UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Vetores de Strings + Structs

Aula 09

Marcos Silvano Almeida
Departamento de Computação
UTFPR Campo Mourão

Visualizando vetores de duas dimensões

Vetor de strings = Matriz de caracteres

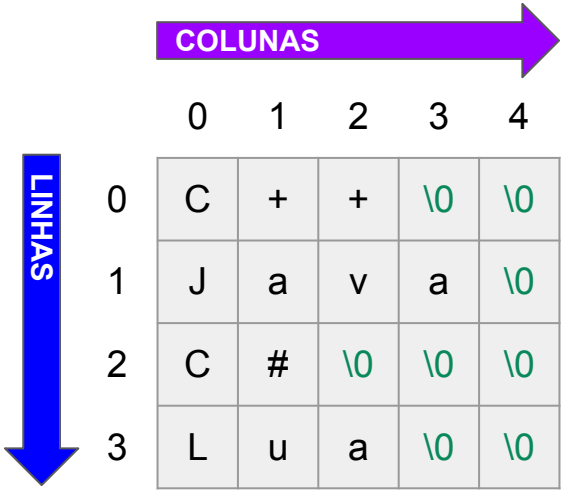
```
int n = 4;
int tamString = 5;

char vetor_de_strings[n][tamString];
```

```
{"C++", "Java", "C#", "Lua"}
```

cada string é um vetor de **chars**

```
{
   {'C', '+', '+' ,'\0','\0'},
   {'J', 'a', 'v' , 'a','\0'},
   {'C', '#', '\0','\0','\0'},
   {'L', 'u', 'a' ,'\0','\0'}
}
```

Visualizando como uma matriz (forma geométrica 2D)



Vetor de strings = Matriz de caracteres

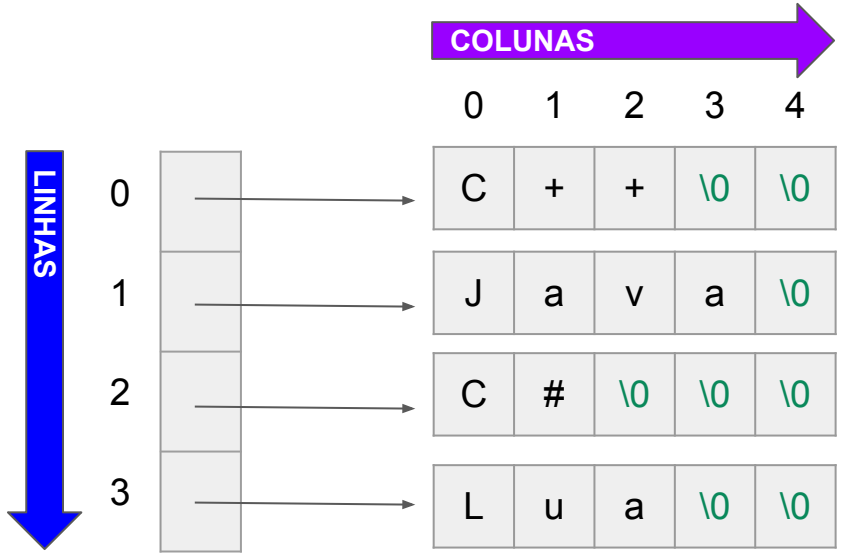
```
int n = 4;
int tamString = 5;

char vetor_de_strings[n][tamString];
```

```
{"C++", "Java", "C#", "Lua"}
```

cada string é um vetor de **chars**

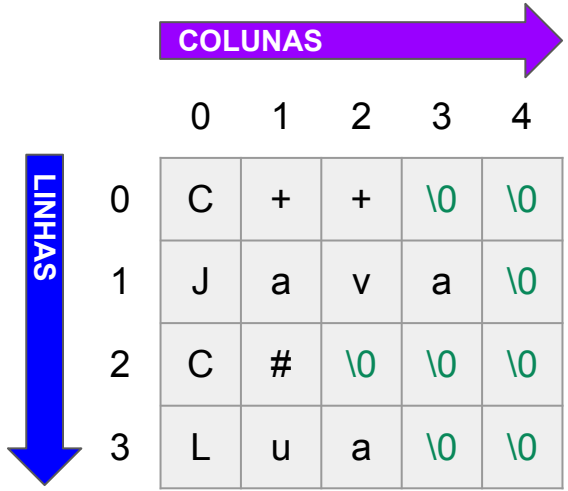
```
{
   {'C', '+', '+' ,'\0','\0'},
   {'J', 'a', 'v' , 'a','\0'},
   {'C', '#', '\0','\0','\0'},
   {'L', 'u', 'a' ,'\0','\0'}
}
```

Visualizando como vetor de vetores



Imprimindo vetor de strings

```
void printStringVector(int n, int len, char v[n][len]) {
 // percorre vetor de strings
 for (int i = 0; i < n; i++) {
   // percorre cada string na posição i
   for (int j = 0; v[i][j] != '\0'; j++) {
     printf("%c", v[i][j]);
   }
   printf("\n");
 }
}
void main() {
 // vetor de 4 strings de 5 chars (4 char + '\0')
 char stringVector[4][5] = {"C++", "Java", "C#", "Lua"};
 printStringVector(4, 5, stringVector);
}
```

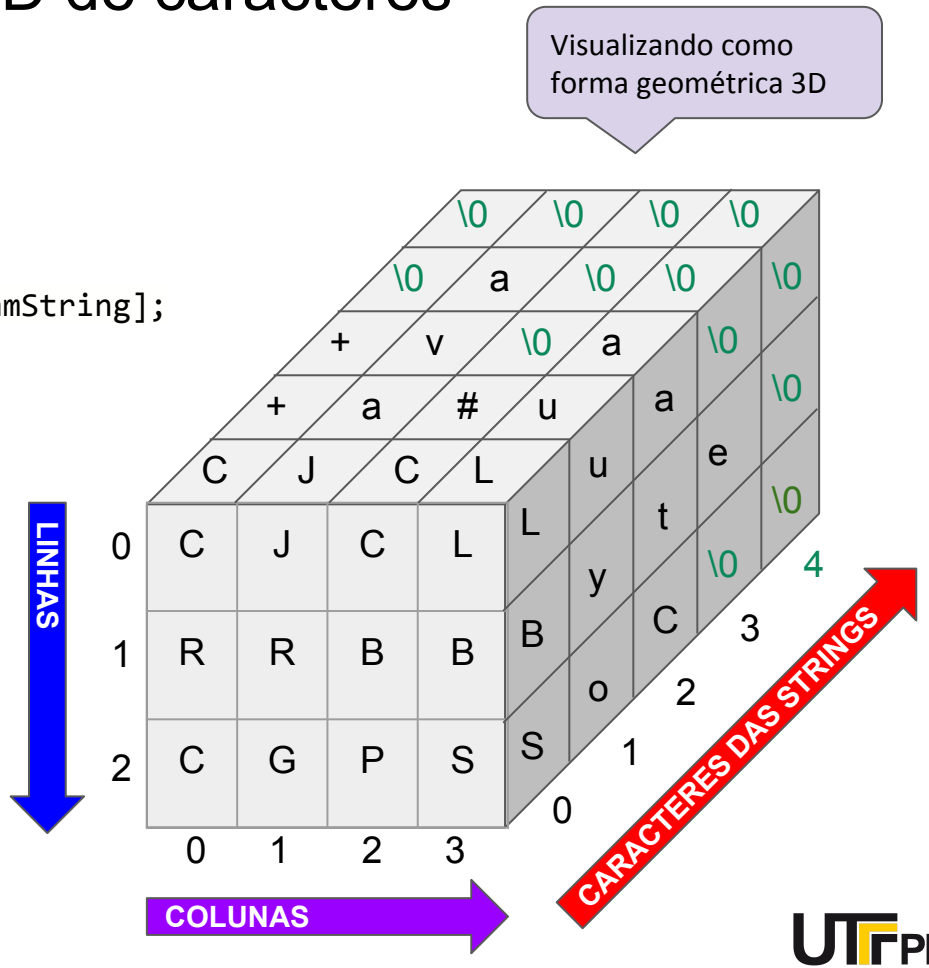


Visualizando vetores de três dimensões

Matriz de strings = Matriz 3D de caracteres

```
int linhas = 3;
int colunas= 4;
int tamString = 5;

char matriz_de_strings[linhas][colunas][tamString];
```

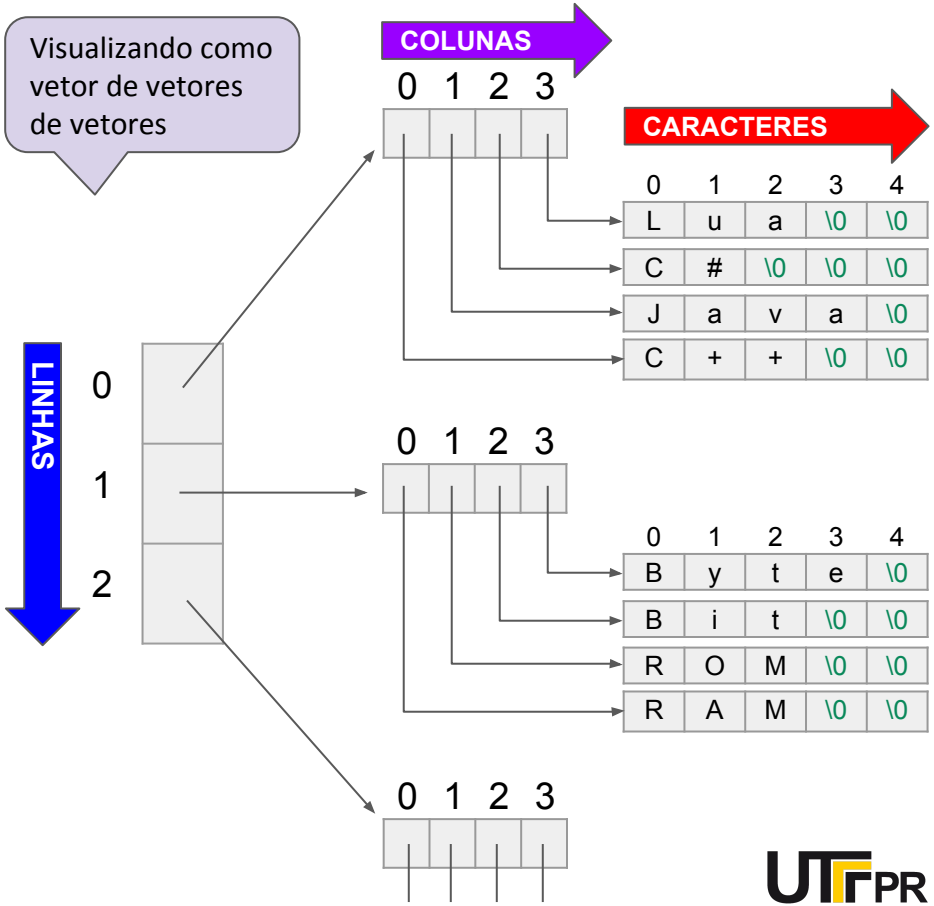
```
{
  {"C++", "Java", "C#", "Lua"},
  {"RAM", "ROM", "Bit", "Byte"},
  {"CPU", "GPU", "PPU", "SoC"}
}
```



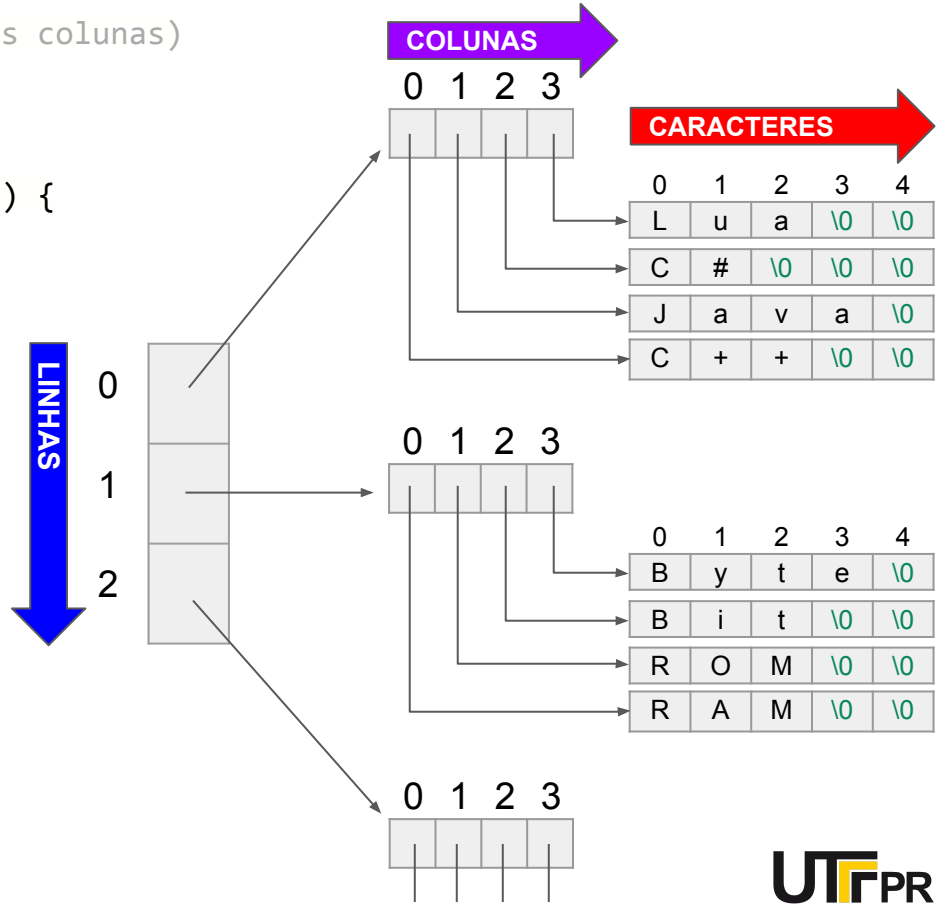
Matriz de strings = Matriz 3D de caracteres

```
int linhas = 3;
int colunas= 4;
int tamString = 5;

char matriz_de_strings
          [linhas][colunas][tamString];
```

```
{
  {"C++", "Java", "C#", "Lua"},
  {"RAM", "ROM", "Bit", "Byte"},
  {"CPU", "GPU", "PPU", "SoC"}
}
```



```c
void printStringMatrix(int rows, int cols, int len, char v[rows][cols][len]) {
 // percorre as linhas da matrix
 for (int i = 0; i < rows; i++) {
   // percorre os elementos de cada linha (as colunas)
   for (int j = 0; j < cols; j++) {
     // percorre cada string na posição i,j
     for (int k = 0; v[i][j][k] != '\0'; k++) {
       printf("%c", v[i][j][k]);
     }
     printf(" ");
   }
   printf("\n");
 } }
void main() {
 // matriz de 3x4 de strings de 5 chars
 char stringMatrix[3][4][5] =  {
   {"C++", "Java", "C#", "Lua"},
   {"RAM", "ROM", "Bit", "Byte"},
   {"CPU", "GPU", "PPU", "SoC"}
 }; }
```



Tipo Estruturado em C: Structs

Struct

- Usado para se definir um tipo composto por campos (um **registro**)
- Formato geral:

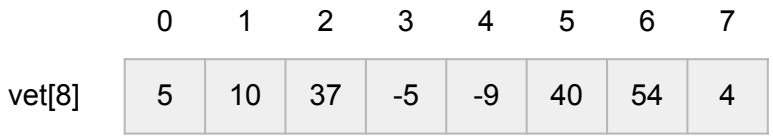
```
struct NOME_DA_ESTRUTURA {
    tipo campo1;
    tipo campo2;
    ...
};
```

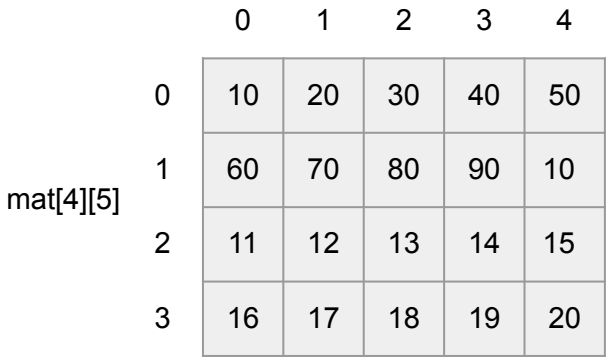
- Exemplo:

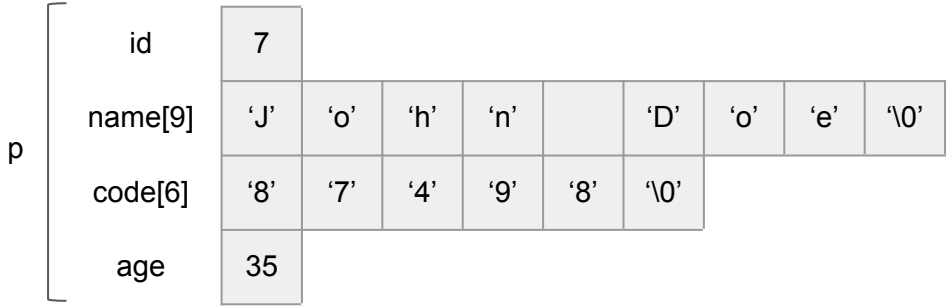
```
struct Person {
    int id;
    char name[31];
    char cpf[12];
    int age;
};
```

Variável vs Struct vs Vetor

```c
int a = 5;
int vet[8] = {5, 10, 37, -5, -9, 40, 54, 4};

int mat[4][5] = {
 { 10, 20, 30, 40, 50},
 { 60, 70, 80, 90, 10},
 { 11, 12, 13, 14, 15},
 { 16, 17, 18, 19, 20}
};

struct Person {
 int id;
 char name[9];
 char code[6];
 int age;
};
struct Person p = {7, "John Doe", "87498", 35};
```







Struct

```c
struct Person {
 int id;
 char name[31];
 char cpf[12];
 int age;
};
void main() {
 // declarando e inicializando variável do tipo struct Person
 struct Person p = {5, "Joanna Dark", "34587090872", 23};
 // acesso a cada campo do registro
 printf("person: %d, %s, %s, %d\n", p.id, p.name, p.cpf, p.age);
  // modificando os campos
 p.id = 564;
 stringCopy(p.name, 31, "Joanna Daft Punk");
 // confirmando modificações
 printf("person: %d, %s, %s, %d\n", p.id, p.name, p.cpf, p.age);
}
```

```c
void stringCopy(char dest[], int n, char src[]) {
 int i;
 for (i = 0; src[i] != 0 && i < n-1; i++) {
   dest[i] = src[i];
 }
 dest[i] = '\0';
}
```

stringCopy copia n caracteres de src (origem) para dest (destino).

Struct

- Passando variáveis do tipo registro (estruturado) para funções

```
void printPerson(struct Person p) {
     printf("\nPessoa\n");
     printf("  id....: %d\n", p.id);
     printf("  nome..: %s\n", p.name);
     printf("  cpf...: %s\n", p.cpf);
     printf("  idade.: %d\n", p.age);
}
void main() {
     struct Person p1 = {1, "John Doe", "08767854315", 25};
     struct Person p2 = {2, "Jane Mill","09357323450", 31};
     struct Person p3 = p1; // struct permite atribuição/cópia
     printPerson(p1);
     printPerson(p2);
     printPerson(p3);
}
```

O parâmetro recebe uma cópia da variável passada (ao contrário de parâmetros vetores)

Recebe uma struct como parâmetro e imprime seus campos

Ao contrário de vetores, variáveis struct podem ser copiadas

Vetor de structs

```
struct Person {
      int id;
      char name[16];
      int age;
};

struct Person v[4] = {
      {1, "John Doe", 25},
      {2, "Juanes Millis",32},
      {3, "Troy Maker",29},
      {4, "Annette Facino",45}
};

for (int i = 0; i < 4; i++) {
  struct Person p = v[i];
  printf("Pessoa {%3d, %-15s, %2d}\n", p.id, p.name, p.age);
}
```

Declara e inicializa vetor de structs Person

Imprime os campos de cada Person no vetor

```
SAÍDA:
Pessoa {  1, John Doe       , 25}
Pessoa {  2, Juanes Millis  , 32}
Pessoa {  3, Troy Maker     , 29}
Pessoa {  4, Annette Facino , 45}
```

Vetor de structs: alterando após declaração

- Considerando as funções anteriores

```
int main() {
 // declara vetor de structs Person
 struct Person v[5];

 struct Person p;
 p.id = 7;
 p.age= 26;
 stringCopy(p.name, "John Doe", 15);
 v[0] = p;

 v[1].id = 9;
 v[1].age= 45;
 stringCopy(v[1].name, "Joanne", 15);
 return 0;
}
```

Copia um struct para posição 0 do vetor

Como não existe atribuição de vetores, precisamos copiar uma string para a outra, char a char.

Define os campos para o struct na posição 1 do vetor

Struct VS Vetor

- Duas formas de agregar variáveis em uma só estrutura
- Resumo:

Vetor	Struct
Usado para armazenar uma **coleção de N dados** do mesmo tipo	Usado para se definir um **tipo composto** por um número fixo de campos
✔️Permite inicialização	✔️Permite inicialização
❌Não permite atribuição/cópia	✔️Permite atribuição/cópia
Parâmetro de função: Quando uma variável **vetor** é passada à um parâmetro de função, este passa a ser uma referência a essa (variável externa). Alterações no parâmetro dentro do código da função **refletem** no vetor externo. Isso se chama **passagem de parâmetro por referência**.	**Parâmetro de função:** Quando uma variável **struct** é passada à um parâmetro de função, este recebe uma cópia dessa. Alterações no parâmetro dentro do código da função **não refletem** no struct externo. Isso se chama **passagem de parâmetro por cópia**.