Aulas Unidade 2

Algoritmos e Programação

Universidade Federal do Rio Grande do Norte – UFRN

Escola Agrícola de Jundiaí – EAJ

Tecnologia em Análise e Desenvolvimento de Sistemas – TADS

Período Letivo 2024.1

Profa. Alessandra Mendes Pacheco (alemendesp@gmail.com)

Ementa do Curso

1. **Estruturas de dados homogêneas**
    - **1.1. Vetores**
    - 1.2. Cadeia de Caracteres – *Strings*
    - 1.3. Matrizes
2. Estruturas de dados heterogêneas
    - 2.1. Registros
3. Vetores de Registros

DEFINIÇÃO

- As estruturas de dados homogêneas correspondem a um **conjunto** de elementos de um **mesmo tipo** e são representadas por um único nome.
- Também são chamadas de ***arrays***.
- Podem ser:
  - **Vetores**: unidimensionais (linha)
  - **Matrizes**: multidimensionais (linhas e colunas)

DECLARAÇÃO

- Os vetores precisam ser declarados **antes** de serem utilizados.
- Na declaração de um vetor devem ser determinados: o **tipo dos dados** que serão armazenados, o **nome** que o identificará e **quantas posições** terá.
- Declaração:

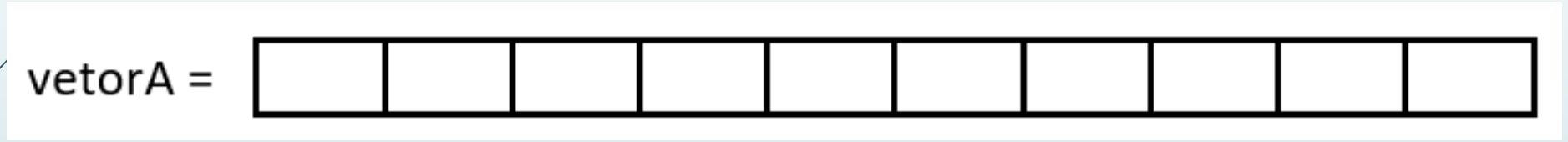
```
tipo nomeVetor[tamanho];
```

```
int vetorA[10];
```

DECLARAÇÃO

- Exemplo:

```
int vetorA[10];
```



O vetorA é **alocado** na memória com **10 espaços** diferentes capazes de armazenar, **cada um**, um valor **inteiro**.

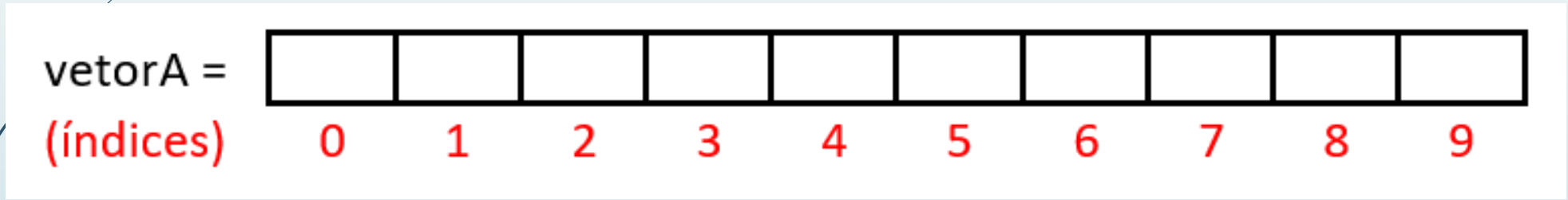
ARMAZENAMENTO DE DADOS

- Um vetor (*armário*) possui vários "**compartimentos**" (*gavetas*) nos quais podem ser armazenados vários dados, porém **todos do mesmo tipo**.
- Cada dado fica armazenado em uma posição diferente do vetor chamada de **índice**.
- Os índices permitem que os elementos armazenados no vetor sejam **individualizados**.

ARMAZENAMENTO DE DADOS

- Exemplo:

```
int vetorA[10];
```



A **primeira posição** do vetor em C++ é o índice **0**

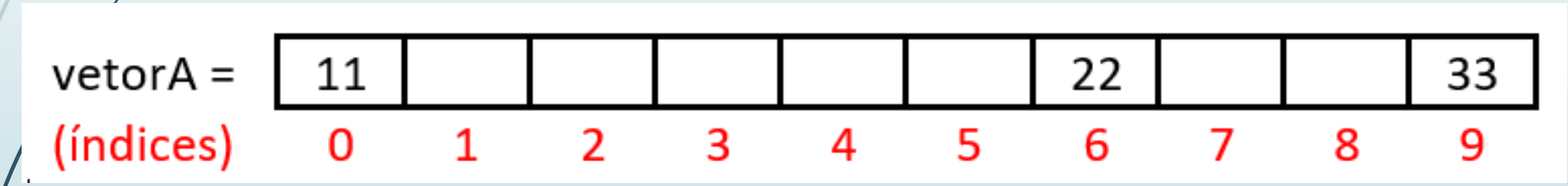
ACESSO AOS ELEMENTOS ARMAZENADOS

- Para acessar os elementos de um vetor são utilizados os **índices**.
- Os índices definem as **posições dos dados** dentro do vetor.
- Os índices (em C++) são **números inteiros que variam de 0** (primeira posição do vetor) a ***n*-1** (última posição do vetor), onde *n* é o total de elementos do vetor.

ACESSO AOS ELEMENTOS ARMAZENADOS

- Exemplos:

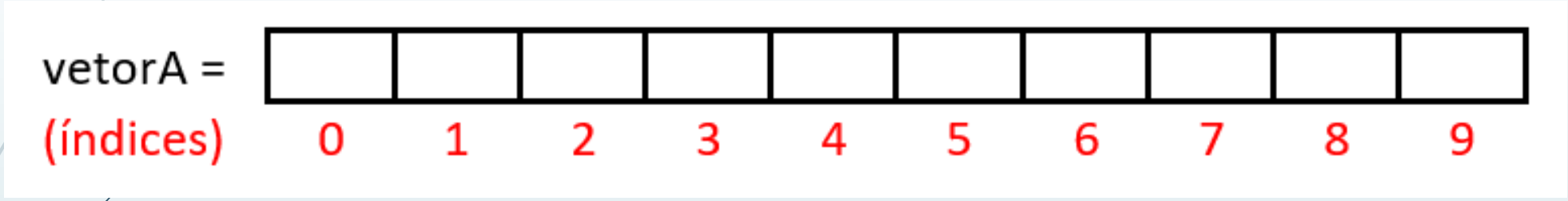
```
vetorA[0] = 11;
vetorA[6] = 22;
vetorA[9] = 33;
```



Pode-se acessar os elementos **individualmente** ou utilizando uma **variável** inteira como índice

ACESSO AOS ELEMENTOS ARMAZENADOS

- Os elementos do vetor podem ser acessados



- Exemplo:

```
int main(){
  int i;
  int vetorA[10];
  for(i=0; i<10; i++){
    vetorA[i] = (i*2);
  }
  cout << "\nElementos de A: ";
  for(i=0; i<10; i++){
    cout << vetorA[i] << " ";
  }
}
```

Qual a saída desse algoritmo?

Ementa do Curso

1. Estruturas de dados homogêneas

- 1.1. Vetores
- **1.2. Cadeia de Caracteres – *Strings***
- 1.3. Matrizes

2. Estruturas de dados heterogêneas

- 2.1. Registros

3. Vetores de Registros

CADEIAS DE CACARTERES

- Faz-se necessário o uso de variáveis para armazenar uma **cadeia de caracteres**.
- Pode-se utilizar ***arrays* do tipo *char***.
- Exemplos:

```
char A[] = {'o','i'};
        // vetor A com tamanho 2 preenchido
char palavra[10];
        // vetor palavra com tamanho 10 não preenchido
cin >> palavra; //digitando "teste", palavra:
```

palavra =	T	E	S	T	E	\0	:	X	?	5
(índices)	0	1	2	3	4	5	6	7	8	9

CADEIAS DE CACARTERES

- Para acesso dos elementos individualmente, deve-se utilizar o seu respectivo índice.
- Exemplo:

```
int main(){
   char texto[10];
   cout << "Digite uma palavra: ";
   cin >> texto;
   cout << "\nTexto digitado: " << texto;
   cout << "\nPrimeira letra: " << texto[0];
   cout << "\nUltima letra: " << texto[9];
}
```

E se a palavra não tiver 10 letras?

- Obs: um *array* de *char* contém um conjunto de caracteres encerrados por aspas duplas e após o último caracter existe um caractere '\0' indicando o fim da sequência.

CADEIAS DE CACARTERES

- Exemplo:

```
int main(){
   char texto[10];
   int i=0;
   cout << "Digite uma palavra: ";
   cin >> texto;
   cout << "\nTexto digitado: " << texto;
   cout << "\nPrimeira letra: " << texto[0];
   while(texto[i]!='\0'){
      i++;
   }
   cout << "\nUltima letra: " << texto[i-1];
}
```

STRINGS

- São objetos que representam cadeias de caracteres.
  - Declaração: `string texto;`
  - Inicialização: `texto = "teste";`
  - Leitura (com espaços): `getline(cin,texto);`
  - Escrita: `cout << texto;`
- Observações:
  - Os elementos podem ser acessados pelo índice.
    - Observar diferença entre ‘ ’ e “ ”.
  - Para comparação utiliza-se o == e para atribuição, utiliza-se o =.

STRINGS

- Em C++, para utilizar *strings* deve-se incluir o arquivo header <*string*>.
- Métodos úteis (considerando *string s*):
  - `s.length()` – retorna o tamanho da string
  - `s.empty()` – testa se a *string* está vazia
  - `s.subst(0, 4)` – retorna uma *string* parcial formada por 4 caracteres da string s a partir da posição 0
  - `s.compare(0,5,"teste")` – retorna 0 se os 5 caracteres a partir da posição 0 forem iguais a "teste"
  - Outras: http://www.cplusplus.com/reference/string/string/

Ementa do Curso

1. Estruturas de dados homogêneas

- 1.1. Vetores
- 1.2. Cadeia de Caracteres – *Strings*
- **1.3. Matrizes**

2. Estruturas de dados heterogêneas
   - 2.1. Registros
3. Vetores de Registros

DECLARAÇÃO

- As matrizes precisam ser declaradas **antes** de serem utilizadas.
- Na declaração de uma matriz devem ser determinados: o **tipo dos dados** que serão armazenados, o **nome** que a identificará e **quantas posições** terá.
- Declaração:

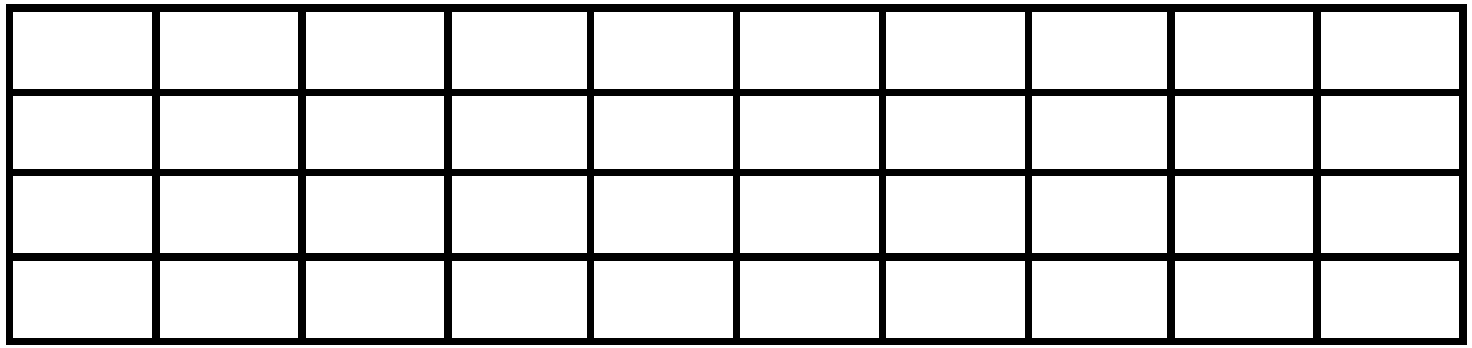
```
tipo nomeMatriz[linhas][colunas];
```

```
int matrizA[4][10];
```

DECLARAÇÃO

- Exemplo: `int matrizA[4][10];`

matrizA =



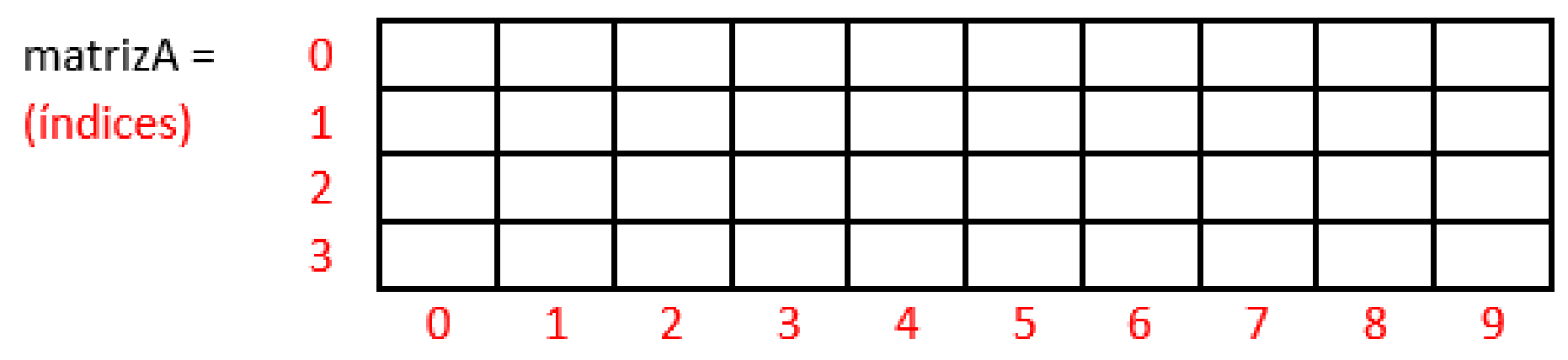
A matrizA é **alocada** na memória com **4x10 espaços** diferentes capazes de armazenar, **cada um**, um valor **inteiro**.

ARMAZENAMENTO DE DADOS

- Uma matriz (*armário*) possui vários "**compartimentos**" (*gavetas*) nos quais podem ser armazenados vários dados, porém **todos do mesmo tipo**.
- Cada dado fica armazenado em uma posição diferente da matriz chamada de **índice**.
- Os índices permitem que os elementos armazenados na matriz sejam **individualizados**.

ARMAZENAMENTO DE DADOS

- Exemplo: `int matrizA[4][10];`



A **primeira posição** da matriz em C++ é a **0,0**

ACESSO AOS ELEMENTOS ARMAZENADOS

- Para acessar os elementos de uma matriz são utilizados os **índices**. Os índices definem as **posições dos dados** dentro da matriz.
  - A matriz bidimensional tem dois índices, um para linhas e outro para colunas.
- Os índices são **números inteiros que variam de 0,0** (primeira posição da matriz) a ***n*-1,*m*-1** (última posição da matriz), onde *n* é o total de linhas e *m* é o total de colunas da matriz.

ACESSO AOS ELEMENTOS ARMAZENADOS

- Exemplo:

```
matrizA[0][2] = 11;
matrizA[1][5] = 22;
matrizA[3][9] = 33;
```

matrizA = (índices)

	0	1	2	3	4	5	6	7	8	9
0			11							
1						22				
2										
3										33

Pode-se acessar os elementos **individualmente** ou utilizando duas **variáveis** inteiras como índices

ACESSO AOS ELEMENTOS ARMAZENADOS

- Os elementos da matriz podem ser acessados utilizando-se laços de repetição
  - Dois laços *for* aninhados podem ser utilizados neste caso.
- Exemplo:

```
int main(){
  int i,j;
  int matrizA[4][10];
  for(i=0; i<4; i++){
    for(j=0; j<10; j++){
      matrizA[i][j] = (i+j);
      cout << matrizA[i][j] << " ";
    }
    cout << "\n";
  }
}
```

Qual a saída desse algoritmo?

Ementa do Curso

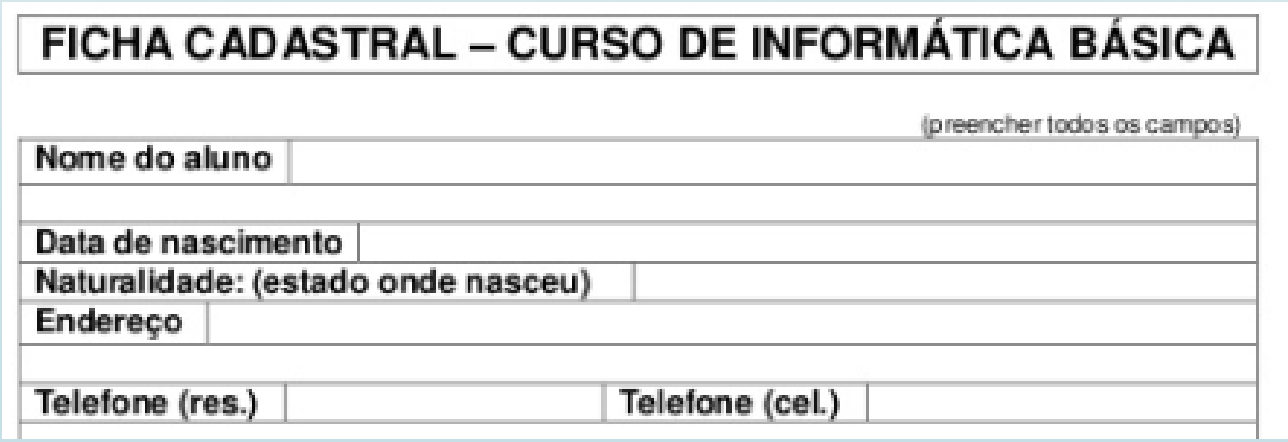
1. Estruturas de dados homogêneas
    - 1.1. Vetores
    - 1.2. Cadeia de Caracteres – *Strings*
    - 1.3. Matrizes
2. **Estruturas de dados heterogêneas**
    - **2.1. Registros**
3. Vetores de Registros

DEFINIÇÃO

- As estruturas de dados heterogêneas correspondem a um **conjunto** de elementos de **tipos diferentes** que são agrupados em uma única estrutura: **o registro**.
- Os registros possibilitam que vários dados de tipos diferentes (os campos) estejam **na mesma estrutura**.
  - **Os índices aqui são os nomes dos campos**.
- Para se declarar uma variável do tipo registro, deve-se antes **criar um tipo** com a estrutura do registro desejado.

DEFINIÇÃO

- Os registros são ***análogos*** às fichas de cadastro!
  - Possuem **campos** que identificam os dados que serão armazenados;
  - Cada campo funciona como uma **variável primitiva**;
  - todos os campos estão **agrupados no registro**.



FICHA CADASTRAL – CURSO DE INFORMÁTICA BÁSICA

(preencher todos os campos)

Nome do aluno			
Data de nascimento			
Naturalidade: (estado onde nasceu)			
Endereço			
Telefone (res.)		Telefone (cel.)	

DECLARAÇÃO

- Os **tipos de registros** precisam ser criados **antes** que as variáveis sejam declaradas.
- Na declaração de um registro devem ser determinados os **campos** e os seus respectivos **tipos**.
- Exemplo:

```
Tipo <nome_da_variavel> = registro
    <nome_campo_1>: <tipo_campo_1>
    <nome_campo_2>: <tipo_campo_2>
      …
    <nome_campo_n>: <tipo_campo_n>
fimregistro
```

DECLARAÇÃO

- Exemplo típico: **registro Aluno**
  - Matrícula (inteiro)
  - Nome (literal);
  - Data de nascimento (literal);
  - Média Geral (real);

etc...

Registro Aluno:

CURSO DE INFORMÁTICA É AGORA OU NUNCA	
Matrícula	
Nome	
Data de Nascimento	
Média Geral	

DECLARAÇÃO

- Exemplo do Registro Aluno em C++:

```
struct Aluno{
    int matricula;
    string nome;
    string dataNascimento;
    float mediaGeral;
};
```

O registro é a **forma**.

ARMAZENAMENTO E ACESSO

- Após a especificação do registro, as variáveis devem ser declaradas.

```
Aluno a;
```

- Para acessar os campos do registro, deve-se utilizar o “.”

```
a.Matricula = 1;
cin >> a.matricula;
getline(cin, a.nome);
```

A variável é o **bolo**.

ARMAZENAMENTO E ACESSO

- Exemplo: `Aluno a;`

Variável a	
matricula	(*int*)
nome	(*string*)
dataNascimento	(*string*)
mediaGeral	(*float*)

A variável **a** do "tipo" Aluno é **alocada** na memória com **quatro espaços diferentes** capazes de armazenar um dado do tipo ***int*** no campo matricula, um dado ***string*** no campo nome, um dado ***string*** no campo dataNadcimento e um dado ***float*** no campo mediaGeral.

ARMAZENAMENTO E ACESSO

- Exemplo:

```
a.Matricula = 1;
a.nome = "Ana";
a.dataNascimento = "22/04/1990";
a.mediaGeral = 7.5;
```

a	
matricula	1
nome	Ana
dataNascimento	22/04/1990
mediaGeral	7.5

Os **dados** podem ser armazenados e acessados através dos **campos**.

Ementa do Curso

1. Estruturas de dados homogêneas
    1.1. Vetores
    1.2. Cadeia de Caracteres – *Strings*
    1.3. Matrizes
2. Estruturas de dados heterogêneas
    2.1. Registros
3. **Vetores de Registros**

DEFINIÇÃO

- Os **vetores de registro** são utilizados quando existe a necessidade de armazenar um **conjunto de registros**.
- Exemplo: Cadastro de Alunos.

Cadastro de Alunos:

aluno1
matricula
nome
dataNascimento
mediaGeral

aluno2
matricula
nome
dataNascimento
mediaGeral

aluno3
matricula
nome
dataNascimento
mediaGeral

aluno4	
matricula	999
nome	Vander Lee Sucesso
dataNascimento	29/03/1970
mediaGeral	8.9

DECLARAÇÃO

- Os **tipos de registros** precisam ser criados **antes** que as variáveis sejam declaradas, inclusive os vetores.
- Na declaração de um vetor de registro devem ser determinados o tipo (registro) e tamanho do vetor.
- Os elementos devem ser acessados através do índice do vetor e do campo do registro.
- Exemplo:

```
Aluno vetorAlunos[200];
vetorAlunos[0].nome = "Ana";
vetorAlunos[0].mediaGeral = 7.8;
```

Vamos à prática...

Dúvidas?