# Aula 10: Strings

Prof. Sérgio Montazzolli Silva
smsilva@uel.br

# Introdução

- Strings são vetores de caracteres terminados por '\0' ("barra zero" é caractere especial, assim como '\n', '\t', etc)
- Usadas por um programa para armazenar texto
- Operações concatenação, contagem de caracteres, substituição, comparação, e muitas outras, são realizadas por meio de funções disponíveis na biblioteca **string.h**
- Nesta aula veremos algumas funções de manipulação de strings

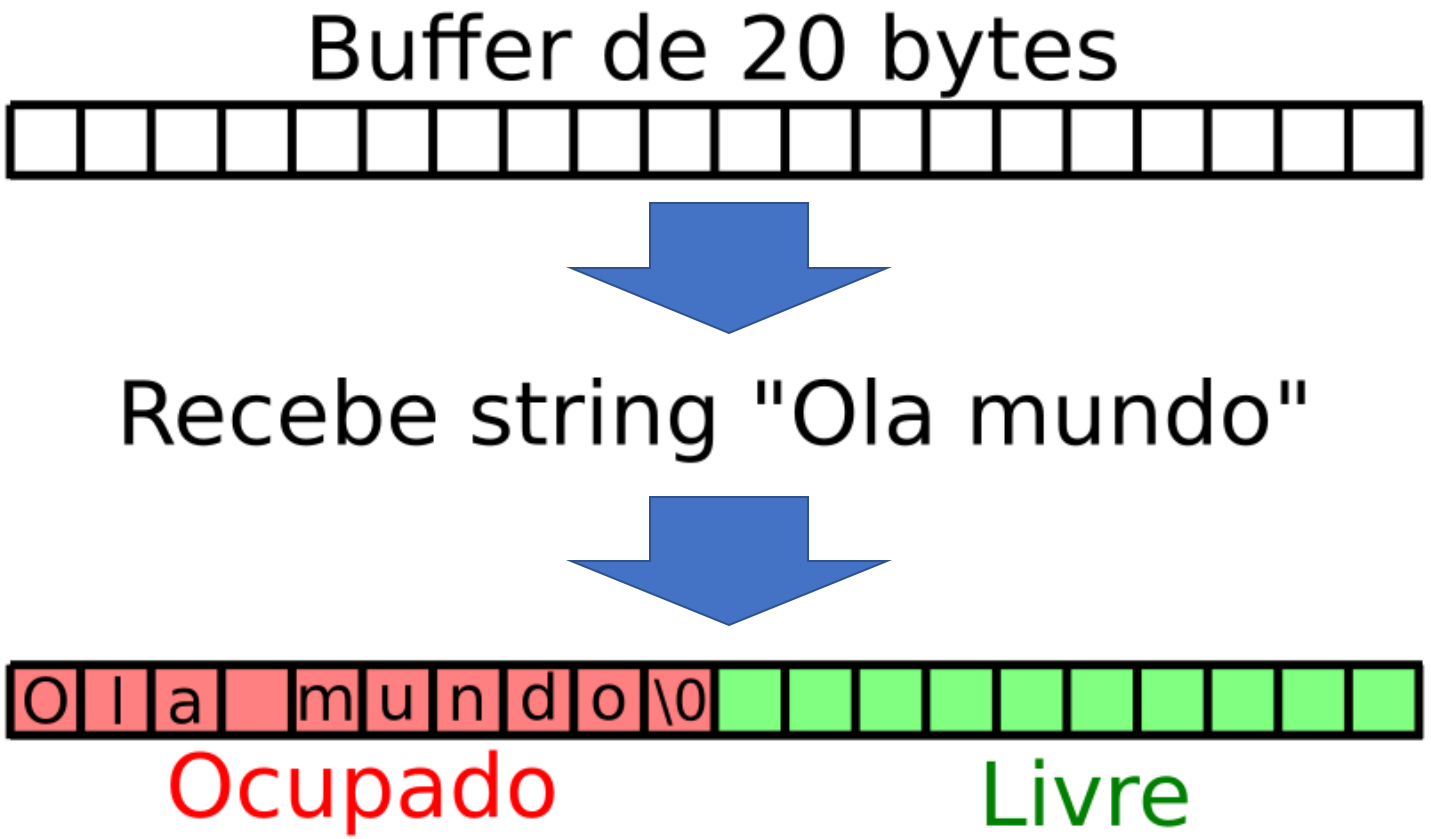
# Strings - Criação

- Como visto anteriormente, uma string é um vetor de caracteres onde o último caractere é dado por '\0' (indicador de término)
- Por exemplo:
  - **char** string[4] = {'O','l','a','\0'};
- Uma forma mais simples de criar a mesma string acima é:
  - **char** string[] = "Ola";
- Neste caso o compilador já entende que deve ser criado um vetor de N+1 posições, onde N é o número de caracteres (incluindo espaços), e 1 é indicador de término ('\0')

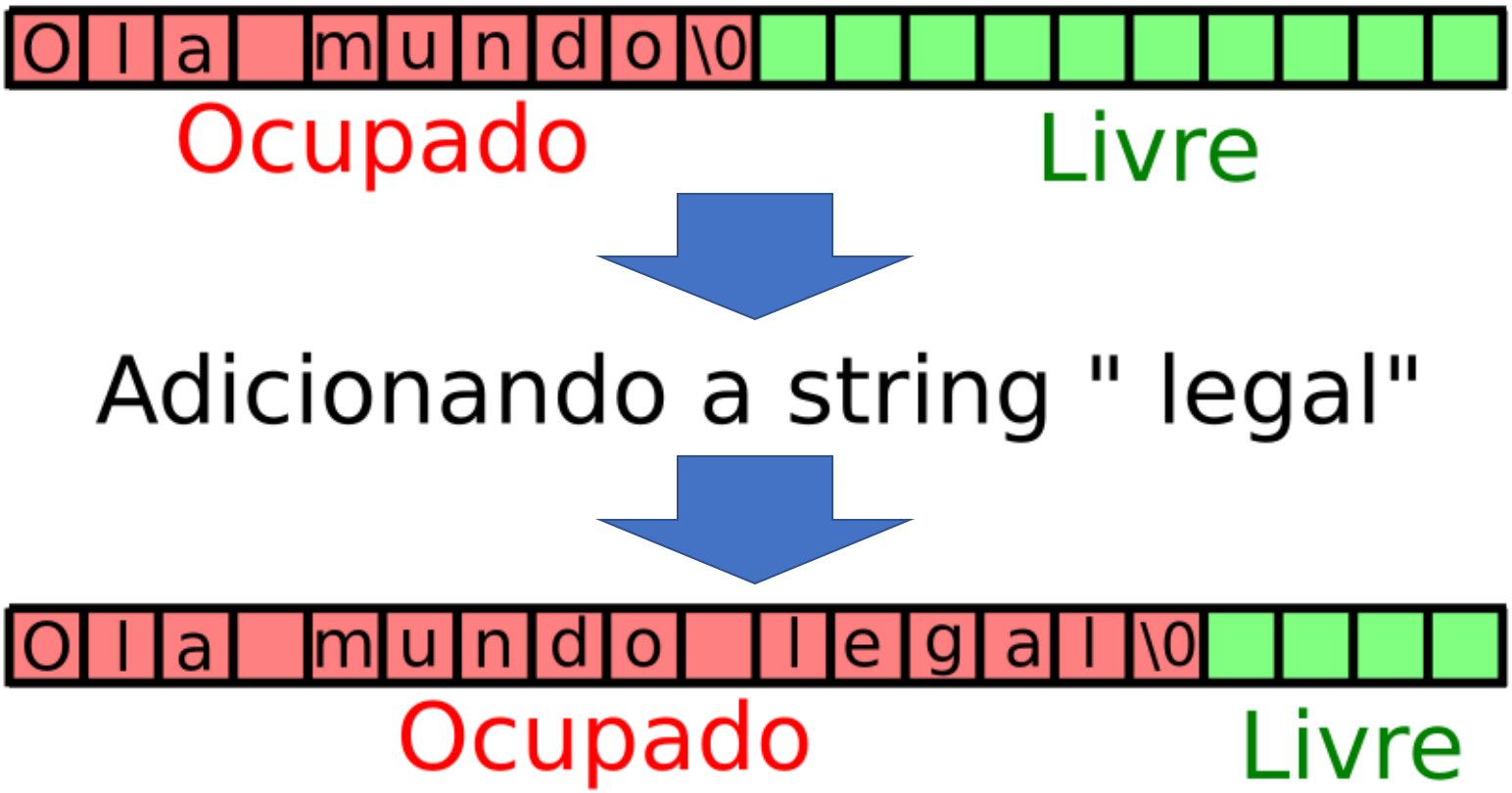
# Strings - Criação

- A criação de strings também pode ser "sobredimensionada", da seguinte forma:
  - **char** string[5000] = "Ola";
- Repare que "Ola" ocupa apenas 4 posições em um vetor de 5000, logo existem outras 4996 posições que estão livres
- Nestas posições é possivel adicionar mais dados textuais
- Chamamos este tipo de vetor por "**buffer**"

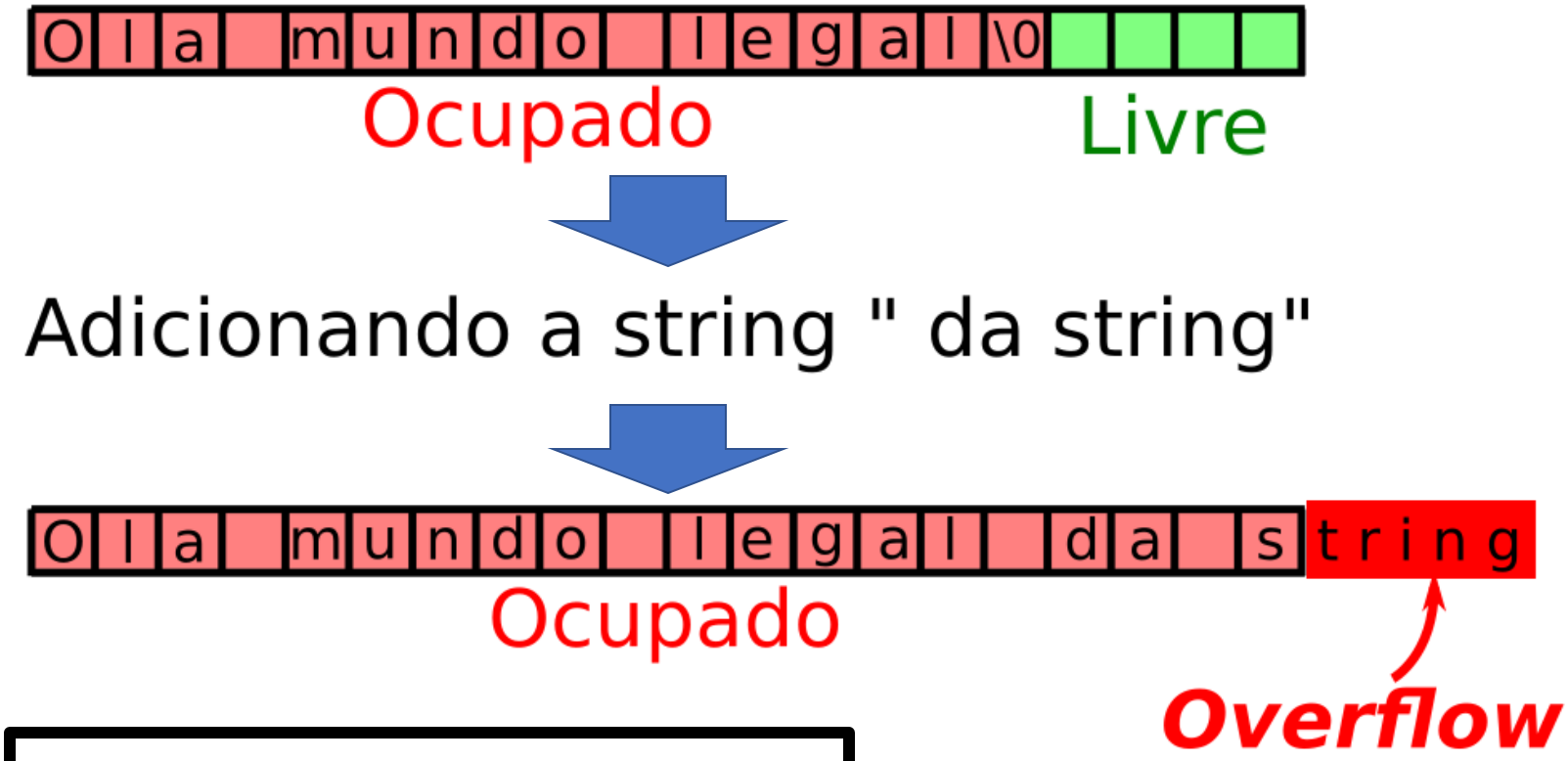
# Buffer


Buffer de 20 bytes
Recebe string "Ola mundo"
O l a m u n d o \0
Ocupado
Livre

# Buffer


O l a m u n d o \0
Ocupado
Livre
Adicionando a string " legal"
O l a m u n d o l e g a l \0
Ocupado
Livre

# Buffer




Problemas de execução!!!

# Strings - Impressão

- Para imprimir uma variável do tipo string basta utilizar a função **printf** com o especificador de formato **%s** :

```
char string[] = "programar em C!";
printf("Eu gosto de %s", string);
```

- Note por este exemplo que o primeiro argumento do **printf** também é uma string
- Outra forma de se obter o mesmo efeito é:

```
printf("Eu gosto de %s", "programar em C!");
```

# Função SIZEOF

- A função **sizeof** retorna a quantidade de bytes que uma variável ocupa na memória
- Sabendo que uma variável caractere do tipo **CHAR** ocupa 1 byte, um vetor de 10 caracteres ocupará 10 bytes
- Da mesma forma, um número inteiro do tipo **INT** ocupa 4 bytes, logo um vetor de 10 números inteiros ocupará 40 bytes
- Você pode verificar isso da seguinte forma:

```
char string1[4] = {'O','l','a','\0'};
char string2[]  = "Ola";

printf("Tamanho string 1: %d \n",sizeof(string1));
printf("Tamanho string 2: %d \n",sizeof(string2));

int vetor[] = {0,1,2,3,4,5,6,7,8,9};
printf("Tamanho vetor: %d \n",sizeof(vetor));
```

Altere a string2 para conter o seu nome completo, por exemplo

# Exercício em sala

- Crie uma string com a seguinte frase: "Eu odeio programar em C" e a imprima na tela.

- Agora, **sem** criar outra string ou modificar a atual, substitua a palavra "odeio" por "adoro". Faça isto alterando caractere por caractere.

# Operações com string

- Operações básicas com strings:

- Contagem de caracteres:
  - "Ola mundo!" => 10
- Concatenação:
  - Junção("Ola"," mundo") = "Ola mundo"
- Comparação:
  - "Ola" == "Ola" ? => Sim
  - "Ola" == "Mundo" ? => Não

string.h

- Conversão:
  - "123" => 123 (Inteiro)
  - "0.5" => 0.5 (Real)

stdlib.h

# Contagem

- Para fazer a contagem de caracteres em um vetor, utilize a função **strlen**
  - Conta todos os caracteres até encontrar um indicador de fim da string (\0)
- Modo de uso:
  - int **strlen**(char[])
- Exemplo:

```
char string[] = "Quantos caracteres tem aqui?";
int r = strlen(string);
printf("A frase '%s' possui %d caracteres",string,r);
```

- Resulta em:
  - A frase 'Quantos caracteres tem aqui?' possui 28 caracteres

# Contagem

- Agora considerando o seguinte trecho de código:

```
char string[] = "Quantos caracteres tem aqui?";
printf("A frase '%s' possui %d caracteres \n",string,strlen(string));
printf("A frase '%s' possui %d caracteres \n",string,sizeof(string));
```

  - Os resultados são iguais?
  - Se não são, explique o porquê.

# Concatenação

- Para juntar duas strings, usa-se a função **strcat**
- Usar esta função exige muito cuidado, pois ela adiciona a segunda string na primeira. Logo, a primeira string deve estar dentro de um vetor grande o suficiente para esta operação (ver slides sobre ***buffer***)
- Exemplo:

```
char string1[20] = "Ola ";
char string2[] = "Mundo";
strcat(string1,string2);
printf("%s",string1);
```

  - Note que string1 é um vetor de 20 caracteres, mas só ocupa 5 deles (3 letras "Ola" + 1 espaço em branco + '\0')
  - Como ainda sobram 15 caracteres, é possível acomodar a string2 ao final
  - O **strcat** remove o '\0' da primeira string, e o substitui pelo primeiro caractere da segunda string, copiando os demais na sequência.

# Concatenação

- Concatenando de forma segura (opcional):

```
char string1[20] = "Ola ";
char string2[] = "Mundo";

if (sizeof(string1) - strlen(string1) > strlen(string2)) {
    strcat(string1,string2);
}
printf("%s",string1);
```

- Explicação:
  - **sizeof**(string1): tamanho do vetor
  - **strlen**(string1) e **strlen**(string2): quantidade de caracteres que fazem parte da string
  - "**sizeof**(string1) - **strlen**(string1)": número de espaços vazios disponíveis no vetor string1 ao término da string (do '\0' em diante), parte LIVRE do buffer

# Comparação

- Para comparar o conteúdo de duas strings, usa-se a função **strcmp**()
  - Mode de uso:
  - int **strcmp**(char[],char[])

- **strcmp()** retorna 0 (zero) se as strings são iguais, ou outro valor caso não. Exemplo:

```
char string1[20] = "Ola";
char string2[] = "Mundo";
char string3[] = {'O','l','a','\0'};

if (strcmp(string1,string2) == 0) {
    printf("string1 eh IGUAL string2\n");
}
else {
    printf("string1 eh DIFERENTE string2\n");
}

if (strcmp(string1,string3) == 0) {
    printf("string1 eh IGUAL string3\n");
}
else {
    printf("string1 eh DIFERENTE string3\n");
}
```

Saída:

```
string1 eh DIFERENTE string2
string1 eh IGUAL string3
```

# Conversão

- Conversão de números em strings: usa-se as funções **atoi()** e **atof()**
- O **atoi** converte uma string contendo um conjunto de dígitos para um número inteiro
- O **atof** converte uma string contendo um conjunto dígitos + ".", para um número real

**atof** e **atoi** fazem parte da biblioteca **stdlib.h**

# Conversão

- Exemplo de conversão usando **atof** e **atoi**

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

int main() {

    char string1[] = "0.123";
    float numerof = atof(string1);
    printf("Numero real: %f \n",numerof);

    char string2[] = "8291";
    int numeroi = atoi(string2);
    printf("Numero inteiro: %d \n",numeroi);

}
```

Saída:

```
Numero real: 0.123000
Numero inteiro: 8291
```

# Exercícios em sala

1) Crie um programa com uma variável do tipo string para cada parte do seu nome, depois concatene todas estas strings na primeira string, e imprima o resultado (que deverá ser seu nome completo). Lembre que a primeira string é um buffer.

   Caso seu nome tenha apenas dois nomes (nome e um sobrenome), utilize o nome completo da imperador Dom Pedro I

2) Crie um programa que conte quantos caracteres existem na string "Eu estudo fisica na UEL e adoro programar em C", imprimindo na tela a string e a contagem.

3) Agora resolva o exercício anterior sem utilizar a função **strlen**()

# Leitura (scanf)

- Até agora utilizamos a função scanf() para a leitura de diversos tipos de dados
- Porém, para strings, o scanf **não** é adequado pois ele interpreta espaços em branco como sendo o final da string
- Por exemplo, teste seguinte código no seu computador:

```
char nome_completo[200];
printf("Digite o seu nome completo: ");
scanf("%s",nome_completo);
printf("Nome completo: %s",nome_completo);
```

- O que acontece?

# Leitura (scanf)

- Para o código anterior, teremos a seguinte saída:

**Saída:**

```
Digite o seu nome completo: Pedro Alvares Cabral
Nome completo: Pedro
```

- Apesar de exister o especificador de formato **%s** referente a strings, o scanf termina sua leitura no primeiro espaço, insering '\0' em seu lugar.

- Então como podemos fazer para ler strings com espaços?

# Leitura (gets)

- Utilizar a função **gets()**
  - Modo de uso: gets(char[]);
- O gets() lê todos os caracteres digitados pelo usuário até encontrar uma quabra de linha (\n), que é inserido quando o usuário pressiona ENTER.
- Para usa-lo simplesmente passe como parâmetro uma string:

```
char nome_completo[200];
printf("Digite o seu nome completo: ");
gets(nome_completo);
printf("Nome completo: %s",nome_completo);
```

# Leitura (gets)

- Resultado para o código anterior:

**Saída:**

```
Digite o seu nome completo: Pedro Alvares Cabral
Nome completo: Pedro Alvares Cabral
```

- **Cuidado**: o gets não verifica se a string passada (ou buffer) é ou não maior do que a entrada digitada pelo usuário!
  - Por exemplo, o código abaixo com a entrada acima:

```
char nome_completo[10];
printf("Digite o seu nome completo: ");
gets(nome_completo);
printf("Nome completo: %s\n",nome_completo);
```

# Leitura (fgets)

- Devido ao perigo potencial dado pelo gets(), esta função se tornou absoleta em C no padrão 2011
- Nos padrões atuais, usa-se a função fgets()
- O fgets é semelhante ao gets com a diferença de que é necessário também passar como parâmetro o tamanho da string:
  - fgets(char[],int,stdin);
    - char[] : string
    - int: tamanho da string
    - stdin: indica leitura do teclado

**fgets** é apenas menção, e **não** será o cobrado seu uso neste curso

# Exercícios em sala

- Crie um programa que leia seu nome completo e substitua espaços em branco pelo caractere "_", imprimindo a saída.

**Exemplo de saída:**

```
Digite o seu nome completo: Pedro Alvares Cabral
Pedro_Alvares_Cabral
```

- Crie um programa que leia dois números reais, e imprima a soma destes números. **Não utilize a função scanf.**

# Exercícios em Sala

- Faça um programa que permita ao usuário digitar um número inteiro, e depois imprima na tela a versão escrita de cada dígito do número digitado.

**Exemplo de saída:**

```
Numero: 5819233
cinco oito um nove dois tres tres
```