

# MC-102 — Aula 11

## Strings

Eduardo C. Xavier

Instituto de Computação – Unicamp

28 de Junho de 2017

# Roteiro

## 1 Strings

- Definição de Strings em C
- Leitura e Escrita de Strings
- Inicialização de Strings
- Strings: Exemplos

## 2 Biblioteca string.h

## 3 Processamento de Texto

## 4 Exercícios

# Strings em C

- A linguagem C não possui o tipo *string* explicitamente, mas podemos considerar um vetor de caracteres como uma *string*.
- Em C uma string é sempre terminada pelo caractere especial: `'\0'`

## Definição

Uma string em C corresponde a um vetor de caracteres terminado pelo caractere especial `'\0'`.

- Sempre declare uma string com um caractere a mais do que precisa, já que também será preciso armazenar o caractere `'\0'`.
  - ▶ Se por exemplo, estivermos trabalhando com strings de 10 caracteres, declare uma variável com tamanho 11:

```
char st[11];
```

# Strings em C

- **Lembre-se:** o caractere `'\0'` identifica o final da string.
- No programa abaixo gostaríamos que fosse impresso "ola".

```
int main(){  
    char st[80];  
  
    st[0] = 'o';  
    st[1] = 'l';  
    st[2] = 'a';  
  
    printf("%s\n", st);  
}
```

- Mas as vezes será impresso uma palavra diferente, como "ola8uj", pois não identificamos o final da string.

# Strings em C

- A versão correta do programa seria esta abaixo.

```
int main(){  
    char st[80];  
  
    st[0] = 'o';  
    st[1] = 'l';  
    st[2] = 'a';  
    st[3] = '\\0';  
  
    printf("%s\\n", st);  
}
```

- Note que a variável **st** pode armazenar strings com até 79 caracteres, mas neste exemplo só estamos usando 3 (fora o '\\0').

# Leitura e Escrita de Strings

- Para ler ou imprimir uma string do teclado usamos o operador especial `%s`.

```
int main(){
    char st[80];
    int id;

    printf("Entre com o nome:");
    scanf("%s",st);
    printf("Entre com a idade:");
    scanf("%d",&id);

    printf("Digitado: %s e %d\n",st,id);
}
```

- Note que para strings não é utilizado o `&` antes do identificador da variável no comando `scanf`.
- O comando `scanf` automaticamente coloca um `'\0'` ao final da string lida.

## Leitura e Escrita de Strings

- O comando **scanf** com o operador **%s** faz com que a leitura da string termine em uma quebra de linha ou em um espaço.

```
int main(){
    char st[80];
    int id;

    printf("Entre com o nome:");
    scanf("%s",st);
    printf("Entre com a idade:");
    scanf("%d",&id);

    printf("Digitado: %s e %d\n",st,id);
}
```

- No exemplo acima, se digitarmos

```
Joao da Silva
19
```

será salvo apenas "Joao" em **st**, e um valor diferente de 19 em **id**.

- Isto ocorre pois o **scanf** lê a string até o primeiro espaço, e converte o próximo dado (que é a string "da") em um inteiro.

# Leitura e Escrita de Strings

- Para ler strings **incluindo espaços** use o comando **fgets** cuja sintaxe é:

`fgets(identificador, limite, stdin);`

onde **identificador** é o nome da variável para onde será lida a string, **limite-1** é a quantidade máxima de caracteres que poderá ser lida, e **stdin** é uma palavra reservada que indica que a leitura se dará da entrada padrão.

- Neste comando serão lidos todos os caracteres até a quebra de linha, e todos serão armazenados na variável, incluindo a quebra de linha. Ao final é incluído um `'\0'`.
- Caso **limite-1** caracteres tenham sido lidos, a função para a leitura antes da quebra de linha.



# Leitura e Escrita de Strings

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

```
int main(){  
    char st[80];  
    int id;  
  
    printf("Entre com o nome:");  
    fgets(st, 80, stdin);  
    printf("Entre com a idade:");  
    scanf("%d",&id);  
  
    printf("Digitado: %s e %d\n",st,id);  
}
```

- No exemplo acima se digitarmos

Joao da Silva  
19

será salvo "Joao da Silva\n\0" em **st**, e o valor 19 em **id**.

- Como **st** pode armazenar até 80 caracteres, usamos este valor como parâmetro para o limite de caracteres que podem ser lidos do teclado, já que serão lidos até 79, e deve ser incluído o '\0' no final.

# Leitura e Escrita de Strings

- Em geral é mais seguro usar o **fgets** do que o **scanf** pois o primeiro especifica o tamanho máximo da string a ser lida.
- Se um usuário digitar uma string maior do que o vetor declarado, o **scanf** pode dar problemas pois irá ler todos caracteres até um espaço ou `'\0'` ou `'\n'`, sobreescrevendo posições inválidas da memória.
- Existe uma ataque conhecido como *buffer overflow* que explora justamente este problema do **scanf**.
- Já o **fgets** sempre lê uma string até o máximo especificado.

# Inicialização de Strings

- Em algumas situações, ao criarmos uma string, pode ser útil atribuir valores já na sua criação.
- No caso de strings, podemos atribuir diretamente uma constante string para a variável.

## Exemplo

```
char st[100] = "sim isto é possível";
```

- O comando de inicialização automaticamente insere o caractere '\0' no final da string.

# Strings: Exemplos

- Ler uma string de até 79 caracteres (incluindo '\n') e salvar a inversa desta em um vetor.
- Imprimir a inversa da string lida.

# Strings: Exemplos

- Primeiramente determinamos o final da string.

```
int main(){
    char st[80], stInv[80];
    int i, j, fim;

    fgets(st, 80, stdin);

    //Determinamos o final da string
    fim = 0;
    while(st[fim] != '\0' && st[fim] != '\n')
        fim++;
    ...
}
```

# Strings: Exemplos

- Depois escrevemos os caracteres em **stInv** na ordem inversa de aparição em **st**.

```
int main(){
    char st[80], stInv[80];
    int i, j, fim;

    fgets(st, 80, stdin);

    //Determinamos o final da string
    ...

    //Escrevemos os caracteres na inversa
    i = fim - 1;
    j = 0;
    while(j < fim){
        stInv[j] = st[i];
        i--;
        j++;
    }
    stInv[fim] = '\0';

    printf("Inversa:\n%s\n", stInv);
}
```

# Strings: Exemplos

```
#include <stdio.h>

int main(){
    char st[80], stInv[80];
    int i, j, fim;

    fgets(st, 80, stdin);

    //Determinamos o final da string
    fim = 0;
    while(st[fim] != '\0' && st[fim]!='\n')
        fim++;

    //Escrevemos os caracteres na inversa
    i = fim-1;
    j=0;
    while(j<fim){
        stInv[j] = st[i];
        i--;
        j++;
    }
    stInv[fim]='\0';

    printf("Inversa:\n%s\n", stInv);
}
```

# Strings: Exemplos

A mesma coisa mas com laço **for**:

```
int main(){
    char st[80], stInv[80];
    int i, j, fim;

    fgets(st, 80, stdin);

    //Determinamos o final da string
    for(fim = 0; st[fim] != '\0' && st[fim] != '\n'; fim++)
        ;

    //Escrevemos os caracteres na inversa, st2
    for(i=fim-1, j=0; j<fim; i--, j++)
        stInv[j] = st[i];

    stInv[fim]='\0';

    printf("Inversa:\n%s\n", stInv);
}
```



# Biblioteca string.h

- A biblioteca **string.h** possui várias funções úteis para se trabalhar com strings.
- Vamos apresentar algumas funções comuns:
  - ▶ **char \*strcat(char \*s1, const char \*s2)** : Para fazer a concatenação de strings.
  - ▶ **int strcmp(const char \*s1, const char \*s2)** : Para fazer a comparação lexicográfica (utilizada em ordenação) de duas strings.
  - ▶ **char \*strcpy(char \*s1, const char \*s2)** : Para fazer a cópia de strings.
  - ▶ **int strlen(const char \*s1)** : Para se determinar o tamanho de uma string.

# Biblioteca string.h

Exemplo de uso da função **strcat** para fazer concatenação de strings.

- A função recebe duas strings como parâmetro e concatena a string segundo parâmetro no final da string primeiro parâmetro.
- Deve haver espaço suficiente na primeira string, caso contrário ocorrerá um erro.

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

int main(){
    char s1[80]="ola ", s2[80]="turma de 102!";

    //concatena s2 no final de s1
    strcat(s1, s2);

    printf("%s\n", s1);
}
```

Saída será

ola turma de 102!

## Biblioteca string.h

Exemplo de uso da função **strcmp** para fazer comparação de strings.

- A função recebe duas strings **s1** e **s2** como parâmetro e devolve:
  - ▶ 0 caso as duas strings sejam iguais.
  - ▶ um valor menor que 0 caso **s1** seja lexicograficamente menor que **s2**.
  - ▶ um valor maior que 0 caso **s1** seja lexicograficamente maior que **s2**.

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

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

```
int main(){
    char s1[80]="aab", s2[80]="aac";
    int r;
    r = strcmp(s1, s2);
    if(r < 0)
        printf("%s vem antes que %s\n", s1, s2);
    else if(r > 0)
        printf("%s vem antes que %s\n", s2, s1);
    else
        printf("sao iguais\n");
}
```

Saída será

aab vem antes que aac

# Biblioteca string.h

Exemplo de uso da função **strcpy** para fazer cópia de strings.

- A função recebe duas strings como parâmetro e copia a string segundo parâmetro na string primeiro parâmetro.

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

int main(){
    char s1[80], s2[80]="ola pessoal";

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

Saída será

ola pessoal

# Biblioteca string.h

Exemplo de uso da função **strlen** para calcular o tamanho de uma string.

- A função recebe uma string como parâmetro e devolve o número de caracteres na string até o `'\0'`.

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

int main(){
    char s1[80]="ola pessoal";
    int t;

    t = strlen(s1);
    printf("%d\n", t);
}
```

Saída será

11

# Processamento de Texto

- Como exemplo de uso de strings vamos implementar duas funcionalidades básicas de processadores de texto:
  - 1 Contar o número de palavras em um texto.
  - 2 Fazer a busca de uma palavra em um texto.

# Processamento de Texto

Programa que conta o número de palavras em textos sem pontuação:

```
int main(){
    char s[80];
    int i=0, n=0;

    fgets(s, 80, stdin);

    while(s[i]!='\n' && s[i] != '\0'){ //Enquanto não terminou o texto

        while(s[i]==' ') //Pula possíveis espaços
            i++;
        //Achou o começo de uma palavra ou o fim do texto

        if(s[i]!='\n' && s[i]!='\0'){ //Se achou uma palavra
            n++; //incrementa numero de palavras
            while(s[i]!=' ' && s[i] != '\n' && s[i]!='\0')//passa pela palavra
                i++;
        }
    }

    printf("Total de palavras: %d\n", n);
}
```

# Processamento de Texto

- Fazer um programa que acha todas as posições de ocorrência de uma palavra em um texto (de tam. max 79 incluindo incluindo '\n').

Exemplo:

Texto=a tete tetete

Palavra=tete

A resposta é 2, 7 e 9.



# Processamento de Texto

Ideia do algoritmo:

- Para cada possível posição no texto onde a palavra pode iniciar, checamos se a palavra ocorre naquela posição ou não.
- Seja **tamT** (respectivamente **tamP**) o tamanho do texto (tamanho da palavra respectivamente).
- As posições válidas onde a palavra pode iniciar no texto vão de **0** até **tamT - tamP**.

```
int main(){
    char tex[80], pal[80];
    int i, j, iguais;
    int tamP, tamT;

    printf("Digite o texto: ");
    fgets(tex, 80, stdin);
    printf("Digite a palavra: ");
    fgets(pal, 80, stdin);

    tamP = strlen(pal)-1;
    tamT = strlen(tex)-1; //O -1 é pelo \n

    for(i=0; i <= tamT - tamP; i++){
        //Para cada i verificar se palavra
        //ocorre a partir de i
        ...
    }
```

# Processamento de Texto

Como testar se a palavra ocorre exatamente a partir de uma posição *i*?

- Checar se todos os caracteres da palavra são iguais aos do texto a partir de *i*.

```
j = 0;
iguais = 1;
while(j < tamP && iguais){
    if(pal[j] != tex[i+j])
        iguais = 0;
    j++;
}
if(iguais)
    printf("%d\n", i);
}
```

# Processamento de Texto

## Programa completo:

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

int main(){
    char tex[80], pal[80];
    int i, j, iguais;
    int tamP, tamT;

    printf("Digite o texto: ");
    fgets(tex, 80, stdin);
    printf("Digite a palavra: ");
    fgets(pal, 80, stdin);

    tamP = strlen(pal)-1;
    tamT = strlen(tex)-1; //O -1 é pelo \n

    for(i=0; i <= tamT - tamP; i++){
        //Para cada i verificar se palavra
        //ocorre a partir de i
        j = 0;
        iguais = 1;
        while(j < tamP && iguais){
            if(pal[j] != tex[i+j])
                iguais = 0;
            j++;
        }
        if(iguais)
            printf("%d\n", i);
    }
}
```

# Exercício

- Escreva um programa que lê uma string de até 50 caracteres, e imprime "Palindromo" caso a string seja um palindromo e "Nao Palindromo" caso contrário.
- OBS: Um palindromo é uma palavra ou frase, que é igual quando lida da esquerda para a direita ou da direita para a esquerda (espaços em brancos são descartados). Assuma que as palavras são todas em minúsculas e sem acentos.
- Exemplo de palindromo: saudavel leva duas.

# Exercício

- Refaça o exemplo visto em aula de inversão de uma string de tal forma que não seja utilizado nenhum vetor adicional! Ou seja devemos computar a inversa no próprio vetor original.