

Aula 15: Variáveis compostas - Strings ou array de caracteres*

Joice Otsuka

*Adaptado do livro de Andre Backes

Strings

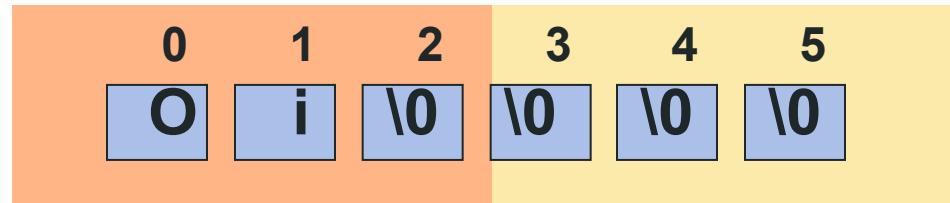
String - Definição

- Sequência de caracteres adjacentes na memória.
- Essa sequência de caracteres, que pode ser uma palavra ou frase
- Strings são arrays do tipo **char**.
- Ex:
 - **char str[6];**

String - Definição

- As strings têm no elemento seguinte à última letra da palavra/frase armazenado um caractere '\0' (barra invertida + zero).
 - O caracter '\0' indica o fim da sequência de caracteres.
- Exemplo
 - `char str[6] = "Oi";`

Região inicializada:
2 letras + 1
caractere
terminador '\0'



String - Definição

- **Importante**

- Ao definir o tamanho de uma string, devemos considerar o caractere '\0'.
- Isso significa que a string **str** comporta uma palavra de no máximo 5 caracteres.

- Exemplo:

- **char str[6] = "Teste";**



T e s t e \0

Definição

- Por se tratar de um array, cada caractere pode ser acessado individualmente por meio de um índice
- Exemplo
 - `char str[6] = "Teste";`

A horizontal sequence of six light blue rectangular boxes, each containing a single character from the string "Teste". The characters are T, e, s, t, e, and \0 respectively. This visualizes the string as an array of characters in memory.

- `str[0] = 'L';`

A horizontal sequence of six light blue rectangular boxes, each containing a single character from the modified string. The characters are L, e, s, t, e, and \0 respectively. This visualizes the state of the array after the character at index 0 was changed from 'T' to 'L'.

Definição

- IMPORTANTE:

- Na inicialização de string, usa-se “**aspas duplas**”.
 - Ex: `char str[6] = "Teste";`

T e s t e \0

- Na atribuição de um caractere, usa-se ‘**aspas simples**’
 - `str[0] = 'L' ;`

L e s t e \0

Definição

- **Importante:**

- “A” é diferente de ‘A’
 - “A” (string)



A \0

- ‘A’ (char)



A

Manipulação de strings

Manipulando strings - Leitura

- **scanf:** lê uma string do teclado e armazena em uma string.
 - Exemplo:

```
char s1[5];
int i;
scanf("%s",s1);
```

//s1 já é um endereço, por isso não colocamos &

- O scanf lerá a entrada até encontrar um espaço em branco ou Enter

Problemas:

- Não permite a entrada de strings com espaço em branco.
- Não é um método seguro, pois não limita o número de caracteres lidos.

```
int main()
{
    int i;
    char c;
    char s1[5];
    char s2[5];
    scanf("%d",&i);
    scanf("%*c %c",&c); // %*c Lê um caractere (\n), mas não armazena em variável
    scanf("%s %s",s1,s2);
    printf("Saída i:%d\n",i);
    printf("Saída c:%c\n",c);
    printf("Saída s1:%s\n",s1);
    printf("Saída s2:%s\n",s2);
    return 0;
}
```

O que acontece se s1 tiver mais do que 4 caracteres?

Manipulando strings - Leitura

- **fgets(str,tamanho,stdin)**
 - permite a leitura de strings do teclado ou de arquivo
 - A função **fgets** recebe 3 argumentos
 - a string a ser lida, **str**;
 - o limite máximo de caracteres a serem lidos, **tamanho**;
 - A variável FILE ***fp**, que está associado ao arquivo de onde a string será lida (ou **stdin** para leitura do teclado)
 - E retorna
 - NULL (0) em caso de erro ou fim do arquivo;
 - O ponteiro para o primeiro caractere recuperado em **str**.

Exemplo

- Função fgets com leitura de entrada pelo teclado.

```
int main() {
    char nome[30];
    printf("Digite um nome: ");
    fgets(nome, 30, stdin);
    printf("O nome digitado foi: %s", nome);

    return 0;
}
```

Manipulando strings - Leitura

- Funcionamento da função **fgets**
 - A função **lê a string até que um caractere de nova linha seja lido ou tamanho-1 caracteres** tenham sido lidos.
 - **A string resultante sempre terminará com '\0'** (por isto somente *tamanho* - 1 caracteres, no máximo, serão lidos).
 - **Se o caractere de nova linha ('\n') for lido, ele fará parte da string (se houver espaço).**
 - Se ocorrer algum erro ou fim de arquivo, a função devolve um ponteiro nulo (**NULL**) em **str**.

Manipulando strings - Leitura

- A função **fgets** possui as seguintes vantagens:
 - pode fazer a leitura a partir de um arquivo de dados e incluir o caractere de nova linha “\n” na string (desde que haja espaço suficiente);
 - especifica o tamanho máximo da string de entrada (mas excesso fica no buffer - stdin)

```
int main()
{
    char str1[5];
    char str2[5];
    fgets(str1,5,stdin);
    fgets(str2,5,stdin);
    printf("%s\n",str1);
    printf("%s\n",str2);
    return 0;
}
```

O que acontece se no primeiro fgets for digitada uma string com mais de 4 caracteres?

Limpando o buffer

- Entre a leitura de duas strings usando o fgets pode ser necessário limpar o buffer do teclado
- Alguns métodos:
 - fflush (stdin): uso **não recomendado**, pois o comportamento é indefinido pelos padrões da linguagem
 - Mais recomendado:
 - **while ((getchar())! = '\n');**
 - **while ((fgetc(stdin))! = '\n');**
 - **while ((getc(stdin))! = '\n');**

Saiba mais: <https://acervolima.com/limpando-o-buffer-de-entrada-em-c-c/>

Saiba mais: <https://acervolima.com/uso-de-fflush-stdin-em-c/>

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

int main() {
    char c;
    char s1[15];
    char s2[15];

    fgets(s1, 15, stdin);

    // consome excesso que da Leitura anterior quando há
    if (strlen(s1) == 14)
        while((c = fgetc(stdin)) != EOF && c != '\n'){}

    fgets(s2, 15, stdin);
    printf("%s\n", s1);
    printf("%s\n", s2);
    return 0;
}
```

Manipulando strings - Escrita

- Para se escrever uma string na tela utilizamos a função **printf()**.

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

- Outra função que permite a escrita de strings é a **fputs()**

```
fputs(str, stdout);
```

Manipulando strings

- Strings são arrays. Portanto, **não** podemos atribuir uma string a outra!

```
#include <stdio.h>

int main() {
    char str1[20] = "CAP 2019 - Turma D";
    char str2[20];
    str2 = str1; X
    return 0;
}
```

Copiando uma string

- O correto é copiar a string elemento por elemento.

```
#include <stdio.h>

int main() {
    int i;
    char str1[20] = "CAP 2019 - Turma D";
    char str2[20];
    for (i=0;i<20;i++){
        str2[i]=str1[i];
    }
    printf("%s\n",str1);
    printf("%s\n",str2);
    return 0;
}
```

Manipulando strings

- A biblioteca padrão C possui funções especialmente desenvolvidas para esse tipo de tarefa
 - `#include <string.h>`

Manipulando strings - Tamanho

- `strlen(str)`: retorna o tamanho da string str. Ex:

```
char str[15] = "teste";
printf("%d", strlen(str));
```

- Neste caso, a função retornará 5, que é o número de caracteres na palavra “teste” e não 15, que é o tamanho do array.
 - O ‘\0’ também não é considerado pela `strlen`, mas vale lembrar que ele está escrito na posição `str[5]` do vetor.

Manipulando strings - Cópia

- **strcpy(dest, fonte)**: copia a string contida na variável **fonte** para **dest**.
- Exemplo

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

int main() {
    char s1[101], s2[101];
    scanf("%s", s1);

    strcpy(s2, s1);
    printf("%s\n", s2);

    return 0;
}
```

Manipulando strings - Concatenação

- **strcat(dest, fonte)**: concatena duas strings.
- Neste caso, a string contida em **fonte** permanecerá inalterada e será anexada ao final da string de **dest**.
- Exemplo

```
char str1[15] = "bom ";
char str2[15] = "dia";
strcat(str1, str2);
printf("%s", str1);
```

Manipulando strings - Comparação

- **strcmp(str1, str2)**: compara duas strings e
 - retorna VALOR NEGATIVO se **str1** “menor”* que **str2**
 - retorna ZERO se as strings forem iguais.
 - retorna VALOR POSITIVO se **str1** “maior”* que **str2**

*ordem lexicográfica

- Exemplo

```
if(strcmp(str1,str2) == 0)
    printf("Strings iguais");
else
    printf("Strings diferentes");
```

Leitura - fim de arquivo

A entrada termina com fim-de-arquivo.

O **scanf** retorna **EOF** quando chega ao fim de arquivo

```
int main() {  
    char nome[11];  
  
    while (scanf("%s", nome) != EOF) {  
        printf("String lida = %s\n", nome);  
    }  
  
    return 0;  
}
```

```
#include <stdio.h>

int main() {
    int a, b;
    int result;

    printf("Enter two integers: ");
    result = scanf("%d %d", &a, &b);

    if (result == 2) {
        printf("Successfully read two integers: a = %d, b = %d\n", a, b);
    } else if (result == 1) {
        printf("Successfully read one integer. The other value is missing or invalid");
    } else if (result == 0) {
        printf("No integers were read. Invalid input.\n");
    } else if (result == EOF) {
        printf("End of input or input error.\n");
    }
    return 0;
}
```

scanf
retorna o número de itens lidos ou EOF, em caso de erro ou fim de arquivo

Leitura - fim de arquivo

A entrada termina com fim-de-arquivo.

O **fgets** retorna NULL(0) quando chega ao final do arquivo

```
int main() {  
    char nome[11];  
  
    while (fgets(nome, 11, stdin)) {  
        printf("String lida = %s", nome);  
    }  
  
    return 0;  
}
```

Observação final

- Ao inicializar uma string em sua declaração, as regiões do vetor que não foram utilizadas pela string são preenchidas com zeros ('\0')
- Entretanto, esse comportamento não ocorre com o **strcpy**, **scanf** e **fgets**. Nessas funções as posições não usadas não são alteradas (ficam lixos).
 - Ex: **char str[6] = "Oi";**

The diagram illustrates the memory layout of the string "Oi" in a character array of size 6. It consists of six light blue rectangular boxes arranged horizontally. The first two boxes contain the characters 'O' and 'i' respectively. The remaining four boxes are empty and contain the character '\0', representing null bytes.

O	i	\0	\0	\0	\0
---	---	----	----	----	----

Observação final

- Exemplos

- `char str[6] = "Teste";`



- `scanf("%s", str); //digite "Ola" no prompt`



- `strcpy(str, "Oi");`



Referências

- BACKES, André. **Linguagem C**: completa e descomplicada. Rio de Janeiro: Elsevier, 2013. 371 p. ISBN 978-85-352-6855-3. Disponível na Biblioteca.