UNIFG

LAUREATE INTERNATIONAL UNIVERSITIES®

TÉCNICAS DE PROGRAMAÇÃO

Aula 04 – Manipulando Strings

Objetivos de Aprendizagem

1. Descrever as funções existentes para manipulação de Strings.
2. Criar aplicações com a utilização de funções que manipulam Strings.

Strings

- Uma **string** em C é um vetor de caracteres que termina com um caractere **NULL '\ 0'**.
- Uma declaração de string pode ser feita de várias maneiras, cada uma com suas próprias considerações.
- Por exemplo:

```
char str_nome[str_tam + 1] = "string";
```

Strings

- A declaração anterior cria uma string denominada **`str_nome`** de caracteres **`str_tam + 1`** e a inicializa com o valor "string".
- Quando você fornece uma “cadeia de caracteres” para inicializar a string, o compilador adiciona automaticamente um caractere **NULL '\ 0'** ao vetor de caracteres.

Strings

- As instruções abaixo criam sequências de caracteres que incluem o caractere **NULL**.
- Se a declaração não incluir um tamanho do vetor de caracteres, será calculada com base no comprimento da string na inicialização mais um para '\0':

```
char str1[6] = "olá";
char str2[ ] = "mundo";  /* tamanho 6 */
```

Strings

- Uma string também pode ser declarada como um conjunto de caracteres:

```
char str3[6] = {'h', 'e', 'l', 'l', 'o', '\0'};
char str4[ ] = {'h', 'e', 'l', 'l', 'o', '\0'};
/* tamanho 6 */
```

- Com essa abordagem, o caractere **NULL** deve ser adicionado explicitamente.
  - Observe que os caracteres estão entre aspas simples.

Strings

- Como em qualquer **vetor**, o nome de uma **string** atua como um ponteiro.
- Uma string é um texto entre aspas duplas.

Strings

- Um caractere, como 'b', é indicado por aspas simples e não pode ser tratado como uma sequência.
- Uma declaração de ponteiro de string, como **`char * str = "stuff";`** é considerado uma constante e não pode ser alterado de seu valor inicial.

Manipulação de Strings

- Para operar com segurança e conveniência com strings, você pode usar as funções de biblioteca padrão para strings.
  - Não se esqueça de incluir **`<string.h>`**.

Manipulação de Strings

- `strlen ()` - obtém o comprimento de uma string
- `strcat ()` - mescla duas strings
- `strcpy ()` - copia uma string para outra
- `strlwr ()` - converte string para minúsculas
- `strupr ()` - converte string para maiúsculas
- `strrev ()` - string reversa
- `strcmp ()` - compare duas strings

String Input

- Os programas geralmente são interativos, solicitando a entrada do usuário.
- Para recuperar uma linha de texto ou outra string do usuário, C fornece as funções **`scanf ()`**, **`gets ()`** e **`fgets ()`**.
- Você pode usar **`scanf ()`** para ler a entrada de acordo com os especificadores de formato.

String Input

```
char nome[25];
int idade;
printf(“Digite seu nome e idade: \n");
scanf("%s %d", nome, &age);
```

- Quando **`scanf ()`** é usado para ler uma **string**, não é necessário **`&`** acessar o endereço da variável porque o nome de um veotr atua como um ponteiro.

String Input

- **`scanf ()`** pára de ler a entrada quando atinge um espaço. Para ler uma **string** com espaços, use a função **`gets ()`**.
- Ele lê a entrada até que uma nova linha final seja alcançada (a tecla Enter é pressionada).
- Por exemplo:

```
char nome_completo[50];
printf("Digite seu nome completo: ");
gets(nome_completo);
```

String Input

- Uma alternativa mais segura para **`gets ()`** é **`fgets ()`**, que lê até um número especificado de caracteres.
- Essa abordagem ajuda a evitar um estouro de buffer, o que acontece quando a string não é grande o suficiente para o texto digitado.

String Input

```
char nome_completo[50];
printf("Digite seu nome completo: ");
fgets(nome_completo, 50, stdin);
```

- Os argumentos **`fgets ()`** são o nome da *string*, o número de caracteres a serem lidos e um ponteiro de onde você deseja ler a *string*.
  - **`stdin`** significa ler da entrada padrão, que é o teclado.

String Input

- Outra diferença entre **`get`** e **`fgets`** é que o caractere de nova linha é armazenado por **`fgets`**.
- **`fgets ()`** lê apenas **`n - 1`** caracteres de **`stdin`** porque deve haver espaço para **`'\0'`**.

String Output

- A saída da *string* é tratada com as funções **`fputs ()`**, **`putf ()`** e **`printf ()`**.
- O **`fputs ()`** requer o nome da sequência e um ponteiro para onde você deseja imprimir a sequência.
- Para imprimir na tela, use **`stdout`**, que se refere à saída padrão.

Exemplo (fputs)

```
#include <stdio.h>
int main()
{
  char cidade[40];
  printf(“Digite o nome da sua cidade favorita: ");
  gets(cidade);
  // Obs: por segurança, use
  // fgets(cidade, 40, stdin);
  fputs(cidade, stdout);
  printf(" é uma cidade divertida.");

  return 0;
}
```

Exemplo (puts)

```
#include <stdio.h>
int main()
{
  char cidade[40];
  printf("Digite o nome da sua cidade favorita: ");
  gets(cidade);
  // Obs: por segurança, use
  // fgets(cidade, 40, stdin);

  puts(cidade);

  return 0;
}
```

As funções sprintf e sscanf

- Uma sequência formatada pode ser criada com a função **`sprintf ()`**.
- Isso é útil para criar uma sequência a partir de outros tipos de dados.

Exemplo

```
#include <stdio.h>
int main()
{
  char info[100];
  char dep[ ] = “RH";
  int emp = 75;
  sprintf(info,“O dep %s tem %d empregados.",dep,emp);
  printf("%s\n", info);

  return 0;
}
```

As funções sprintf e sscanf

- Outra função útil é **`sscanf ()`** para varrer uma **string** em busca de valores.
- A função lê valores de uma sequência e os armazena nos endereços variáveis correspondentes.

Exemplo

```
#include <stdio.h>
int main()
{
  char info[ ] = “Recife PE 50260-000";
  char cidade[50];
  char estado[50];
  int populacao;
  sscanf(info, "%s %s %d", cidade, estado, &populacao);
  printf("%d pessoas vivem
em %s, %s.", populacao, cidade, estado);

  return 0;
}
```

Exercício

- Faça um programa que solicita o usuário digitar o nome e endereço completo (armazenando em duas strings).
- Em seguida o programa imprime na tela o que foi digitado.

A biblioteca string.h

- A instrução `#include <string.h>` na parte superior do seu programa fornece acesso ao seguinte:
  - `strlen (str)` retorna o comprimento da string armazenada em `str`, sem incluir o caractere `NULL`.
  - `strcat (str1, str2)` anexa (concatena) `str2` ao final de `str1` e retorna um ponteiro para `str1`.
  - `strcpy (str1, str2)` copia `str2` para `str1`. Esta função é útil para atribuir um novo valor a uma string.

Exemplo

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

int main()
{
   char s1[ ] = “A raposa cinza";
   char s2[ ] = " pulou.";
```

Exemplo

```
    strcat(s1, s2);
    printf("%s\n", s1);
    printf(“Tamanho de s1 é %d\n", strlen(s1));
    strcpy(s1, s2);
    printf("s1 agora é %s \n", s1);

    return 0;
}
```

A biblioteca string.h

- As funções adicionais string.h incluem:
  - `strncat (str1, str2, n)` anexa (concatena) os primeiros n caracteres de `str2` no final de `str1` e retorna um ponteiro para `str1`.
  - `strncpy (str1, str2, n)` copia os primeiros **n** caracteres de `str2` para `str1`.
  - `strcmp (str1, str2)` retorna 0 quando `str1` é igual a `str2`, menor que 0 quando `str1 <str2` e maior que 0 quando `str1> str2`.

A biblioteca string.h

- `strncmp (str1, str2, n)` retorna 0 quando os primeiros n caracteres de `str1` são iguais aos primeiros n caracteres de `str2`, menor que 0 quando `str1 <str2` e maior que 0 quando `str1> str2`.
- `strchr (str1, c)` retorna um ponteiro para a primeira ocorrência de char c em `str1`, ou NULL se o caractere não for encontrado.

A biblioteca string.h

- `strrchr (str1, c)` Pesquisa `str1` no sentido inverso e retorna um ponteiro para a posição de `char c` em `str1`, ou `NULL` se o caractere não for encontrado.
- `strstr (str1, str2)` Retorna um ponteiro para a primeira ocorrência de `str2` em `str1`, ou `NULL` se `str2` não for encontrado.

Exercício

- Faca um programa que solicita o usuário digitar uma mensagem (string).
- Em seguida o programa converte todos os caracteres da string para maiúsculo e depois imprime os resultados.

Formatação da Saída Padrão

- A função **`scanf ()`** é usada para atribuir entrada a variáveis.
- Uma chamada para essa função verifica a entrada de acordo com os especificadores de formato que convertem a entrada conforme necessário.

Formatação da Saída Padrão

- Se a entrada não puder ser convertida, a atribuição não será feita.
- A instrução **`scanf ()`** aguarda entrada e, em seguida, faz atribuições:

```
int x;
float num;
char texto[20];
scanf("%d %f %s", &x, &num, texto);
```

Formatação da Saída Padrão

- Os especificadores de formato começam com um sinal de porcentagem % e são usados para atribuir valores aos argumentos correspondentes após a cadeia de controle.
  - Espaços em branco, guias e novas linhas são ignorados.

Formatação da Saída Padrão

- O caractere de conversão converte o argumento, se necessário, no tipo indicado:
  - d decimal
  - c caractere
  - s string
  - f ponto flutuante
  - x hexadecimal

Formatação da Saída Padrão

- Um especificador de formato pode incluir várias opções junto com um caractere de conversão:
  - Caractere de conversão `%` **`[*] [max_field]`**
    - O opcional **`*`** pulará o campo de entrada.
  - O opcional **`max_width`** fornece o número máximo de caracteres a serem atribuídos a um campo de entrada.

Formatação da Saída Padrão

```c
int x, y;
char texto[20];

scanf("%2d %d %*f %5s", &x, &y, texto);
/* Entrada: 1234  5.7  elefante */
printf("%d  %d  %s", x, y, texto);
/* Saída: 12  34  elefa */
```

Formatação da Saída Padrão

- A função **`printf`** foi introduzida no seu primeiro programa Hello World.
- Uma chamada para esta função requer uma sequência de formato que pode incluir:
  - Sequências de escape para gerar caracteres especiais
  - Especificadores de formato que são substituídos por valores

Formatação da Saída Padrão

```
printf("A árvore tem %d maçãs.\nv, 22);
/* A árvore tem 22 maçãs. */

printf("\"Olá mundo!\"\n");
/* "Olá mundo" */
```

Formatação da Saída Padrão

- As sequências de escape começam com uma barra invertida \:
- **\n** nova linha
- **\t** guia horizontal
- **** barra invertida
- **\b** backspace
- **\'** citação única
- **\"** citação dupla

Formatação da Saída Padrão

- Os especificadores de formato começam com um sinal de porcentagem % e são substituídos pelos argumentos correspondentes após a sequência de caracteres do formato.
- Um especificador de formato pode incluir várias opções junto com um caractere de conversão.
  - %[-][width].[precision]conversion character

Formatação da Saída Padrão

- O opcional – especifica o alinhamento à esquerda dos dados na sequência.
- O opcional **`width`** fornece o número mínimo de caracteres para os dados.
- O período `.` separa a largura da precisão.

Formatação da Saída Padrão

- A precisão opcional fornece o número de casas decimais para dados numéricos.
  - Se **s** for usado como caractere de conversão, a precisão determinará o número de caracteres a serem impressos.

Formatação da Saída Padrão

- O caractere de conversão converte o argumento, se necessário, no tipo indicado:
  - d decimal
  - c caractere
  - s string
  - f ponto flutuante
  - e notação científica
  - x hexadecimal

Formatação da Saída Padrão

```
printf(“Cor: %s, Número: %d, float: %5.2f \n", “vermelh
o", 42, 3.14159);
/* Cor: vermelho, Número: 42, float:  3.14  */

printf("Pi = %3.2f", 3.14159);
/* Pi = 3.14 */
```

Formatação da Saída Padrão

```
printf("Pi = %8.5f", 3.14159);
/* Pi =   3.14159 */

printf("Pi = %-8.5f", 3.14159);
/* Pi = 3.14159 */

printf("Há %d %s na árvore.", 22, "maçãs");
/* Há 22 maçãs na árvore. */
```

Convertendo uma String em um Número

- Converter uma sequência de caracteres numéricos em um valor numérico é uma tarefa comum na programação C e é frequentemente usada para evitar um erro em tempo de execução.
  - A leitura de uma string é menos propensa a erros do que a expectativa de um valor numérico, apenas para que o usuário digite acidentalmente um "o" em vez de um "0" (zero).

Convertendo uma String em um Número

- A biblioteca stdio.h contém as seguintes funções para converter uma sequência em um número:
- int atoi (str) Representa ASCII para inteiro. Converte str no valor int equivalente. 0 será retornado se o primeiro caractere não for um número ou nenhum número for encontrado.

Convertendo uma String em um Número

- double atof (str) Representa a flutuação do ASCII. Converte str no valor duplo equivalente. 0.0 será retornado se o primeiro caractere não for um número ou nenhum número for encontrado.

Convertendo uma String em um Número

- long int atol (str) Significa ASCII para long int. Converte str no valor inteiro longo equivalente. 0 será retornado se o primeiro caractere não for um número ou nenhum número for encontrado.

Exemplo

```
#include <stdio.h>
int main()
{
  char input[10];
  int num;

  printf("Digite um número: ");
  gets(input);
  num = atoi(input);

  return 0;
}
```

Matrizes de Strings

- Uma matriz bidimensional pode ser usada para armazenar strings.
- Considere a seguinte declaração que declara uma matriz com 3 elementos, cada um contendo 15 caracteres:

```
char trip[3][15] = {
  “mala",
  "passporte",
  “bilhete"
};
```

Matrizes de Strings

- Embora os comprimentos da string variem, é necessário declarar um tamanho grande o suficiente para conter a string mais longa.
  - Além disso, pode ser muito complicado acessar os elementos.

Matrizes de Strings

- A referência ao viagem [0] para "mala" é suscetível a erros.
  - Em vez disso, você deve pensar no elemento em [0] [0] como 's', o elemento em [2] [3] como 'k' e assim por diante.

Matrizes de Strings

```
char *viagem[ ] = {
  “mala",
  "passporte",
  “bilhete"
};

printf(“Por favor, traga os seguintes itens:\n");
for (int i = 0; i < 3; i++) {
  printf("%s\n", viagem[ i ]);
}
```

Matrizes de Strings

- Como cada elemento pode variar em comprimento, a matriz de ponteiros de sequência de caracteres possui uma estrutura mais irregular, em oposição a uma estrutura de grade bidimensional.
  - Com essa abordagem, não há limite para o comprimento da string.
  - Os itens podem ser referidos por um ponteiro para o primeiro caractere de cada sequência.

Matrizes de Strings

- Lembre-se de que uma declaração como char **`* items [3];`** apenas reserva espaço para três ponteiros; as strings reais estão sendo referenciadas por esses ponteiros.

Exercício

1. Faça um programa que solicita o usuário digitar o nome e sobrenome
2. Em seguida o programa solicita o usuário digitar rua, numero, bairro, cidade (capturando todos os dados como string).
3. Finalmente o programa concatena o nome e sobrenome e mostra na tela.
4. Depois o programa concatena os dados do endereço e imprime o endereço de uma só vez.