaração de Variáveis

- Uma vez definida a estrutura, uma variável pode ser declarada de modo similar aos tipos já existentes:
  - `struct dados_pacientes paciente1;`

- Obs: por ser um tipo definido pelo programador, a palavra **struct** deve anteceder o tipo da nova variável

Declaração de Variáveis

- Declaração de uma variável do tipo struct:
  - `struct dados_pacientes paciente1;`

Tipo de dado — Variável

- Declaração de uma variável inteira:
  - `int a;`

Tipo de dado — Variável

Exercício

- Declare uma estrutura capaz de armazenar a matrícula (nro inteiro) e 3 notas para um dado aluno.

Estruturas

- O uso de estruturas facilita na manipulação dos dados do programa

- **Ex:** declaração do cadastro de 4 pacientes diferentes poderia ser feita por:

```
char nome1[10], nome2[10], nome3[10], nome4[10];
int idade1, idade2, idade3, idade4;
double grau_miopia1[2],grau_miopia2[2],grau_miopia3[2],grau_miopia4[2];
```

OU

```
char nome[4][10];
int idade[4];
double grau_miopia[4][2];
```

- Usando struct, a declaração pode ser feita por:

```
// Declaração do tipo de dados (struct)
struct dados_pacientes {
   int  idade;
   char nome[10];
   double peso;
   double altura;
   int estado_civil;
   float grau_miopia[2];
};

// Declaração da variável (vetor de pacientes)
struct dados_pacientes pacientes[4];
```

Acesso aos Campos da Estrutura

- Como é feito o acesso aos campos de uma variável do tipo struct?

- **R:** usar o operador ponto "." para indicar o campo a ser acessado

- **Exemplo:**

```
// declarando a variável da struct
struct dados_pacientes cliente_especial;

// acessando os campos da struct
cliente_especial.idade = 18;
cliente_especial.peso= 80.5;
strcpy(cliente_especial.nome, "João");
```

Inicialização de Variáveis Estruturadas

- Assim como nos arrays, uma variável struct pode ser inicializada na sua declaração:

```
struct ponto {
    int x;
    int y;
};

struct ponto p1 = { 220, 110 };
```

Leitura de Variáveis Estruturadas

- Como ler os valores dos campos de uma variável struct do teclado?
- **R:** Ler cada variável independentemente, respeitando seus respectivos tipos
- **Exemplo:**

```
gets(cliente_especial.nome);//string
scanf("%d",&cliente_especial.idade);//int
scanf("%f",&cliente_especial.grau_miopia[0]);//float
scanf("%f",&cliente_especial.grau_miopia[1]);//float
```

Leitura de Variáveis Estruturadas

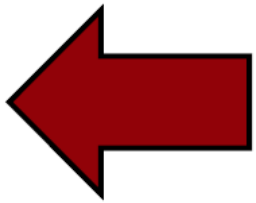
- Cada campo dentro de uma estrutura pode ser acessado como uma variável independente
  - Seu uso não sofre interferência dos demais campos da estrutura
  - **Ex:** ler o campo `paciente_especial.idade` não me obriga a ler o campo `paciente_especial.peso`

Exemplo de um Programa

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

struct dados_pacientes {
        int  idade, e_civil;
        char nome[10];
        double peso, altura;
        float grau_miopia[2];
};

int main() {

    struct dados_pacientes paciente;

    // lembre que string é um vetor, não pode atribuir direto
    strcpy(paciente.nome,"Jose");
    paciente.altura = 1.25;
    paciente.peso = 73;
    paciente.e_civil = 1; // 0:solteiro, 1:casado, 2:outro
    paciente.grau_miopia[0] = 1.75; // olho esquerdo
    paciente.grau_miopia[1] = 0; // olho direito
}
```

Exemplo de um Programa

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

struct dados_pacientes {
        int  idade, e_civil;
        char nome[10];
        double peso, altura;
        float grau_miopia[2];
};

int main() {
    struct dados_pacientes paciente;

    // lembre que string é um vetor, não pode atribuir direto
    strcpy(paciente.nome,"Jose");
    paciente.altura = 1.25;
    paciente.peso = 73;
    paciente.e_civil = 1; // 0:solteiro, 1:casado, 2:outro
    paciente.grau_miopia[0] = 1.75; // olho esquerdo
    paciente.grau_miopia[1] = 0; // olho direito
}
```



- A struct pode ser declarada fora da `main()`
- Isso é o mais comum e será importante quando a struct for usada por outras funções no programa

Dúvida

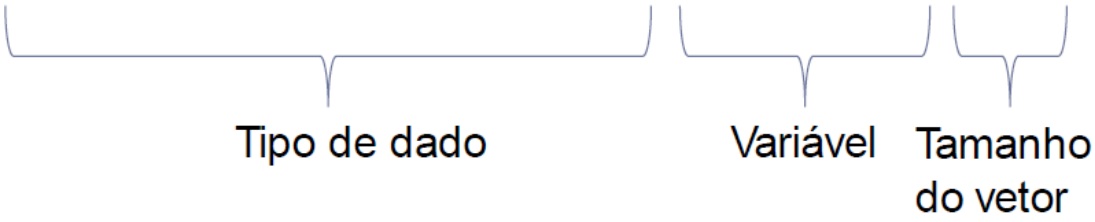
Considerando a `struct dados_pacientes`, como podemos alterar o código para fazer o cadastro de 100 pacientes?

Vetor de Estruturas

Dúvida: considerando a `struct dados_pacientes`, como podemos alterar o código para fazer o cadastro de 100 pacientes?

SOLUÇÃO: criar um **vetor de struct**

- Declaração similar a um array de tipo básico
  - `struct dados_pacientes pacientes[100];`



  - A variável pacientes contém 100 posições, onde cada uma é do tipo `struct dados_pacientes`

Vetor de Estruturas

- **Lembrando:**
  - **struct:** define um "conjunto" de campos que podem ser de tipos diferentes
    - Deve somar o tamanho de todos os campos para obter o tamanho de um elemento do tipo struct
  - **Array:** é uma "lista" de elementos de mesmo tipo
    - Deve multiplicar o tamanho de um elemento do tipo struct pela quantidade de elementos da lista
  - *Essa conta fica como exercício e pode ser facilmente verificada usando o operador sizeof*

Exercício 1

- Utilizando a estrutura abaixo, faça um programa para ler o número e as 3 notas de 10 alunos. Calcule a média para cada aluno e armazene na estrutura.

```
struct aluno {
  int num_aluno;
  float nota1, nota2, nota3;
  float media;
};
```

Exercício 1 – Solução (sem printfs)

```c
struct aluno {
    int num_aluno;
    float nota1, nota2, nota3;
    float media;
};

int main(){

  struct aluno a[10];
  int i;
  for (i=0;i<10;i++){
    scanf("%d",&a[i].num_aluno);
    scanf("%f",&a[i].nota1);
    scanf("%f",&a[i].nota2);
    scanf("%f",&a[i].nota3);
    a[i].media = (a[i].nota1 + a[i].nota2 + a[i].nota3)/3.0;
  }
}
```

Exercício 2

- Modifique o exercício anterior para considerar a estrutura abaixo
  - Note que temos um vetor dentro da estrutura

```
struct aluno {
  int num_aluno;
  float nota[3];
  float media;
};
```

Exercício 2 – Solução (sem printfs)

```
int main(){
  struct aluno a[10];
  int i;
  for (i=0;i<10;i++){
    scanf("%d",&a[i].num_aluno);
    scanf("%f",&a[i].nota[0]);
    scanf("%f",&a[i].nota[1]);
    scanf("%f",&a[i].nota[2]);
    a[i].media = (a[i].nota[0] + a[i].nota[1] + a[i].nota[2])/3.0;
  }
}
```

Atribuição entre Estruturas

- Atribuições entre 2 estruturas só podem ocorrer se os **campos forem IGUAIS**

  - **Exemplo 1:**

```
struct cadastro c1, c2;
c1 = c2; // CORRETO!
```

  - **Exemplo 2:**

```
struct cadastro c1;
struct ficha c2;
c1 = c2; // ERRADO! (TIPOS DIFERENTES)
```

Atribuição entre Estruturas

- A atribuição entre diferentes elementos do array de estruturas é válida
  - Elementos de um mesmo array são sempre IGUAIS
  - **Exemplo:**

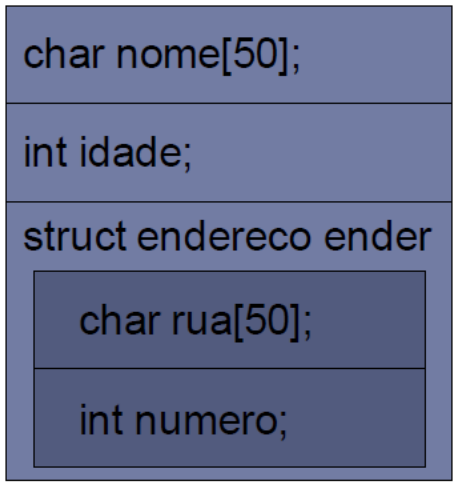
```
struct cadastro c[10];
c[1] = c[2]; // CORRETO!
```

Estrutura de Estruturas

- Considerando que uma estrutura é um tipo de dado, podemos declarar uma estrutura que utilize outra estrutura previamente definida

Exemplo:

```
struct endereco{
  char rua[50]
  int numero;
};
struct cadastro{
  char nome[50];
  int idade;
  struct endereco ender;
};
```

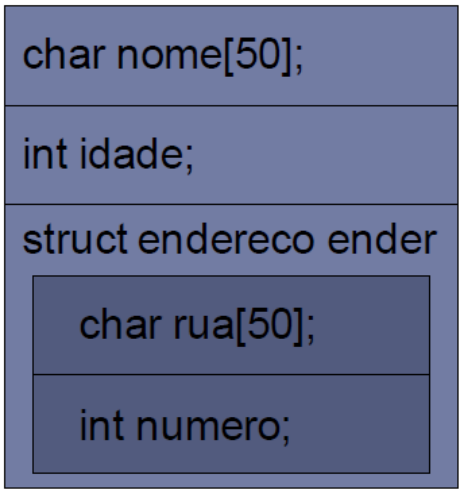


cadastro

Estrutura de Estruturas

- Nesse caso, o acesso aos dados do **endereço** do cadastro é feito utilizando novamente o operador "."

- **Exemplo:**



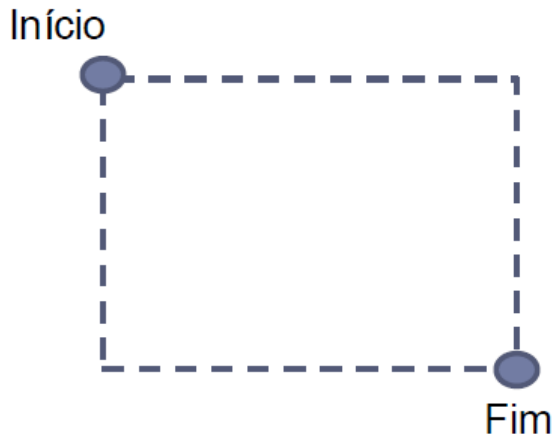
cadastro

```
struct cadastro c;

gets(c.nome);
scanf("%d",&c.idade);
gets(c.ender.rua);
scanf("%d",&c.ender.numero);
```

Estrutura de Estruturas

- **Ex:** Representação de um retângulo

```
struct ponto {
   int x, y;
};

struct retangulo {
   struct ponto inicio, fim;
};

struct retangulo r;

scanf("%d",&r.inicio.x);
scanf("%d",&r.inicio.y);
scanf("%d",&r.fim.x);
scanf("%d",&r.fim.y);
```



Estrutura de Estruturas

- Inicialização de uma estrutura de estruturas:

```
struct ponto {
   int x, y;
};

struct retangulo {
   struct ponto inicio, fim;
};

struct retangulo r = {{10,20},{30,40}};
```

inicio fim

Exercício

- Considerando a `struct retangulo` definida no slide anterior, faça um programa que leia as coordenadas dos pontos que definem um retângulo e retorne a sua área.

- Ao final, teste o seu programa para as coordenadas:

1ª execução: (10,20) e (30,40)

2ª execução: (30,40) e (10,20)

Referências

✓ Básica

- **DAMAS, Luís. "Linguagem C". Grupo Gen-LTC, 2016.**
- **MIZRAHI, Victorine V. "Treinamento em linguagem C", 2a. ed., São Paulo, Pearson, 2008.**

✓ Extra

- BACKES, André. *"Programação Descomplicada Linguagem C"*. Projeto de extensão que disponibiliza vídeo-aulas de C e Estruturas de Dados. Disponível em: `https://www.youtube.com/user/progdescomplicada`. Acessado em: 25/04/2022.

✓ Baseado nos materiais dos seguintes professores:

- Prof. André Backes (UFU)
- Prof. Bruno Travençolo (UFU)
- Prof. Luiz Gustavo de Almeida Martins (UFU)

Prof. Me. Claudiney R. Tinoco
profclaudineytinoco@gmail.com

Faculdade de Computação (FACOM)
Universidade Federal de Uberlândia (UFU)