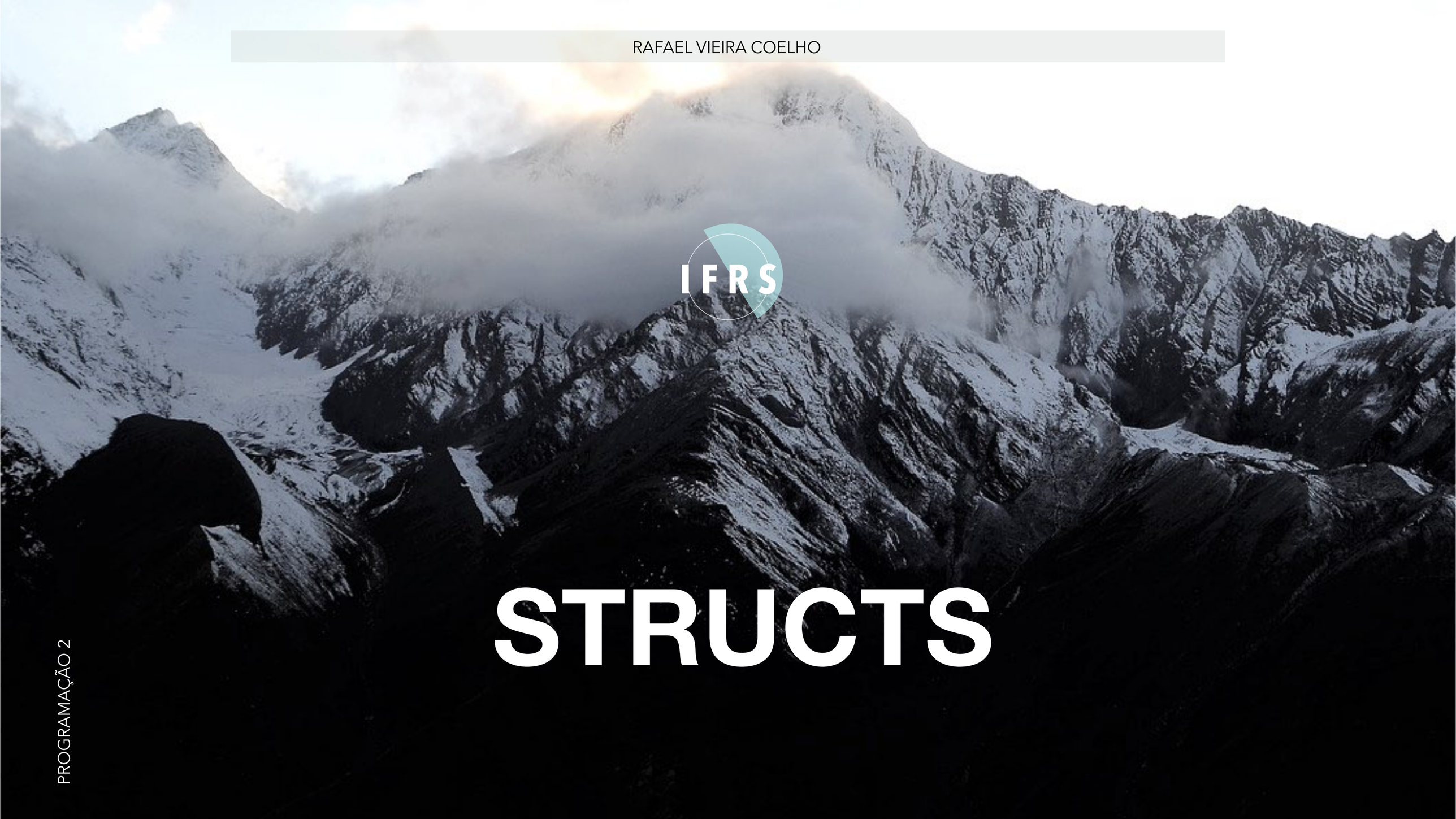
RAFAEL VIEIRA COELHO



STRUCTS

PROGRAMAÇÃO 2

CAPÍTULO 12: ENUMERAÇÕES

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

Trata-se de um tipo de dados definido pelo usuário e consiste em uma série possíveis de valores constantes que este tipo pode ter.

Example 1

```c
enum color{ RED, GREEN, BLUE };

void printColor(enum color chosenColor)
{
    const char *color_name = "Invalid color";
    switch (chosenColor)
    {
       case RED:
         color_name = "RED";
         break;

       case GREEN:
        color_name = "GREEN";
        break;

       case BLUE:
        color_name = "BLUE";
        break;
    }
    printf("%s\n", color_name);
}
```

With a main function defined as follows (for example):

```c
int main(){
    enum color chosenColor;
    printf("Enter a number between 0 and 2");
    scanf("%d", (int*)&chosenColor);
    printColor(chosenColor);
    return 0;
}
```

ENUMERAÇÃO

ENUMERAÇÃO

(This example uses designated initializers which are standardized since C99.)

```c
enum week{ MON, TUE, WED, THU, FRI, SAT, SUN };

static const char* const dow[] = {
  [MON] = "Mon", [TUE] = "Tue", [WED] = "Wed",
  [THU] = "Thu", [FRI] = "Fri", [SAT] = "Sat", [SUN] = "Sun" };

void printDayOfWeek(enum week day)
{
   printf("%s\n", dow[day]);
}
```

DANDO UM NOME PARA A ENUMERAÇÃO

```
typedef enum
{
    RED,
    GREEN,
    BLUE
} color;

color chosenColor = RED;
```

CAPÍTULO 13: STRUCTS

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

REGISTROS (STRUCTS): DECLARAÇÃO

Trata-se de uma forma de agrupar um conjunto de variáveis de diversos tipos em apenas uma unidade de memória.

E esta unidade pode ser referenciada como uma variável através de um nome.

```c
struct ex1
{
    size_t foo;
    int flex[];
};

struct ex2_header
{
    int foo;
    char bar;
};

struct ex2
{
    struct ex2_header hdr;
    int flex[];
};

/* Merged ex2_header and ex2 structures. */
struct ex3
{
    int foo;
    char bar;
    int flex[];
};
```

Exemplo do tipo struct point que representa um ponto (x,y) no plano cartesiano.

```c
struct point {
    int x;
    int y;
};
```

```c
/* draws a point at 10, 5 */
struct point p;
p.x = 10;
p.y = 5;
draw(p);
```

STRUCTS

DANDO UM NOME PARA A STRUCT

```
typedef struct {
    int x;
    int y;
} point;
```

```
point p;
```

OPERADORES DE ACESSO

Os operadores **.** e **->** são usados para acessar os membros de um ***struct***

```c
struct MyStruct
{
    int x;
    int y;
};

struct MyStruct myObject;
myObject.x = 42;
myObject.y = 123;

printf(".x = %i, .y = %i\n", myObject.x, myObject.y); /* Outputs ".x = 42, .y = 123". */
```

OPERADORES DE ACESSO

A relação **x->y** é um atalho para **(*x).y**

```c
struct MyStruct
{
    int x;
    int y;
};

struct MyStruct myObject;
struct MyStruct *p = &myObject;

p->x = 42;
p->y = 123;

printf(".x = %i, .y = %i\n", p->x, p->y); /* Outputs ".x = 42, .y = 123". */
printf(".x = %i, .y = %i\n", myObject.x, myObject.y); /* Also outputs ".x = 42, .y = 123". */
```

PONTEIROS X STRUCTS

Para acessarmos os membros de um struct deve-se usar o ponto(.)

```
struct dma {
   int dia;
   int mes;
   int ano;
};
struct dma x;  // um registro x do tipo dma
struct dma y;  // um registro y do tipo dma
```

```
x.dia = 31;
x.mes = 12;
x.ano = 2018;
```

Já quando temos um ponteiro que aponta para um struct, acessa-se com a seta (->).

```
data *p;        // p é um ponteiro para registros dma
data  x;
p = &x;         // agora p aponta para x
(*p).dia = 31; // mesmo efeito que x.dia = 31
 p->dia = 31;    // mesmo efeito que (*p).dia = 31
```

STRUCTS COMO PARÂMETRO

```c
struct coordinates
{
    int x;
    int y;
    int z;
};

// Passing and returning a small struct by value, very fast
struct coordinates move(struct coordinates position, struct coordinates movement)
{
    position.x += movement.x;
    position.y += movement.y;
    position.z += movement.z;
    return position;
}
```

OBRIGADO!

RAFAELVC2@GMAIL.COM

https://books.goalkicker.com