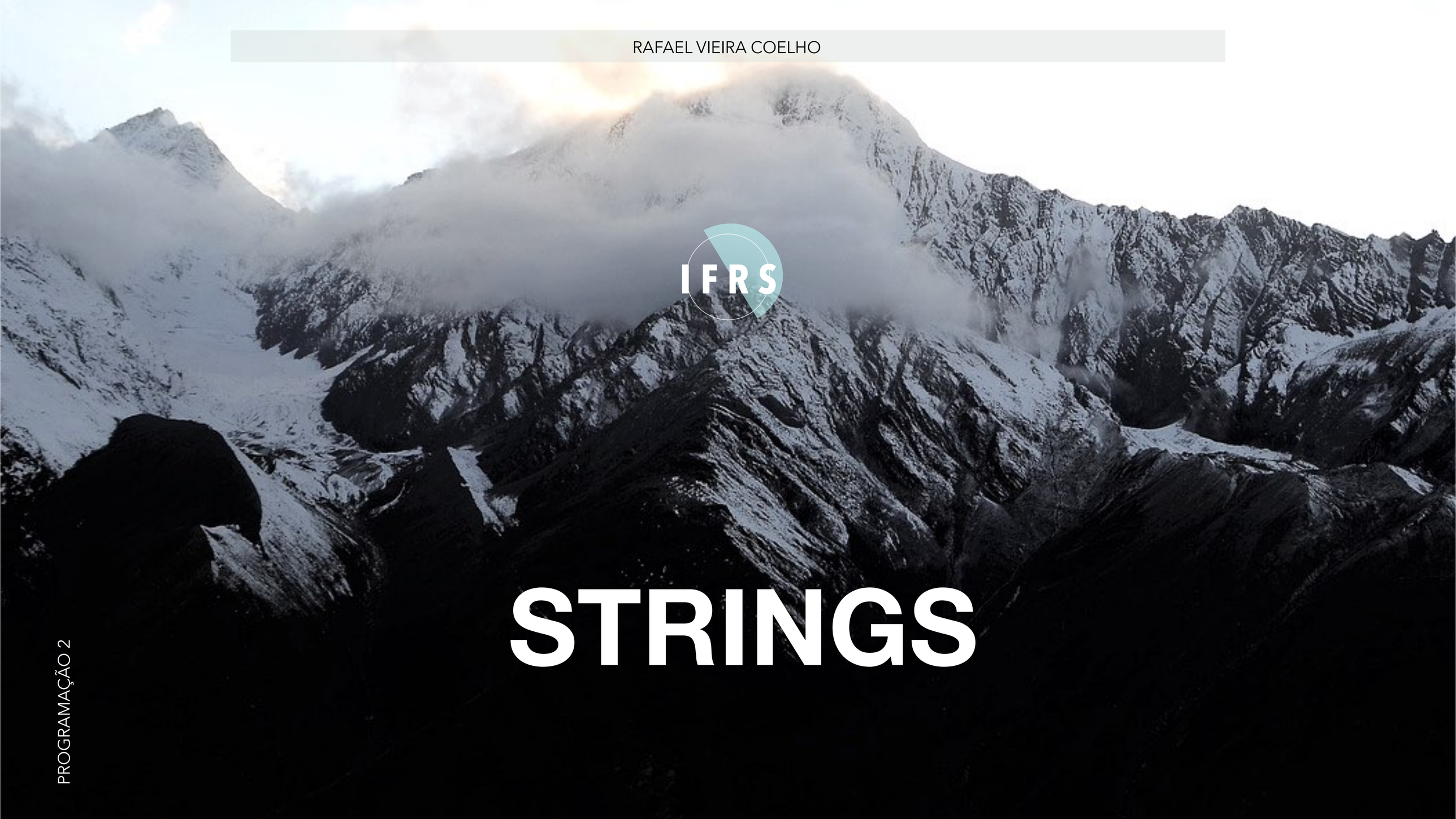
RAFAEL VIEIRA COELHO



IFRS

STRINGS

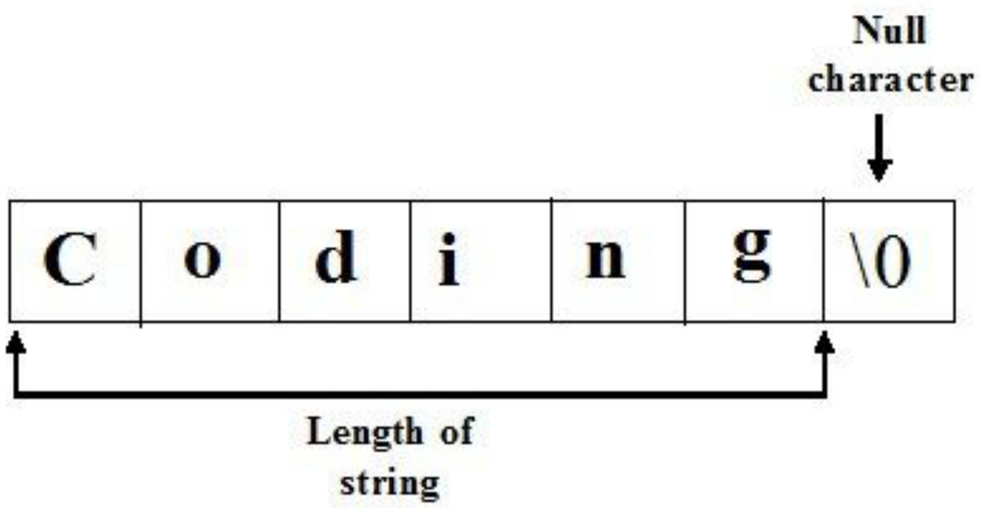
PROGRAMAÇÃO 2

CAPÍTULO 6: STRINGS

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

CADEIA DE CARACTERES (STRINGS)

Em Linguagem C não existe o tipo String. No entanto, podemos usar um vetor de caracteres que termine em **'\0'**



'abc" will have four characters 'a', 'b', 'c' and '\0'.

DEFININDO STRINGS

Existem algumas formas de definir um vetor de caracteres:

```
char * name = "John Smith";
```

```
char name[] = "John Smith";
```

```
char name[] = "John Smith";
/* is the same as */
char name[11] = "John Smith";
```

STRINGS CONSTANTES

Podemos definir uma string constante (não pode ser modificada):

```
char const * string = "hello world";
```

```
char const string_arr[] = "hello world";
```

Ou uma string modificável:

```
char modifiable_string[] = "hello world";
```

```
char modifiable_string[] = {'h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '\0'};
```

LITERAIS

Podemos usar uma string literal (const char) para algumas coisas que não podemos fazer com um vetor de char, como por exemplo ser retornado por uma função.

```
const char *get_hello() {
    return "Hello, World!";  /* safe */
}
```

Qualquer tentativa de alterar uma string através de um ponteiro para char, mesmo que não seja constante gera erro de comportamento desconhecido.

```
char *foo = "hello";
foo[0] = 'y';  /* Undefined behavior - BAD! */
```

O ideal neste caso é definir logo de início em sua declaração que se trata de uma literal constante.

```
const char *foo = "hello";
/* GOOD: can't modify the string pointed to by foo */
```

LITERAIS

No entanto, podemos usar o ponteiro para que ele aponte para outra string.

```
char *foo = "hello";
foo = "World!"; /* OK - we're just changing what foo points to */
```

Caso queiramos alterar uma string, devemos declará-la como um vetor de caracteres.

```
char foo[] = "hello";
foo[0] = 'y';  /* OK! */
```

FORMATANDO O PRINTF

Existem algumas formas de definir um vetor de caracteres:

```c
char * name = "John Smith";
int age = 27;

/* prints out 'John Smith is 27 years old.' */
printf("%s is %d years old.\n", name, age);
```

USANDO O SPRINTF

Podemos escrever em uma string formatada conforme o segundo parâmetro.

```
int sprintf ( char * str, const char * format, ... );
```

No exemplo, usamos a função **sprintf()** para escrever um float em uma string.

```
#include <stdio.h>
int main ()
{
  char buffer [50];
  double PI = 3.1415926;
  sprintf (buffer, "PI = %.7f", PI);
  printf ("%s\n",buffer);
  return 0;
}
```

LENDO UMA STRING

Podemos utilizar a função **scanf()** com a string "%[^\n" para funcionar com strings compostas.

```c
#include <stdio.h>

int main() {
    char str2[50];

    printf("Digite um texto: ");
    scanf("%[^\n]", str2); fflush(stdin);
    printf("Texto 2: %s \n", str2);
    return 0;
}
```

Obs: não esqueça de dar o **fflush()** para limpar a cache de leitura.

USANDO O SSCANF PARA LEITURA

Podemos ler o conteúdo formatado de uma string:

```
int sscanf ( const char * s, const char * format, ...);
```

O exemplo abaixo lê os dados e os mostra em outro formato:

```
#include <stdio.h>
int main ()
{
  char sentence []="date : 06-06-2012";
  char str [50];
  int year;
  int month;
  int day;
  sscanf (sentence,"%s : %2d-%2d-%4d", str, &day, &month, &year);
  printf ("%s -> %02d-%02d-%4d\n",str, day, month, year);
  return 0;
}
```

% c especifica um char

% d especifica um int

% u especifica um unsigned int

% f,% e,% g especificam um float

% lf, % le, % lg especificam um double

% sespecifica uma cadeia de caracteres

TAMANHO DE UMA STRING

A função strlen() de string.h retorna a quantidade de caracteres da String.

```
char * name = "Nikhil";
printf("%d\n",strlen(name));
```

COMPARANDO DUAS STRINGS

A função **strcmp()** compara duas strings e retorna a quantidade de caracteres distintos. Retorna zero se são iguais

```
char * name = "John";

if (strncmp(name, "John", 4) == 0) {
    printf("Hello, John!\n");
} else {
    printf("You are not John. Go away.\n");
}
```

A função **strncmp()** permite delimitar o tamanho de caracteres que serão testados (4 no exemplo).

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

void compare(char const *lhs, char const *rhs)
{
    int result = strcmp(lhs, rhs); // compute comparison once
    if (result < 0) {
        printf("%s comes before %s\n", lhs, rhs);
    } else if (result == 0) {
        printf("%s equals %s\n", lhs, rhs);
    } else { // last case: result > 0
        printf("%s comes after %s\n", lhs, rhs);
    }
}

int main(void)
{
    compare("BBB", "BBB");
    compare("BBB", "CCCCC");
    compare("BBB", "AAAAAA");
    return 0;
}
```

Outputs:

```
BBB equals BBB
BBB comes before CCCCC
BBB comes after AAAAAA
```

COMPARAÇÃO DE STRINGS

CONCATENANDO DUAS STRINGS

A função **strncat()** une duas strings e ela recebe como terceiro parâmetro o número máximo de caracteres que devem ser concatenados.

```
char dest[20]="Hello";
char src[20]="World";
strncat(dest,src,3);
printf("%s\n",dest);
strncat(dest,src,20);
printf("%s\n",dest);
```

CONCATENANDO STRINGS

Para não termos um *overflow* caso a string destino seja menor.

```c
char dst[24] = "Clownfish: ";
char src[] = "Marvin and Nemo";
size_t len = strlen(dst);

strncat(dst, src, sizeof(dst) - len - 1);
printf("%zu: [%s]\n", strlen(dst), dst);
```

The output is:

```
23: [Clownfish: Marvin and N]
```

COPIANDO STRINGS

Atribuir ponteiros não copia strings.

```c
#include <stdio.h>

int main(void) {
    int a = 10, b;
    char c[] = "abc", *d;

    b = a; /* Integer is copied */
    a = 20; /* Modifying a leaves b unchanged - b is a 'deep copy' of a */
    printf("%d %d\n", a, b); /* "20 10" will be printed */

    d = c;
    /* Only copies the address of the string -
    there is still only one string stored in memory */

    c[1] = 'x';
    /* Modifies the original string - d[1] = 'x' will do exactly the same thing */

    printf("%s %s\n", c, d); /* "axc axc" will be printed */

    return 0;
}
```

COPIANDO STRINGS

Não podemos atribuir vetores.

```c
#include <stdio.h>

int main(void) {
    char a[] = "abc";
    char b[8];

    b = a; /* compile error */
    printf("%s\n", b);

    return 0;
}
```

COPIANDO STRINGS

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

int main(void) {
    char a[] = "abc";
    char b[8];

    strcpy(b, a); /* think "b special equals a" */
    printf("%s\n", b); /* "abc" will be printed */

    return 0;
}
```

Version ≥ C99

PERCORRENDO STRINGS

Se soubermos o tamanho da string, podemos usar um laço para percorrer os caracteres através de seu número de caracteres (no exemplo, 11).

```c
char * string = "hello world"; /* This 11 chars long, excluding the 0-terminator. */
size_t i = 0;
for (; i < 11; i++) {
    printf("%c\n", string[i]);    /* Print each character of the string. */
}
```

Caso não soubermos o tamanho da string, podemos usar um laço para percorrer os caracteres através da função strlen().

```c
size_t length = strlen(string);
size_t i = 0;
for (; i < length; i++) {
    printf("%c\n", string[i]);    /* Print each character of the string. */
}
```

PERCORRENDO STRINGS

Outra forma de percorrer uma string sem saber o seu tamanho é testando o caractere final padrão de strings '\0'.

```c
size_t i = 0;
while (string[i] != '\0') {        /* Stop looping when we reach the null-character. */
    printf("%c\n", string[i]);     /* Print each character of the string. */
    i++;
```

QUEBRA DE STRINGS EM TOKENS

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

int main(void)
{
    int toknum = 0;
    char src[] = "Hello,, world!";
    const char delimiters[] = ", !";
    char *token = strtok(src, delimiters);
    while (token != NULL)
    {
        printf("%d: [%s]\n", ++toknum, token);
        token = strtok(NULL, delimiters);
    }
    /* source is now "Hello\0, world\0\0" */
}
```

Output:

```
1: [Hello]
2: [world]
```

A função **strtok()** quebra uma string em strings menores a partir de um delimitador.

VETOR DE STRINGS

Um vetor de strings pode significar duas coisas:

(1) um vetor cujos elementos são char *

(2) Ou um vetor cujos elementos são vetores de char

```
char * string_array[] = {
    "foo",
    "bar",
    "baz"
};
```

```
char modifiable_string_array_literals[][4] = {
    "foo",
    "bar",
    "baz"
};
```

É equivalente à:

```
char modifiable_string_array[][4] = {
    {'f', 'o', 'o', '\0'},
    {'b', 'a', 'r', '\0'},
    {'b', 'a', 'z', '\0'}
};
```

OBRIGADO!

RAFAELVC2@GMAIL.COM

https://books.goalkicker.com