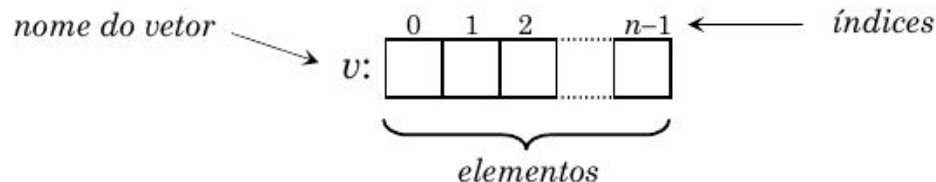


Tópico 4

Vetores, strings e matrizes

Vetores

- Um *vetor* é uma coleção de variáveis de um mesmo tipo, que compartilham o mesmo nome e que ocupam posições consecutivas de memória.
- Cada uma dessas variáveis é identificada por um índice.
 - Se \mathbf{v} é um vetor com n posições, seus elementos são $\mathbf{v}[0]$, $\mathbf{v}[1]$, $\mathbf{v}[2]$, ..., $\mathbf{v}[n-1]$.



Em C os vetores são sempre indexados a partir de zero e, portanto, o último elemento de um vetor de tamanho n ocupa a posição $n-1$ do vetor.

Declarando vetores

- Um vetor para armazenar 5 números inteiros pode ser criado da seguinte maneira: `int v[5];`
- Um vetor pode ser indexado com qualquer expressão cujo valor seja inteiro
 - Por ex. considere $i=5$:

① <code>w[0] = 17;</code>	⑤ <code>w[i] = w[2];</code>
② <code>w[i/2] = 9;</code>	⑥ <code>w[i+1] = w[i]+w[i-1];</code>
③ <code>w[2*i-2] = 95;</code>	⑦ <code>w[w[2]-2] = 78;</code>
④ <code>w[i-1] = w[8]/2;</code>	⑧ <code>w[w[i]-1] = w[1]*w[i];</code>
- O C não faz verifica a consistência dos valores usados como índices.
 - Qualquer valor pode ser usado como índice, mesmo que seja inadequado
 - É responsabilidade do programador definir corretamente os índices

Inicializando um vetor

- Inicializando um vetor sem especificar a quantidade de elementos
 - `int valores[] = {3,5,7}`
- Iniciando apenas alguns elementos do vetor:
 - `int valores[5] = {2,4,6}` será equivalente a `int valores[5] = {2,4,6,0,0}`
 - posições não preenchidas recebem valor zero
- Operador **sizeof()**
 - retorna o tamanho em bytes que uma variável está utilizando na memória
 - Também retorna o tamanho de tipos
 - Ex.: `int v[] = { 3, 4, 6, 7}`
 - o número de elementos de v será `sizeof(v)/sizeof(int)`
 - Uma variável inteira equivale a 4 bytes, então, o nº de elementos em v será `(4+4+4+4)/4`

Vetores como argumento de funções

```
#include <stdio.h>

void funcao(int v[]); //protótipo. Não
necessita explicitar o tamanho do vetor

int main (void) {
    int v[3];
    v[0]=12;
    v[1]=13;
    funcao(v);
    return 0;
}

void funcao(int v[3]){
    printf("v[1]=%d v[2]=%d\n", v[0],v[1]);
}
```

- Assim como nas outras variáveis, a cada chamada de função, é criada uma cópia do vetor passado como parâmetro
 - a função não manipula o vetor original, mas uma cópia sua
- Para manipular o vetor original, esse deve ser passado por **referência**
 - estudaremos isso no assunto sobre **ponteiros**

Exercício

- Escreva um programa onde o usuário preencha um vetor v1 com até n valores ($n \leq 50$). Em seguida, armazene os valores em ordem inversa em outro vetor v2. Por fim, mostre os valores de v2.

Exercício

- Considere dois vetores $v1$ e $v2$ ordenados de maneira não-decrescente. Crie um programa que mescle $v1$ e $v2$ em um terceiro vetor $v3$, de maneira que $v3$ também fique ordenado.
 - **Exemplo**
 - **entrada:**
 - $v1 = 1\ 4\ 6\ 7$
 - $v2 = 2\ 3\ 5\ 9$
 - **Saída:**
 - $v3 = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 9$

Exercício

- Escreva um programa em C para deletar um valor de uma certa posição (fornecida como entrada) de um vetor, em seguida, mostre esse vetor.

Exemplo:

Entrada:

vetor: 1 2 4 6 8

posição: 2

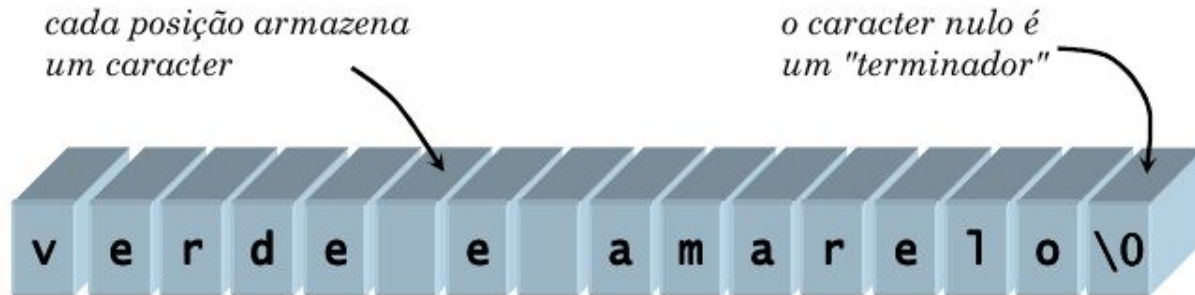
Saída:

vetor: 1 2 6 8

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.:

```
#include <stdio.h>
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'

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
 - Mais detalhes sobre serão apresentados no tópico sobre **Ponteiros**

- ***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);
```

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);
```