SENAI CIMATEC

SISTEMA FIEB

TECNOLOGIA E INOVAÇÃO EM PROL DA INDÚSTRIA

SENAI CIMATEC

SISTEMA FIEB

Técnico em Informática

SENAI CIMATEC SISTEMA FIEB
Federação das Indústrias do Estado da Bahia

Programação de Aplicativos– 140h

Profª: Francisleide Almeida

Strings x Caractere

- Para escrever/guardar uma string usa-se aspas dulpas;
- Para guardar valor em uma variável tipo caractere usa-se aspas simples;

- Na variável tipo Caractere, guarda apenas 1 byte;
- Na variável tipo String guarda mais de 1 byte

Strings

- Como em C as strings não são um tipo básico, a única forma de representar caracteres é recorrendo a um vetor de caracteres
- Uma string e um vetor de caracteres, mas nem todo vetor de caracteres é uma string;

Strings

- Na linguagem C strings são vetores de caracteres que possuem um caractere que indica o término de seu conteúdo, o caractere nulo ‘\0’ (contra barra zero).

Declaração de Strings

- Como a string possui o caractere nulo para delimitar o final do seu conteúdo, o tamanho da string deve ser definido com um caractere a mais do que será efetivamente necessário.

Sintaxe:

char identificador-da-string [tamanho+1];

Declaração de Strings

- Ex:

char vetc [6];

- **vetc** é um vetor de caracteres (string) de tamanho 6. Pode receber uma palavra de no máximo 5 letras

Declaração de Strings

- Uma string pode ser inicializada na sua declaração com uma sequência de caracteres entre chaves e separadas por vírgula.

```
char vetc[6]= {'T', 'e', 'x', 't', 'o', '\0'};
```

- *Lembre-se que o compilador só reconhecerá um caractere se este estiver entre aspas simples, logo usar uma atribuição do tipo* {t,e,x,t,o,\0} *ou* {texto\0} ***irá gerar um erro de compilação****.*

- Uma string pode também ser inicializada por uma sequência de caracteres entre aspas duplas. Neste caso, não é necessário o uso de aspas simples e virgulas, o compilador C coloca automaticamente o '\0' no final.

char vetc[6] = “Texto”;

- Assim como vetores e matrizes, na inicialização de uma string o seu tamanho pode ser omitido.

```
char vetc[ ] = “Texto”;
 /* vetor não-dimensionado, o compilador coloca automaticamente o ‘\0’ no final */
```

Leitura de Strings

- *Utilizando a função* ***scanf()***

- A sintaxe para receber uma string por meio da *scanf()* é:

scanf(“%s”, nome_da_string);

Leitura de Strings

- *Utilizando a função* ***scanf()***

- A sintaxe para receber uma string por meio da *scanf()* é:

scanf(“%s”, nome_da_string);

OBS.: não é necessário colocar o operador &, pois o nome da string em si já é um endereço de memória.

Leitura de Strings

- ***Exemplo:*** *C*rie um aplicativo em C que peça ao usuário seu nome, armazene em uma String, peça o sobrenome, armazene em outra string e exiba o nome do usuário de maneira formal (Sobrenome, Nome).

Leitura de Strings

- ***Exemplo:*** Crie um aplicativo em C que peça ao usuário seu nome, armazene em uma String, peça o sobrenome, armazene em outra string e exiba o nome do usuário de maneira formal (Sobrenome, Nome).

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char nome[21], sobrenome[21];

    printf("Primeiro nome: ");
    scanf("%s", nome);

    printf("Ultimo sobrenome: ");
    scanf("%s", sobrenome);

    printf("Ola senhor %s, %s. Bem-vindo ao curso de linguagem C.\n", sobrenome, nome);

    system("pause");

}
```

- O que aconteceria se fosse digitado um nome composto no ‘nome’ ou no ‘sobrenome’?

- A *scanf()* vai simplesmente cortar seu nome composto. Essa função pega tudo **até** encontrar um espaço em branco, caractere *new line* \n, tab ou ENTER.

- ***Funções para Manipulação de Strings***

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
 char nome[] = "fulano";
 char sobrenome[] = "de tal";
 char nomeCompleto[] = nome + sobrenome;
 int i=0;

 while(nomeCompleto[i] != '\0'){
  printf("%c",nomeCompleto[i]);
  nomeCompleto[i++];
 }

 return(0);
 system("pause");
 }
```

Leitura de Strings

- ***Funções para Manipulação de Strings***

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
 char nome[] = "fulano";
 char sobrenome[] = "de tal";
 char nomeCompleto[] = nome + sobrenome;
 int i=0;

 while(nomeCompleto[i] != '\0'){
  printf("%c",nomeCompleto[i]);
  nomeCompleto[i++];
 }

 return(0);
 system("pause");
 }
```

PORQUE O PROGRAMA NÃO COMPILA?

Um erro muito comum no uso de string em C esta sendo cometido na linha 7. char nomeCompleto[] = nome + sobrenome. String não poder ser concatenadas utilizando o operador +. Existe uma diretiva em C que implementa diversas funções de manipulação de valores em string.

Manipulação de Strings

- **Get String (gets):**
- A função **gets()** lê os caracteres do dispositivo padrão de entrada (teclado) para o seu argumento – um vetor do tipo **char** – até que um caractere de nova linha ou o indicador de fim de arquivo seja encontrado. Um caractere **NULL** (**'\0'**) será adicionado ao vetor quando a leitura terminar.
- Sua forma geral é:

    gets (nome_da_string);

Manipulação de Strings

- **Get String (gets)**

```c
#include <stdio.h>

main(){
    char nome[21], sobrenome[21];
    printf("Informe seu primeiro nome: ");
    gets(nome);
    printf("Informe seu ultimo nome: ");
    gets(sobrenome);
    printf("Seja bem vindo %s, %s", sobrenome,nome);

}
```

- Como o primeiro argumento da função **printf()** é uma string também é válido fazer:

printf (string); /* isto simplesmente imprimirá a string. */

Manipulação de Strings

- **Get String (gets)**
  - Não é uma função segura, pois o tamanho da string não é especificado. A função *scanf()* pega tudo até aparecer o primeiro espaço em branco, e pára. Já a *gets()* não, ela pega tudo até aparecer uma *new line* \n, inclusive nada. Ou seja, se você der um ENTER, a *gets()* vai armazenar esse enter na string

Manipulação de string

```c
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    char nome[31], sobrenome[31], nascimento[11];
    int idade;

    printf("Nome: ");
    gets(nome);

    printf("Sobrenome: ");
    gets(sobrenome);

    printf("Idade: ");
    scanf("%d", &idade);

    printf("Data de nascimento: ");
    gets(nascimento);

    printf("\nNome completo: %s %s\n", nome, sobrenome);
    printf("Idade: %d\n", idade);
    printf("Data de nascimento: "); puts(nascimento);

    system("pause");
}
```

Get String (gets) e Scanf

O problema é que a função *gets()* vai pegar o que está armazenado nesse buffer e vai armazenar o que estiver lá na string de data de nascimento!

E como evitar isso? É só apagar esse ENTER que está no buffer, usando o *fflush(stdin)* caso use Windows, ou *__fpurge(stdin)* caso use Linux.

- É possível alterar o funcionamento da ***scanf()***. Por exemplo, se quisermos ler strings que tenham espaço, nós temos que dizer isso dentro da função. Para dizer para a scanf() parar de pegar nossa string somente quando encontrar um caractere de NEW LINE (um enter). Usamos o operador: [^\n]
- Logo, nosso código da ***scanf()*** para ler strings com espaços e armazenar na variável "str" é:

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

- Podemos ainda limitar o tamanho de nossa string, basta colocar um numero inteiro ao lado do %, representando o número de caracteres máximo, o que é uma excelente prática, pois essa função pode ocasionar problemas na memória, caso você estoure os limites da string.
- Por exemplo:

```
scanf ( "%256[^\n]", str);
```

- A função ***gets()*** peca nesse quesito, de tamanho da string, pois podemos digitar mais caracteres do que a string alocou de memória, e "quebraríamos" o programa por conta de um *overflow*.
- Uma solução para isso é usar a função ***fget()***, que é mais segura.

- A string que vai armazenar o que vai ser digitado (no nosso caso é a variável "str");
- O tamanho da string e de onde vai ler (ela pode ler de um arquivo de texto, por exemplo);
- Para ler do teclado, usamos ***stdin***.

fget()

- fget() recebe 3 dados:
  - A string que vai armazenar o que vai ser digitado (no nosso caso é a variável "str");
  - O tamanho da string e de onde vai ler (ela pode ler de um arquivo de texto, por exemplo);
  - Para ler do teclado, usamos ***stdin***.

Outras funções

Incluir a biblioteca **string.h**

- *Strings não podem ser comparadas com o operador de comparação padrão (==), neste caso deve-se usar função* ***strcmp()*** *ou a função* ***stricmp()****.*
  - **strcmp(s1,s2)** – Retorna 0 se s1 e s2 são iguais; menor que 0 se s1<s2; maior que 0 se s1>s2 (comparação alfabética).
  - **stricmp(s1,s2)** – Retorna 0 se s1 e s2 são iguais; menor que 0 se s1<s2; maior que 0 se s1>s2 (comparação alfabética). Essa função considera letras maiúsculas ou minúsculas como símbolos iguais.

Outras funções

Incluir a biblioteca **string.h**

- *Strings não podem ser atribuídas com o operador de atribuição (=), para uma atribuição usa-se a função* ***strcpy()****.*

  strcpy(s1,s2) – Copia s2 em s1.

- OBS.: A string-destino deve ser grande o suficiente para armazenar a string origem e seu caractere **NULL** de terminação que também é copiado.

Outras funções

Incluir a biblioteca **string.h**

- *Strings não podem ser concatenadas com o operador (+), para tal usa-se a função* ***strcat()***

  – **strcat(s1,s2)** – Concatena s2 ao final de s1.

  – **strlen(s)** – Retorna o número de caracteres em s (sem contar o caracter nulo (/0)).

Exercícios

- Crie um algoritmo que inverta uma sequência de caracteres inseridas pelo usuário;
- Escreva um algoritmo em C para concatenar 2 dadas strings, devolvendo o resultado na primeira delas.