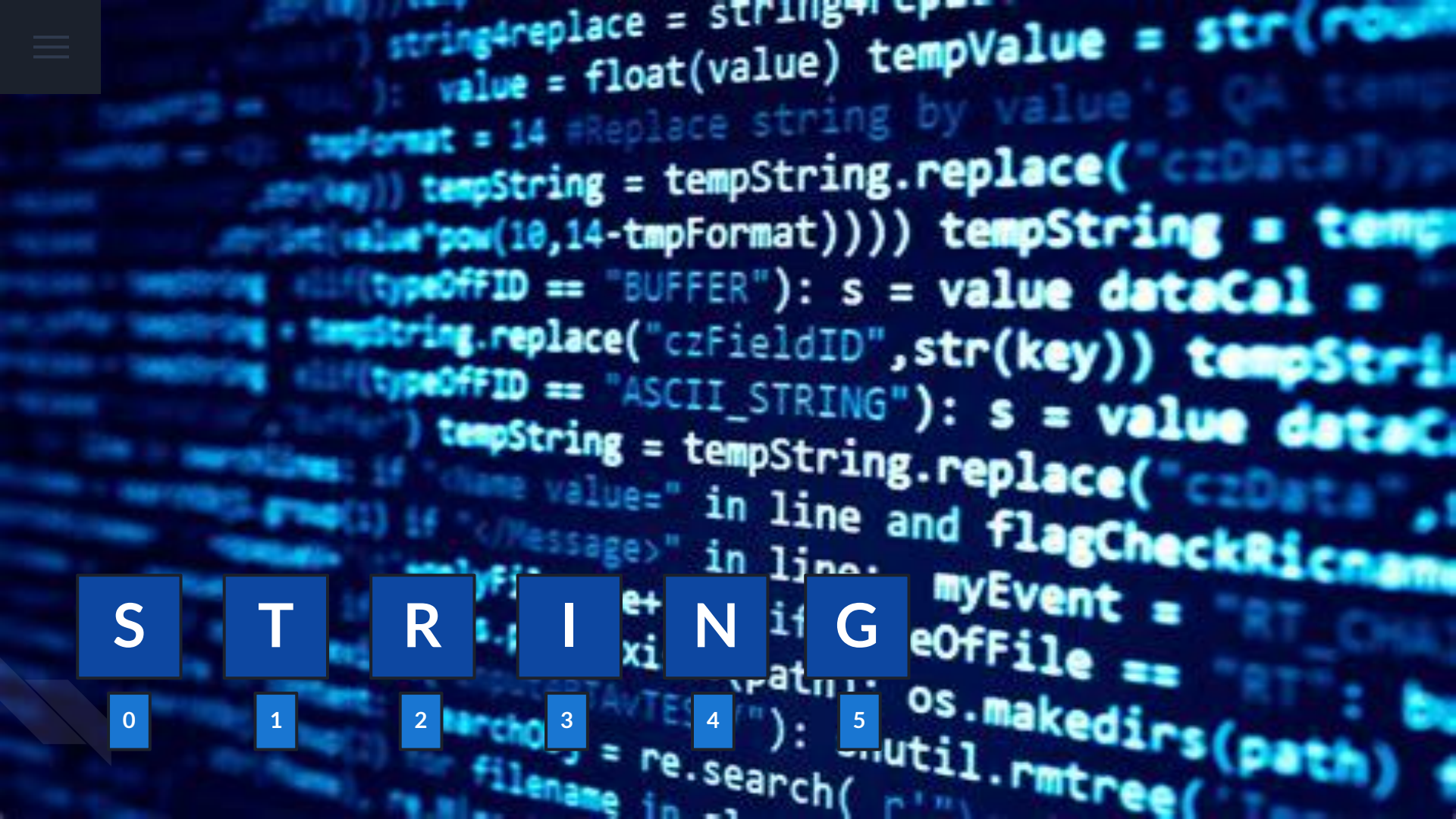

S T R I N G
0 1 2 3 4 5

Conceitos

- ➔ Array de caracteres
- ➔ Armazenadas linearmente na memória
- ➔ Tamanho Fixo (Era pré ponteiros)
- ➔ Invasão de Memória

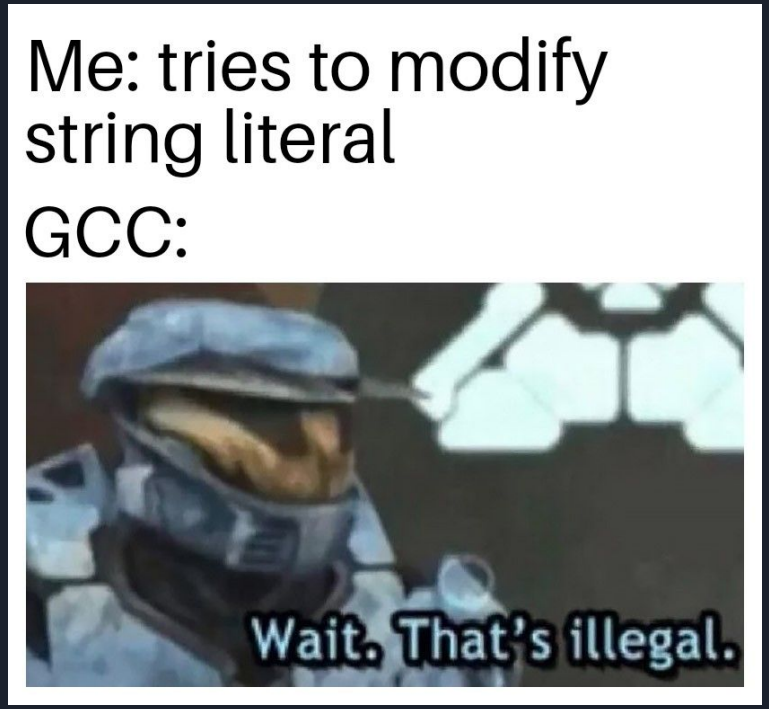


Tabela ASCII

!	33	0041	0x21	a	97	0141	0x61	í	161	0241	0xa1	ß	225	0341	0xe1
"	34	0042	0x22	b	98	0142	0x62	ó	162	0242	0xa2	Ô	226	0342	0xe2
#	35	0043	0x23	c	99	0143	0x63	ú	163	0243	0xa3	Ò	227	0343	0xe3
$	36	0044	0x24	d	100	0144	0x64	ñ	164	0244	0xa4	Õ	228	0344	0xe4
%	37	0045	0x25	e	101	0145	0x65	Ñ	165	0245	0xa5	Õ	229	0345	0xe5
&	38	0046	0x26	f	102	0146	0x66	ª	166	0246	0xa6	µ	230	0346	0xe6
'	39	0047	0x27	g	103	0147	0x67	º	167	0247	0xa7	Þ	231	0347	0xe7
(	40	0050	0x28	h	104	0150	0x68	¿	168	0250	0xa8	Þ	232	0350	0xe8
)	41	0051	0x29	i	105	0151	0x69	®	169	0251	0xa9	Ú	233	0351	0xe9
*	42	0052	0x2a	j	106	0152	0x6a	¬	170	0252	0xaa	Û	234	0352	0xea
+	43	0053	0x2b	k	107	0153	0x6b	½	171	0253	0xab	Ù	235	0353	0xeb
,	44	0054	0x2c	l	108	0154	0x6c	¼	172	0254	0xac	ý	236	0354	0xec
-	45	0055	0x2d	m	109	0155	0x6d	¡	173	0255	0xad	Ý	237	0355	0xed
.	46	0056	0x2e	n	110	0156	0x6e	«	174	0256	0xae	¯	238	0356	0xee
/	47	0057	0x2f	o	111	0157	0x6f	»	175	0257	0xaf	´	239	0357	0xef
0	48	0060	0x30	p	112	0160	0x70	_	176	0260	0xb0		240	0360	0xf0
1	49	0061	0x31	q	113	0161	0x71	_	177	0261	0xb1	±	241	0361	0xf1

Uma parte da tabela ASCII.
ASCII - American Standard Code for Information Interchange.

Armazenamento da string na memória...

char nome[7] = “cleytu”

endereço 1	endereço 2	endereço 3	endereço 4	endereço 5	endereço 6
index [0]	index [1]	index [2]	index [3]	index [4]	index [5]
0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

NOTE: O nome possui só 6 caracteres, mas tem o pobre esquecido (mas você não vai esquecer… NÃO É?) do caractere nulo DEVE ser considerado, então, sempre por + 1 no tamanho do array!

Armazenamento de Memória

Cada caractere ocupa um byte de memória.

```c
int main(){

    char palavra[] = "abc";
    int tam = sizeof(palavra);
    printf("%d\n", tam);

    return 0;
}
```

A saída desse código será 4 pois o tamanho também inclui o caractere nulo.

Iterando a String

```c
int main(){

    int i;
    char palavra[5] = "casa";

    for(i = 0;i < 4;i++){
        printf("%c", palavra[i]);
    }

    return 0;
}
```

Iteramos de [0, tamanho da string-1]
No exemplo, a string “casa” tem tamanho 4.

OBS: cuidado com a invasão de memória

Declaração

Antes de tudo, uma declaração:

```
//Tamanho do vetor é definido automáticamente (Precisa ser inicializado!)
char que[] = "Bababoe";

//Tamanho predefinido
char mosa[13];
//Posso usar depois!

//Vetor de Strings!!!
char latao[4][10];
//4 strings de tamanho 10
```

Atentar para invasão de memória!

Input e Output

Para variáveis conhecidas:

```
scanf( “%d, %f,  %c”, &Inteiros, &Reais, &Letras);
```

Para strings:

```
scanf( “  %s”, String);
```

```
scanf( “ %x[^\n]”, String);
```

Obs: Sempre dar espaço antes!

Obs2: Não usar &!

Input e Output

`gets("string");`

➥Not Safe For Work

Então usamos:

`fgets("string", n, stdin);`

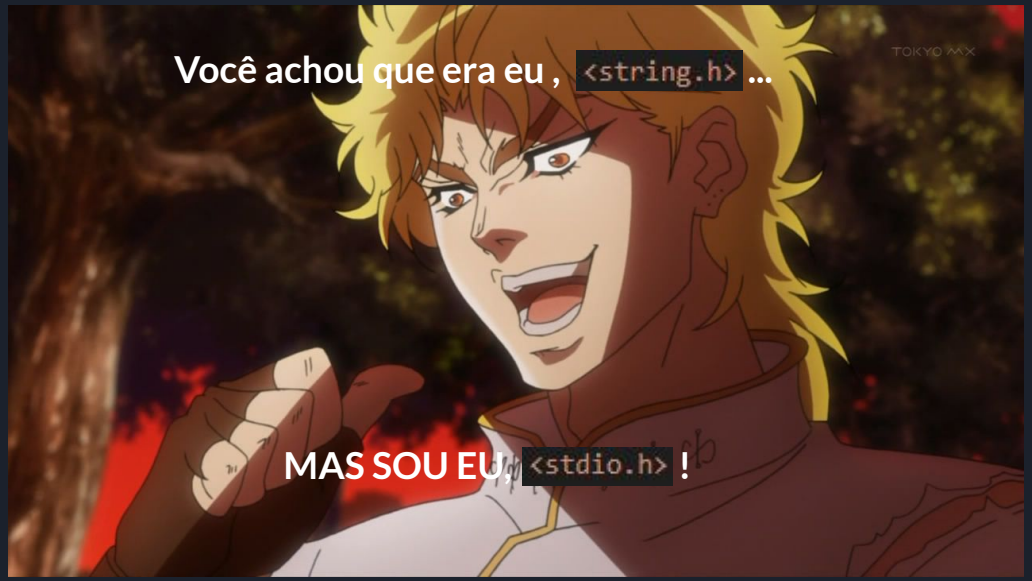
E para printar?

`printf("%s", S);`

Funções

Para facilitar a tua vida, a biblioteca “string.h” tem umas funções amigonas do povo para ajudar a lidar com string

Essas funções usam o **caracter nulo** como referência para funcionarem certinho...



strcpy

copiar uma string em outra (atentar ao tamanho do array)

lembra que não pode fazer reassign de strings diretamente?

```c
char string_origem[] = "cleytinho";
char string_de_destino[15];

strcpy(string_de_destino, string_origem);
printf("copiei %s em %s\n" string_de_destino, string_origem);

// Output
// copiei cleytinho em cleytinho
```

Sem a função

```c
for (int i = 0; i < sizeof(string_origem); ++i)
    string_de_destino[i] = string_origem[i];

printf("copiei %s em %s\n", string_de_destino, string_origem);
```

strcmp

```c
char string_1[] = "cleytinho";
char string_2[] = "cleytinho é legal";

printf("%d\n", strcmp(string_1, string_2));

// output menor que 0
```

compara duas strings e retorna um inteiro que representa a primeira diferença:

retorno 0 → as strings são iguais

retorno < 0 → o conteúdo da primeira é menor que a segunda

retorno > 0 → o conteúdo primeira é maior que a segunda

obs: se os tamanhos forem iguais., a comparação é via código ASCII

strlen

retorna a quantia de caracteres da string, sem contar o caracter nulo!

```c
char frase_cleytinho[] = "cleytinho quer saber o tamanho dele";
printf("%ld\n", strlen(frase_cleytinho));
```

Sem a função

```c
int currentChar = 0;
char frase_cleytinho[] = "cleytinho quer saber o tamanho dele";

while (frase_cleytinho[currentChar] != '\0')
{
    ++currentChar;
}

printf("%d", currentChar);
```

strcat

concatena a segunda string na primeira (se atentar ao tamanho dos arrays)

```
char frase_cleytinho[20] = "cleytinho";
char frase_pra_concatenar[] ="é legal";

strcat(frase_cleytinho, " ");
strcat(frase_cleytinho, frase_pra_concatenar);

printf("%s", frase_cleytinho);
```

sprintf

formar strings de maneira mais dinâmica, com um mecanismo semelhante ao printf

cria a string e armazena em “frase”

```c
int idade_yoda = 900;
char nome_do_yoda[] = "yoda", frase[40];

sprintf(frase, "%s tem (tinha pq faleceu) %d anos", nome_do_yoda, idade_yoda);
printf("%s", frase);

// Output
// yoda tem (tinha pq faleceu) 900 anos
```

OBS: NÃO PRINTA A STRING NO OUTPUT!!