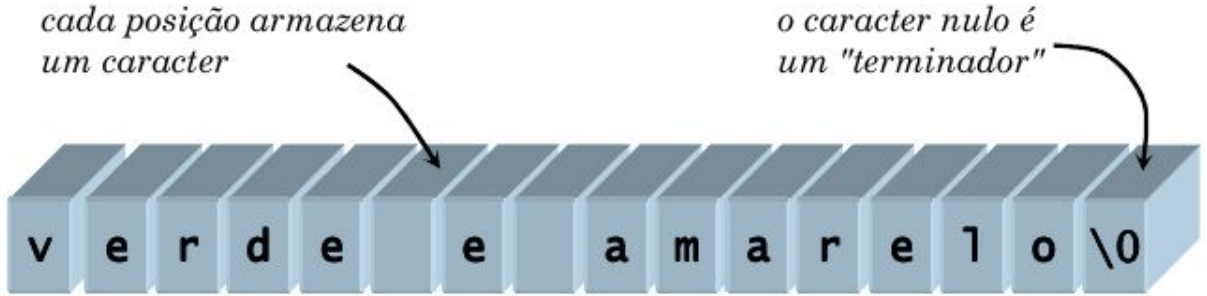
Linguagem de Programação

Strings

Strings

- String não é um tipo básico em C, como em outras linguagens
- É uma série de caracteres terminada com um caractere nulo (‘\0’)
- Representada por um vetor de **char**
  - permitindo o acesso individual de cada caractere, o que aumenta a flexibilidade de manipulação da string
- Por ex., a string “*verde e amarelo*” é armazenada da seguinte maneira na memória:



Inclusão do ‘\0’

- Devido à necessidade do ‘\0’, vetores que armazenam strings devem ter uma posição a mais
- Quando a string é constante, o ‘\0’ é adicionado automaticamente pelo compilador
- Por ex.:

```
#include <stdio.h>
void main(void) {
 printf("Espaço alocado = %d bytes\n",
sizeof("verde e amarelo") );
}
```

- **Saída:** Espaço alocado = 16 bytes
- De fato o ‘\0’ é inserido, pois a string possui apenas 15 caracteres

Inclusão do ‘\0’

- No uso de *strings* variáveis, o ‘\0’ é responsabilidade do programador reservar o espaço adicional
- **Lembre:** o compilador não verifica consistência de indexação!
- Por ex.:

```
void main(void) {
    char n[21];
    printf("Qual o seu nome? ");
    gets(n);
    printf("Olá, %s!",n);}
```

- nesse exemplo, a função gets(n) lê a string do teclado e armazena em *n*
- o **<enter>** digitado é automaticamente substituído por **‘\0’**
  - Então não precisamos nos preocupar em por o ‘\0’

Inicialização de Strings

- Como qualquer outro vetor, *strings* podem ser inicializadas quando declaradas
  - sintaxe convencional
    - `char convencional[3]={'o','i','\0'};`
    - obrigatório por o '\0'
  - sintaxe própria de *strings*
    - `char propria[3]="oi";`
    - '\0' é colocado automaticamente

- Exemplo com erro de declaração:

```
#include <stdio.h>
void main(void) {
  char x[] = "um";  /* inclui '\0'  */
  char y[] = {'d','o','i','s'};  /* não
inclui '\0'
  printf("%s \t %s \n", x, y);
}
```

  - **saída:** *um doisum*
    - a 1ª string é mostrada corretamente
    - na 2ª, como não se sabe onde termina a string, o compilador exibe caracteres até encontrar algum '\0'

Inicialização de Strings

- **“Uma string é um ponteiro”**
  - Arrays são ponteiros para o seu primeiro elemento
  - Como strings são arrays de caracteres, strings também são ponteiros
- Uma string pode ser atribuída em uma declaração para uma variável do tipo `char*`
  - `char *corPtr = "azul";` equivale a `char cor[] = "azul";`
    - Cria uma variável ponteiro **corPtr** que aponta para a string **"azul"** em algum lugar da memória.

Dica de portabilidade: *se for modificar a string, a armazene em um array. Não inicialize como sendo um variável do tipo* `char*`*, pois alguns compiladores podem colocá-la em um local da memória onde não pode ser modificada.*

Leitura de *strings*

- ***scanf():***
  - lê uma string até o encontro do primeiro espaço em branco (espaço, tab, nova linha, etc)
  - ex.:

```
char str1 [80], str2[80];
printf ("Entre com o sobrenome: ");
scanf ("%s",str2);
//especificando o tam. da string a ser lida
scanf ("%79s",str1);
```

  - **obs.:** note que não é preciso por o '**&**', pois a passagem do vetor por si só já informa o endereço da 1ª posição

- ***fgets():***
  - lê uma linha inteira como *string*, até aparecer o '\n' (nova linha)
  - ex.:

```
char nome[30];
printf("Entre com o nome: ");
fgets(nome, sizeof(nome), stdin);
printf("Nome: %s \n");
```

  - `sizeof(nome)`: limita a leitura para o tamanho exato de **'nome'**.

- **gets():**
  - foi removida da mais recente revisão do C standard (2011)
  - pois permite a entrada de qualquer quantidade de caracteres, podendo causar overflow

Erros comuns de programação...

- A função scanf() não lê o caractere ‘\n’
- Vamos testar o seguinte código:

```
void main(void) {
 int idade;
 char nome[30];
 printf("Entre com a idade: ");
 scanf("%d", &idade);
 printf("Entre com o nome: ");
 fgets(nome, 30, stdin);
 printf("idade:%d  nome: %s \n",idade,nome);
}
```

- **Encontraram algum problema?**

- **O que geralmente ocorre:**
  - ao digitar um valor e pressionar <enter>, o scanf() lê o valor e deixa o *‘\n’* no buffer (stdin)
  - em seguida a função fgets() lê a próxima linha (apenas *‘\n’*), e não lê o que deveria ser a entrada de fato

- **Como resolver?**

1) usar scanf, caso precise ler apenas 1 string

```
printf("idade:");
scanf("%d", &idade);
printf("Nome: ");
scanf("%s", nome);
```

2) usar getchar() para ler o ‘\n’, caso precise da linha inteira

```
printf("idade:");
scanf("%d",&idade);
printf("nome:");
getchar();
fgets(nome, 30, stdin);
```

Erros comuns de programação...

- **Como resolver?**

3)

```
scanf(" %[^\n]s",nome);
```

- O espaço no início diz ao scanf() para pular qualquer caractere de “espaço em branco” (incluindo nova linha), antes de ler o próximo caractere
- Ao usar “^\n”, somos capazes de escrever strings com espaços e armazenar em uma variável, até que \n seja encontrado

Funções para manipulação de strings

- Para funções de string, deve-se incluir a biblioteca <string.h>
- As principais funções são:
  - **strlen()** : calcula o tamanho de uma

```
char a[20]="Program";
printf("Tam de a = %ld \n",strlen(a));
```

  - **strcpy()** : copia uma string para outra

```
char str1[10]= "oi";
char str2[10],str3[10];
strcpy(str2, str1);
strcpy(str3, "td bem?");
printf("%s %s",str2,str3);
```

  - **strcmp():** compara 2 strings. Retorna um valor inteiro, que se igual a '0', então as 2 strings são iguais

```
char str1[]="abc", str2[]="abC", str3[]="abc";
int result;
// comparando strings str1 e str2
result = strcmp(str1, str2);
printf("strcmp(str1, str2) = %d\n", result);
// comparando strings str1 e str3
result = strcmp(str1, str3);
printf("strcmp(str1, str3) = %d\n", result);
```

Funções para manipulação de strings

- **strcat:** concatena duas strings

```
char str1[] = "Bom ", str2[] = "dia!";
 //concatena str1 e str2, e a string resultante  é armazenada em str1.
strcat(str1,str2);
printf("%s \n",str1);
```

- Há muitas funções para strings na biblioteca <string.h>
  - Em caso de dúvida, consulte https://en.cppreference.com/w/c

Exercício

- Codifique uma função semelhante à strlen(s), que devolve o número de caracteres armazenados na string s.
  - Lembre-se de que o terminador '\0' não faz parte da string e, portanto, não deve ser contado.

Exercício

- Crie um programa em C para checar se uma string é palíndromo ou não
  - ex.:
    - Entrada = RADAR
    - Saída = É palíndromo

Exercício

- Escreva um programa C para contar o número total de palavras em uma string usando loop

Exercício

- O código de César é uma das mais simples e conhecidas técnicas de criptografia. É um tipo de cifra de substituição na qual cada letra do texto é substituída por outra, que se apresenta no alfabeto abaixo dela um número fixo de vezes (***k***). Considera-se a lista de alfabeto como sendo circular.
    - Por ex.: com *k = 3*, A seria substituído por D, B se tornaria E, e assim por diante.
- Utilizando o código de César, crie uma função para criptografar e outra para descriptografar uma string do teclado.