Vetores de Caracteres (String)

Programação e Desenvolvimento de Software I
Heitor Ramos

Strings

- Representa texto escrito “abcdefghijklmnopqrstuvxz”
- Em C utilizamos um vetor de caracteres

```
char nome[] = "Lorem ipsum dolor";
//[L][o][r][e][m][][i][p][s][u][m][ ][d][o][l][o][r][\0]
```

Strings

- Representa texto escrito “abcdefghijklmnopqrstuvxz”
- Em C utilizamos um vetor de caracteres

```
char nome[] = "Lorem ipsum dolor";
//[L][o][r][e][m][ ][i][p][s][u][m][ ][d][o][l][o][r][\0]
```

- O caractere '\0' representa o fim da string (null em ASCII)
  - Útil quando é alocado mais espaço do que o tamanho da string
  - Sabemos onde termina o texto

Strings

- Representa texto escrito “abcdefghijklmnopqrstuvxz”
- Em C utilizamos um vetor de caracteres

```
char nome[] = "Lorem ipsum dolor";
//[L][o][r][e][m][ ][i][p][s][u][m][ ][d][o][l][o][r][\0]
```

- O caractere '\0' representa o fim da string (null em ASCII)
  - Útil quando é alocado mais espaço do que o tamanho da string
  - Sabemos onde termina o texto

- Caso alocarmos um espaço maior o '\0' nos indica o fim!
  - Dizemos que a string tem lixo no fim
  - Pode ser setado para '\0' dependendo do compilador
  - https://goo.gl/AVUCr9

Strings

- **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.
- Ex: char str[6] = “Teste”;

T	e	s	t	e	\0

Definição

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

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

Manipulando strings - Leitura

- Exemplo de algumas funções para manipulação de strings
- gets(str): lê uma string do teclado e coloca em str. Ex:

```
char str[10];

gets(str);
```

Manipulando strings – Limpeza do buffer

- Às vezes, podem ocorrer erros durante a leitura de caracteres ou strings. Para resolver esses pequenos erros, podemos limpar o buffer do teclado

```
char str[10];

setbuf(stdin, NULL);//limpa o buffer

gets(srt);
```

Manipulando strings - Escrita

- Basicamente, para se escrever uma string na tela utilizamos a função **printf()**.

```
char str[20] = “Hello World”;
printf(“%s”,str);
puts(str) = printf(“%s”, str);
puts(“Texto”) = printf(“Texto\n");
```

Várias formas de declarar strings

```
char nome1[] = “Lorem";
char nome2[] = {'L', 'o', 'r', 'e', 'm','\0'};
char *nome3  = “Lorem";
```

- O nome da string representa o endereço de memória
- Funciona similar a um vetor `nome[0] = ‘L'`
- Alguns truques são complicados mas úteis
  - Só use se souber e entender bem as consequências!

```
char nome[] = 'Lorem';
printf("%s", nome + 3);
//Imprime em
```

Exemplo

```c
#include <stdio.h>
int main(void) {
  char nome[] = "Lorem";
  char nome1[] = {'L', 'o', 'r', 'e', 'm', '\0'}; // note '\0'
  char *nome2 = "Lorem";
  puts(nome+3);  // puts é um printf para strings, só 1
argumento
  return 0;
}
```

Problema

Como saber o tamanho de uma string?
Como comparar 2 strings?

```
char str1[10], str2[10]

int igual =0

if (strlen(str1)!=strlen(str2)
        igual = 0;

else

        for (int i=0; int.....
```

Tamanho

```c
#include <stdio.h>
int main(void) {
  char nome[] = "Lorem";
  int tamanho;
  // Quando que o for abaixo para?!
  for (tamanho = 0; nome[tamanho] != '\0'; tamanho++);
  printf("O tamanho foi %d\n", tamanho);
  return 0;
}
```

Manipulando strings – Limpeza do buffer

- Às vezes, podem ocorrer erros durante a leitura de caracteres ou strings. Para resolver esses pequenos erros, podemos limpar o buffer do teclado

```
char str[10];

setbuf(stdin, NULL);//limpa o buffer

gets(srt);
```

Algumas funções de Strings: Existem várias no <string.h> e <stdio.h>

- strlen(st)
  - retorna o comprimento do string (com exceção do \0)

- strcmp(s1, s2)
  - Comparação por ordem alfabética
  - retorna < 0 se s1 é menor que s2,
  - 0 se s1 é igual a s2,
  - e > 0 se s1 é maior que s2
  - **A comparação entre strings também tem que ser feita caractere a caractere, portanto não se pode usar s1==s2; isso só vai compara os endereços**

Algumas funções de 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.

Algumas funções de Strings

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

```
char str1[100], str2[100];

printf ("Entre com uma string: ");

gets(str1);

strcpy(str2, str1);

printf("%s",str2);
```

Algumas funções de Strings:

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

```
char str1[15] = "bom ";

char str2[15] = "dia";

strcat(str1,str2);

printf("%s",str1);
```

Exercício Poscomp 2009

https://goo.gl/XoMnzc

```c
#include <stdio.h>
#include <string.h>
int main (void) {
  char texto[] = "sem problemas";
  int i;
  for (i = 0; i < strlen(texto); i++) {
    if (texto[i] == ' ') break;
  }
  i++;
  for ( ; i < strlen(texto); i++)
    printf("%c", texto[i]);
  return 0;
}
```

- Qual vai ser a saída?

Como ler uma string com scanf

```c
#include <stdio.h>
int main(void) {
  char texto[100];
  scanf("%s", texto);
  printf("Li %s\n", texto);
  return 0;
}
```

E se a string tiver um espaço?!

```c
#include <stdio.h>
int main(void) {
  char texto[100];
  scanf("%[^\n]", texto);
  printf("Li %s\n", texto);
  return 0;
}
```

E se eu quiser um tamanho fixo!?

```c
#include <stdio.h>
int main(void) {
  char texto[256];
  fgets(texto, 256, stdin); // como é feito no VPL
  printf("Li %s\n", texto);
  return 0;
}
```

Vetores de Strings

- Strings também são variáveis
- Podem ser declaradas dentro de vetores
- Dias da semana abaixo
- Nem todo dia tem 14 caracteres. Lembre-se do '\0'

```
char dia_semana[7][14] =
    {"Domingo", “Segunda“, "Terca", "Quarta", "Quinta", "Sexta",
     "Sabado"};
//...
printf("%s\n", dia_semana[3]);
```

Manipulando strings

- Basicamente, para se ler uma string do teclado utilizamos a função **gets()**.
- No entanto, existe outra função que, utilizada de forma adequada, também permite a leitura de strings do teclado. Essa função é a **fgets()**, cujo protótipo é:

```
char *fgets(char *str, int tamanho,FILE *fp);
```

Manipulando strings

- 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.
- E retorna
  - NULL em caso de erro ou fim do arquivo;
  - O ponteiro para o primeiro caractere recuperado em **str**.

Manipulando strings

- Note que a função **fgets** utiliza uma variável FILE ***fp**, que está associado ao arquivo de onde a string será lida.
- Para ler do teclado, basta substituir FILE ***fp** por **stdin**, o qual representa o dispositivo de entrada padrão (geralmente o teclado).

```
fgets (str, tamanho, stdin);
```

Manipulando strings

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

Manipulando strings

- A função **fgets** é semelhante à função **gets**, porém, com as seguintes vantagens:
    - pode fazer a leitura a partir de um arquivo de dados e incluir o caractere de nova linha **“\n”** na string;
    - específica o tamanho máximo da string de entrada. Evita estouro no buffer;

  Ex: fgets (str, tamanho, stdin);

Manipulando strings

- Basicamente, para se escrever uma string na tela utilizamos a função **printf()**.

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

- No entanto, existe outra função que, utilizada de forma adequada, também permite a escrita de strings. Essa função é a **fputs()**, cujo protótipo é:

  ```
  int fputs (char *str,FILE *fp);
  ```

Manipulando strings

- A função **fputs()** recebe como parâmetro um array de caracteres e a variável FILE ***fp** representando o arquivo no qual queremos escrever.
- Retorno da função
    - Se o texto for escrito com sucesso um valor inteiro diferente de zero é retornado.
    - Se houver erro na escrita, o valor EOF (em geral, −1) é retornado.

Manipulando strings

- Note que a função **fputs** utiliza uma variável FILE ***fp**, que está associado ao arquivo de onde a string será escrita.
- Para escrever no monitor, basta substituir FILE ***fp** por **stdout**, o qual representa o dispositivo de saída padrão (geralmente a tela do monitor).

```
char str[20] = “Teste de escrita”;
fputs (str,stdout);
```