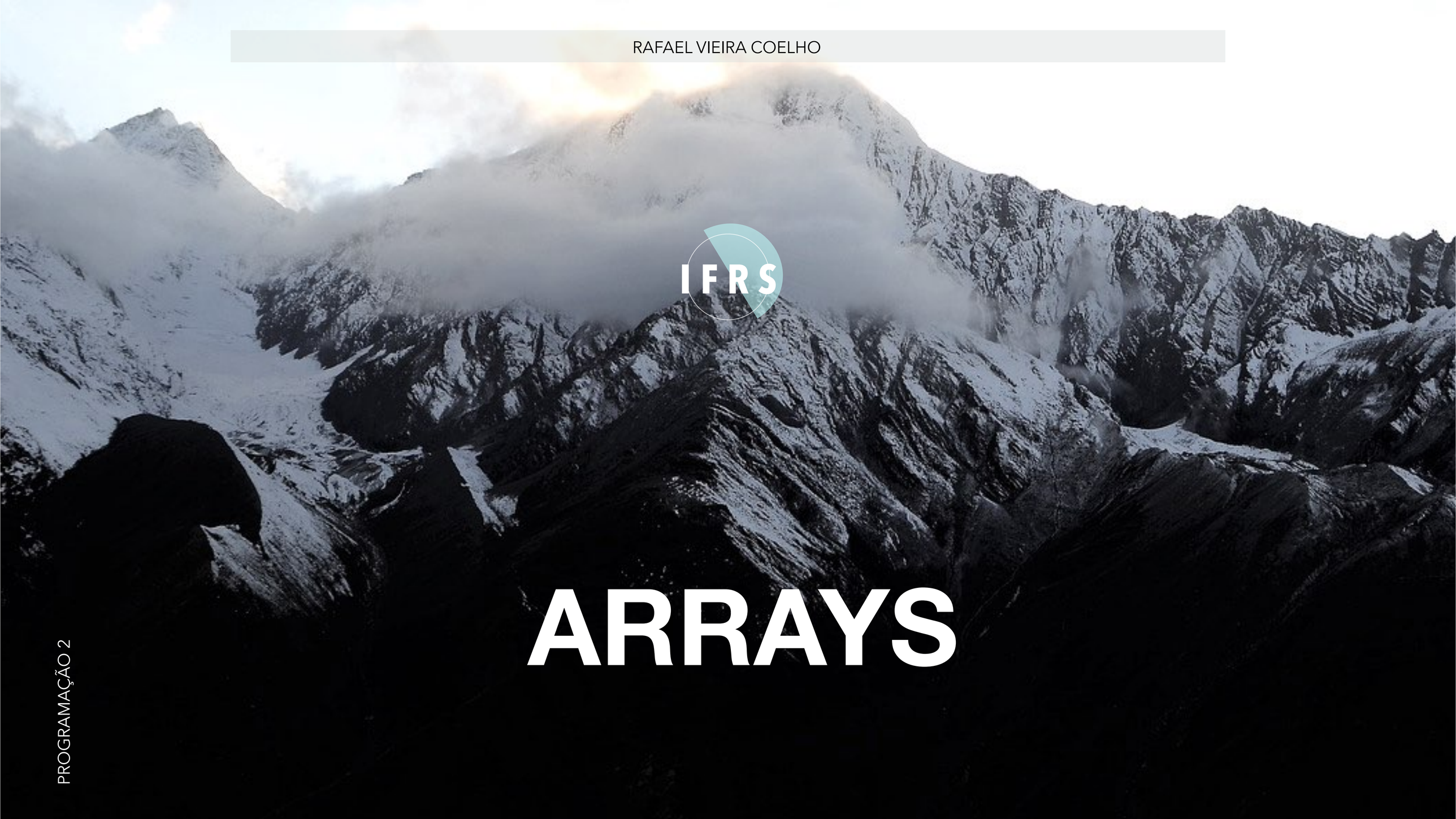
RAFAEL VIEIRA COELHO



IFRS

ARRAYS

PROGRAMAÇÃO 2

CAPÍTULO 10:ARRAYS

https://books.goalkicker.com/CBook/

VETORES (ARRAYS)

São tipos de dados compostos que agrupam contiguamente (sequencial) em memória elementos de mesmo tipo.

Declaração com tamanho definido:

```
int array[10];
```

Declaração e Inicialização:

```
int array[10] = {1, 2, 3};
int array[5] = {[2] = 5, [1] = 2, [4] = 9}; /* array is {0, 2, 5, 0, 9} */
int array[] = {1, 2, 3}; /* an array of 3 int's */
int array[] = {[3] = 8, [0] = 9}; /* size is 4 */
```

VETORES (ARRAYS)

Podemos popular o vetor com valores acessando sua posição.

```c
int numbers[10];

/* populate the array */
numbers[0] = 10;
numbers[1] = 20;
numbers[2] = 30;
numbers[3] = 40;
numbers[4] = 50;
numbers[5] = 60;
numbers[6] = 70;

/* print the 7th number from the array, which has an index of 6 */
printf("The 7th number in the array is %d", numbers[6]);
```

PERCORRENDO UM VETOR

```c
#define ARRLEN 10000
int array[ARRLEN];

size_t i;
for (i = 0; i < ARRLEN; ++i)
{

        array[i] = 0;

}
```

TAMANHO DE UM VETOR

Mesmo sabendo o tamanho do vetor em sua declaração, é possível calcular o seu tamanho em tempo de execução.

```c
int array[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

/* size of `array` in bytes */
size_t size = sizeof(array);

/* number of elements in `array` */
size_t length = sizeof(array) / sizeof(array[0]);
```

Obs: o tipo size_t representa um inteiro positivo (sem sinal) pois não podemos ter tamanho negativo.

TAMANHO DE UM VETOR

Quando passamos um vetor por parâmetro, estamos apenas passando um ponteiro que aponta para o endereço de memória do primeiro elemento.

Sendo assim, é necessário passar também como parâmetro o seu tamanho (número de elementos).

```c
/* array will decay to a pointer, so the length must be passed separately */
int last = get_last(array, length);
```

A função **get_last()** pode ser implementada assim:

```c
int get_last(int input[], size_t length) {
    return input[length - 1];
}
```

TAMANHO DE UM VETOR

Não podemos obter o seu tamanho dentro da função.

```c
int BAD_get_last(int input[]) {
    /* INCORRECTLY COMPUTES THE LENGTH OF THE ARRAY INTO WHICH input POINTS: */
    size_t length = sizeof(input) / sizeof(input[0]));

    return input[length - 1];  /* Oops -- not the droid we are looking for */
}
```

Um aviso (warning) é emitido quando se tenta fazer isto:

```
warning: sizeof on array function parameter will return size of 'int *' instead of 'int []' [-
Wsizeof-array-argument]
        int length = sizeof(input) / sizeof(input[0]);
                             ^
note: declared here
int BAD_get_last(int input[])
```

VETOR MULTIDIMENSIONAL

Declarando um vetor a de duas dimensões (2 linhas x 3 colunas):

```
int val = a[2][3];
```

Inicializando um vetor com 3 linhas x 4 colunas:

```
int a[3][4] = {
   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
   {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
   {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

PERCORRENDO UM VETOR BIDIMENSIONAL

```c
#define ARRLEN 10000
int array[ARRLEN][ARRLEN];

size_t i, j;
for (i = 0; i < ARRLEN; ++i)
{
    for(j = 0; j < ARRLEN; ++j)
    {
        array[i][j] = 0;
    }
}
```

VETORES MULTIDIMENSIONAIS

vetor_bidimencional.c

```c
#include <stdio.h>

int main () {
    /* an array with 5 rows and 2 columns*/
    int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
    int i, j;

    /* output each array element's value */
    for ( i = 0; i < 5; i++ ) {
        for ( j = 0; j < 2; j++ ) {
            printf("a[%d][%d] = %d\n", i,j, a[i][j] );
        }
    }
    return 0;
}
```

```
a[0][0] = 0
a[0][1] = 0
a[1][0] = 1
a[1][1] = 2
a[2][0] = 2
a[2][1] = 4
a[3][0] = 3
a[3][1] = 6
a[4][0] = 4
a[4][1] = 8
```

FUNÇÃO ASSERT

usando_assert.c

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

int main() {
  int a, b;

  printf("Input an integer:\n");
  scanf("%d", &a);
  printf("Input an integer to divide:\n");
  scanf("%d", &b);

  assert(b != 0);

  printf("%d/%d = %.2f\n", a, b, a/(float)b);

  return 0;
}
```

```
mac:ExemplosC coelho$ gcc usando_assert.c -o exe
mac:ExemplosC coelho$ ./exe
Input an integer:
10
Input an integer to divide:
3
10/3 = 3.33
mac:ExemplosC coelho$ ./exe
Input an integer:
10
Input an integer to divide:
0
Assertion failed: (b != 0), function main, file usando_assert.c, line 12.
Abort trap: 6
```

Assert é uma função usada para verificar condições específicas em tempo de execução e é útil para depuração de programas.

INICIALIZANDO VETORES

Podemos inicializar com valor zero percorrendo cada posição.

```c
#include <stdlib.h> /* for EXIT_SUCCESS */

#define ARRLEN (10)

int main(void)
{
  int array[ARRLEN]; /* Allocated but not initialised, as not defined static or global. */

  size_t i;
  for(i = 0; i < ARRLEN; ++i)
  {
    array[i] = 0;
  }

  return EXIT_SUCCESS;
}
```

INICIALIZANDO VETORES

Ou usando a função **memset()**

```
memset(array, 0, sizeof array); /* Use size of the array itself. */
```

```
memset(array, 0, ARRLEN * sizeof *array); /* Use size of type the pointer is pointing to. */
```

```
memset(array, 0, ARRLEN * sizeof (int)); /* Use size explicitly provided type (int here). */
```

ACESSANDO ELEMENTOS DO VETOR

A linguagem C não verifica limites. Isto deve ser feito pelo programador.

```c
int val;
int array[10];

/* Setting the value of the fifth element to 5: */
array[4] = 5;

/* The above is equal to: */
*(array + 4) = 5;

/* Reading the value of the fifth element: */
val = array[4];
```

OBRIGADO!

RAFAELVC2@GMAIL.COM

https://books.goalkicker.com