Cadeias de caracteres

Aula 13

Fábio Henrique Viduani Martinez

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação I, 2012

Conteúdo da aula

Introdução

- vetores de caracteres são diferentes de cadeias de caracteres
- uma cadeia de caracteres tem um caractere nulo `\0` adicionado ao seu final
- essa característica evita, em muitos casos, que tenhamos de manter uma variável que contenha o comprimento do vetor para saber a quantidade de caracteres contidos nele
- cadeias de caracteres têm tratamento especial na linguagem C
- diversas funções que manipulam cadeias de caracteres estão disponíveis na biblioteca `string` da linguagem C

Introdução

- vetores de caracteres são diferentes de cadeias de caracteres
- uma cadeia de caracteres tem um caractere nulo `\0` adicionado ao seu final
- essa característica evita, em muitos casos, que tenhamos de manter uma variável que contenha o comprimento do vetor para saber a quantidade de caracteres contidos nele
- cadeias de caracteres têm tratamento especial na linguagem C
- diversas funções que manipulam cadeias de caracteres estão disponíveis na biblioteca `string` da linguagem C

Introdução

- vetores de caracteres são diferentes de cadeias de caracteres
- uma cadeia de caracteres tem um caractere nulo `\0` adicionado ao seu final
- essa característica evita, em muitos casos, que tenhamos de manter uma variável que contenha o comprimento do vetor para saber a quantidade de caracteres contidos nele
- cadeias de caracteres têm tratamento especial na linguagem C
- diversas funções que manipulam cadeias de caracteres estão disponíveis na biblioteca `string` da linguagem C

Introdução

- vetores de caracteres são diferentes de cadeias de caracteres
- uma cadeia de caracteres tem um caractere nulo `\0` adicionado ao seu final
- essa característica evita, em muitos casos, que tenhamos de manter uma variável que contenha o comprimento do vetor para saber a quantidade de caracteres contidos nele
- cadeias de caracteres têm tratamento especial na linguagem C
- diversas funções que manipulam cadeias de caracteres estão disponíveis na biblioteca `string` da linguagem C

Introdução

- vetores de caracteres são diferentes de cadeias de caracteres
- uma cadeia de caracteres tem um caractere nulo `\0` adicionado ao seu final
- essa característica evita, em muitos casos, que tenhamos de manter uma variável que contenha o comprimento do vetor para saber a quantidade de caracteres contidos nele
- cadeias de caracteres têm tratamento especial na linguagem C
- diversas funções que manipulam cadeias de caracteres estão disponíveis na biblioteca **`string`** da linguagem C

Literais

- na linguagem C, uma constante do tipo cadeia de caractere é chamada de **literal**

```
printf("Programar é bacana!\n");
```

- aspas duplas são usadas para delimitar uma constante do tipo cadeia de caracteres, que pode conter qualquer combinação de letras, números ou caracteres especiais que não sejam as aspas dupla
- é possível inserir as aspas duplas no interior de uma constante cadeia de caracteres, inserindo a sequência `\"` nessa cadeia

Literais

- na linguagem C, uma constante do tipo cadeia de caractere é chamada de **literal**

```
printf("Programar é bacana!\n");
```

- aspas duplas são usadas para delimitar uma constante do tipo cadeia de caracteres, que pode conter qualquer combinação de letras, números ou caracteres especiais que não sejam as aspas dupla
- é possível inserir as aspas duplas no interior de uma constante cadeia de caracteres, inserindo a sequência `\"` nessa cadeia

Literais

- na linguagem C, uma constante do tipo cadeia de caractere é chamada de **literal**

```
printf("Programar é bacana!\n");
```

- aspas duplas são usadas para delimitar uma constante do tipo cadeia de caracteres, que pode conter qualquer combinação de letras, números ou caracteres especiais que não sejam as aspas dupla
- é possível inserir as aspas duplas no interior de uma constante cadeia de caracteres, inserindo a sequência `\"` nessa cadeia

Literais

- uma variável do tipo `char` pode conter apenas um único caractere; para atribuir um caractere a uma variável, o caractere deve ser envolvido por aspas simples;

```
char sinal;
sinal = '+';
```

- uma distinção entre as aspas simples e as aspas duplas, sendo que no primeiro caso elas servem para definir constantes do tipo caractere e no segundo para definir constantes do tipo cadeia de caracteres
- usamos literais especialmente quando chamamos as funções `printf` e `scanf` em um programa, ou seja, quando descrevemos cadeias de caracteres de formatação
- a linguagem C trata as literais como cadeias de caracteres

Literais

- uma variável do tipo `char` pode conter apenas um único caractere; para atribuir um caractere a uma variável, o caractere deve ser envolvido por aspas simples;

```
char sinal;
sinal = '+';
```

- uma distinção entre as aspas simples e as aspas duplas, sendo que no primeiro caso elas servem para definir constantes do tipo caractere e no segundo para definir constantes do tipo cadeia de caracteres
- usamos literais especialmente quando chamamos as funções `printf` e `scanf` em um programa, ou seja, quando descrevemos cadeias de caracteres de formatação
- a linguagem C trata as literais como cadeias de caracteres

Literais

- uma variável do tipo `char` pode conter apenas um único caractere; para atribuir um caractere a uma variável, o caractere deve ser envolvido por aspas simples;

```
char sinal;
sinal = '+';
```

- uma distinção entre as aspas simples e as aspas duplas, sendo que no primeiro caso elas servem para definir constantes do tipo caractere e no segundo para definir constantes do tipo cadeia de caracteres
- usamos literais especialmente quando chamamos as funções `printf` e `scanf` em um programa, ou seja, quando descrevemos cadeias de caracteres de formatação
- a linguagem C trata as literais como cadeias de caracteres

Literais

- uma variável do tipo `char` pode conter apenas um único caractere; para atribuir um caractere a uma variável, o caractere deve ser envolvido por aspas simples;

```
char sinal;
sinal = '+';
```

- uma distinção entre as aspas simples e as aspas duplas, sendo que no primeiro caso elas servem para definir constantes do tipo caractere e no segundo para definir constantes do tipo cadeia de caracteres
- usamos literais especialmente quando chamamos as funções `printf` e `scanf` em um programa, ou seja, quando descrevemos cadeias de caracteres de formatação
- a linguagem C trata as literais como cadeias de caracteres

Literais

- quando o compilador da linguagem C encontra uma literal com n caracteres em um programa, ele reserva $n + 1$ compartimentos de memória para armazenar a cadeia de caracteres correspondente
- essa área na memória conterá os caracteres da cadeia mais um caractere extra, o caractere nulo, que registra o final da cadeia
- o caractere nulo é um byte cujos bits são todos 0 (zeros) e é representado pela sequência `'\0'`
- existe uma diferença entre o caractere nulo e o caractere zero: o primeiro é um caractere não-imprimível, tem valor decimal 0 e constante `'\0'`; o segundo é um caractere imprimível, tem valor 48, símbolo gráfico `0` e constante `'0'`

Literais

- quando o compilador da linguagem C encontra uma literal com n caracteres em um programa, ele reserva $n + 1$ compartimentos de memória para armazenar a cadeia de caracteres correspondente
- essa área na memória conterá os caracteres da cadeia mais um caractere extra, o caractere nulo, que registra o final da cadeia
- o caractere nulo é um byte cujos bits são todos 0 (zeros) e é representado pela sequência `'\0'`
- existe uma diferença entre o caractere nulo e o caractere zero: o primeiro é um caractere não-imprimível, tem valor decimal 0 e constante `'\0'`; o segundo é um caractere imprimível, tem valor 48, símbolo gráfico `0` e constante `'0'`

Literais

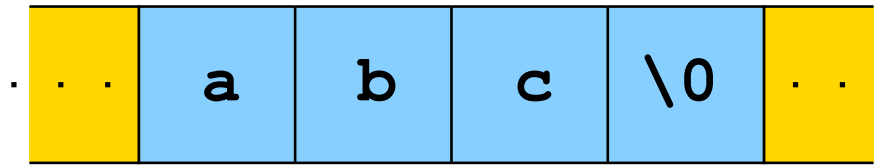
- quando o compilador da linguagem C encontra uma literal com n caracteres em um programa, ele reserva $n + 1$ compartimentos de memória para armazenar a cadeia de caracteres correspondente
- essa área na memória conterá os caracteres da cadeia mais um caractere extra, o caractere nulo, que registra o final da cadeia
- o caractere nulo é um byte cujos bits são todos 0 (zeros) e é representado pela sequência `'\0'`
- existe uma diferença entre o caractere nulo e o caractere zero: o primeiro é um caractere não-imprimível, tem valor decimal 0 e constante `'\0'`; o segundo é um caractere imprimível, tem valor 48, símbolo gráfico `0` e constante `'0'`

Literais

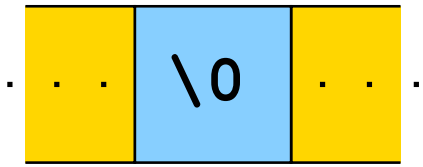
- quando o compilador da linguagem C encontra uma literal com n caracteres em um programa, ele reserva $n + 1$ compartimentos de memória para armazenar a cadeia de caracteres correspondente
- essa área na memória conterá os caracteres da cadeia mais um caractere extra, o caractere nulo, que registra o final da cadeia
- o caractere nulo é um byte cujos bits são todos 0 (zeros) e é representado pela sequência `'\0'`
- existe uma diferença entre o caractere nulo e o caractere zero: o primeiro é um caractere não-imprimível, tem valor decimal 0 e constante `'\0'`; o segundo é um caractere imprimível, tem valor 48, símbolo gráfico `0` e constante `'0'`

Literais

- a literal `"abc"` é armazenada como um vetor de quatro caracteres na memória



- uma literal também pode ser vazia: `""`



Vetores de caracteres

- se queremos trabalhar com variáveis que comportam mais que um único caractere, temos de trabalhar com vetores de caracteres

```
char palavra[MAX+1], frase[MAX+1];
```

- para ler um conteúdo na variável `palavra`, fazemos

```
printf("Informe a palavra: ");
i = 0;
do {
   scanf("%c", &palavra[i]);
   i++;
} while (palavra[i-1] != '\n');
m = i-1;
```

- para imprimir o conteúdo dessa variável, fazemos

```
printf("Palavra: ");
for (i = 0; i < m; i++)
   printf("%c", palavra[i]);
printf("\n");
```

Vetores de caracteres

- se queremos trabalhar com variáveis que comportam mais que um único caractere, temos de trabalhar com vetores de caracteres

```
char palavra[MAX+1], frase[MAX+1];
```

- para ler um conteúdo na variável **palavra**, fazemos

```
printf("Informe a palavra: ");
i = 0;
do {
   scanf("%c", &palavra[i]);
   i++;
} while (palavra[i-1] != '\n');
m = i-1;
```

- para imprimir o conteúdo dessa variável, fazemos

```
printf("Palavra: ");
for (i = 0; i < m; i++)
   printf("%c", palavra[i]);
printf("\n");
```

Vetores de caracteres

- se queremos trabalhar com variáveis que comportam mais que um único caractere, temos de trabalhar com vetores de caracteres

```
char palavra[MAX+1], frase[MAX+1];
```

- para ler um conteúdo na variável `palavra`, fazemos

```
printf("Informe a palavra: ");
i = 0;
do {
   scanf("%c", &palavra[i]);
   i++;
} while (palavra[i-1] != '\n');
m = i-1;
```

- para imprimir o conteúdo dessa variável, fazemos

```
printf("Palavra: ");
for (i = 0; i < m; i++)
   printf("%c", palavra[i]);
printf("\n");
```

Vetores de caracteres

- sempre necessitamos de uma variável adicional para controlar o comprimento de um vetor de caracteres quando da sua leitura
- a variável `m` faz esse papel no primeiro trecho de programa acima
- tanto para leitura como para escrita de um vetor de caracteres, precisamos de uma estrutura de repetição para processar os caracteres um a um
- com cadeias de caracteres, evitamos esta sobrecarga de trabalho para o(a) programador(a)

Vetores de caracteres

- sempre necessitamos de uma variável adicional para controlar o comprimento de um vetor de caracteres quando da sua leitura
- a variável `m` faz esse papel no primeiro trecho de programa acima
- tanto para leitura como para escrita de um vetor de caracteres, precisamos de uma estrutura de repetição para processar os caracteres um a um
- com cadeias de caracteres, evitamos esta sobrecarga de trabalho para o(a) programador(a)

Vetores de caracteres

- sempre necessitamos de uma variável adicional para controlar o comprimento de um vetor de caracteres quando da sua leitura
- a variável `m` faz esse papel no primeiro trecho de programa acima
- tanto para leitura como para escrita de um vetor de caracteres, precisamos de uma estrutura de repetição para processar os caracteres um a um
- com cadeias de caracteres, evitamos esta sobrecarga de trabalho para o(a) programador(a)

Vetores de caracteres

- sempre necessitamos de uma variável adicional para controlar o comprimento de um vetor de caracteres quando da sua leitura
- a variável `m` faz esse papel no primeiro trecho de programa acima
- tanto para leitura como para escrita de um vetor de caracteres, precisamos de uma estrutura de repetição para processar os caracteres um a um
- com cadeias de caracteres, evitamos esta sobrecarga de trabalho para o(a) programador(a)

Cadeias de caracteres

- algumas linguagens de programação de alto nível oferecem ao(à) programador(a) um tipo de dados especial para que variáveis do tipo cadeia de caracteres possam ser declaradas
- na linguagem C não há um tipo de dados como esse e, assim, qualquer vetor de caracteres pode ser usado para armazenar uma cadeia de caracteres
- a diferença é que uma cadeia de caracteres é sempre terminada por um caractere nulo
- uma vantagem dessa estratégia é que não há necessidade de se manter o comprimento da cadeia de caracteres associado a ela
- por outro lado, esse mesmo comprimento, se necessário, só pode ser encontrado através de uma varredura no vetor

Cadeias de caracteres

- algumas linguagens de programação de alto nível oferecem ao(à) programador(a) um tipo de dados especial para que variáveis do tipo cadeia de caracteres possam ser declaradas
- na linguagem C não há um tipo de dados como esse e, assim, qualquer vetor de caracteres pode ser usado para armazenar uma cadeia de caracteres
- a diferença é que uma cadeia de caracteres é sempre terminada por um caractere nulo
- uma vantagem dessa estratégia é que não há necessidade de se manter o comprimento da cadeia de caracteres associado a ela
- por outro lado, esse mesmo comprimento, se necessário, só pode ser encontrado através de uma varredura no vetor

Cadeias de caracteres

- algumas linguagens de programação de alto nível oferecem ao(à) programador(a) um tipo de dados especial para que variáveis do tipo cadeia de caracteres possam ser declaradas
- na linguagem C não há um tipo de dados como esse e, assim, qualquer vetor de caracteres pode ser usado para armazenar uma cadeia de caracteres
- a diferença é que uma cadeia de caracteres é sempre terminada por um caractere nulo
- uma vantagem dessa estratégia é que não há necessidade de se manter o comprimento da cadeia de caracteres associado a ela
- por outro lado, esse mesmo comprimento, se necessário, só pode ser encontrado através de uma varredura no vetor

Cadeias de caracteres

- algumas linguagens de programação de alto nível oferecem ao(à) programador(a) um tipo de dados especial para que variáveis do tipo cadeia de caracteres possam ser declaradas
- na linguagem C não há um tipo de dados como esse e, assim, qualquer vetor de caracteres pode ser usado para armazenar uma cadeia de caracteres
- a diferença é que uma cadeia de caracteres é sempre terminada por um caractere nulo
- uma vantagem dessa estratégia é que não há necessidade de se manter o comprimento da cadeia de caracteres associado a ela
- por outro lado, esse mesmo comprimento, se necessário, só pode ser encontrado através de uma varredura no vetor

Cadeias de caracteres

- algumas linguagens de programação de alto nível oferecem ao(à) programador(a) um tipo de dados especial para que variáveis do tipo cadeia de caracteres possam ser declaradas
- na linguagem C não há um tipo de dados como esse e, assim, qualquer vetor de caracteres pode ser usado para armazenar uma cadeia de caracteres
- a diferença é que uma cadeia de caracteres é sempre terminada por um caractere nulo
- uma vantagem dessa estratégia é que não há necessidade de se manter o comprimento da cadeia de caracteres associado a ela
- por outro lado, esse mesmo comprimento, se necessário, só pode ser encontrado através de uma varredura no vetor

Cadeias de caracteres

- se necessitamos de uma variável capaz de armazenar uma cadeia de caracteres de até 50 caracteres, então a cadeia de caracteres têm de ser declarada com 51 compartimentos para armazenar valores do tipo `char`, já que a cadeia de caracteres necessitará de um caractere nulo no final

```
#define MAX 50
char cadeia[MAX+1];
```

- na declaração de uma variável que pode armazenar uma cadeia de caracteres, sempre devemos reservar um compartimento a mais para que o caractere nulo possa ser armazenado

Cadeias de caracteres

- se necessitamos de uma variável capaz de armazenar uma cadeia de caracteres de até 50 caracteres, então a cadeia de caracteres têm de ser declarada com 51 compartimentos para armazenar valores do tipo `char`, já que a cadeia de caracteres necessitará de um caractere nulo no final

```
#define MAX 50
char cadeia[MAX+1];
```

- na declaração de uma variável que pode armazenar uma cadeia de caracteres, sempre devemos reservar um compartimento a mais para que o caractere nulo possa ser armazenado

Cadeias de caracteres

- a declaração de um vetor de caracteres com dimensão `MAX+1` não quer dizer que ele sempre conterá uma cadeia de caracteres com `MAX` caracteres
- o comprimento de uma cadeia de caracteres depende da posição do caractere nulo na cadeia, não do comprimento do vetor onde a cadeia está armazenada
- como em qualquer vetor, uma cadeia de caracteres também pode ser declarada e inicializada simultaneamente. Por exemplo,

```
char cidade[13] = "Campo Grande";
```

```
char cidade[13] = {'C','a','m','p','o',' ','G','r','a','n','d','e','\0'};
```

Cadeias de caracteres

- a declaração de um vetor de caracteres com dimensão `MAX+1` não quer dizer que ele sempre conterá uma cadeia de caracteres com `MAX` caracteres
- o comprimento de uma cadeia de caracteres depende da posição do caractere nulo na cadeia, não do comprimento do vetor onde a cadeia está armazenada
- como em qualquer vetor, uma cadeia de caracteres também pode ser declarada e inicializada simultaneamente. Por exemplo,

```
char cidade[13] = "Campo Grande";
```

```
char cidade[13] = {'C','a','m','p','o',' ','G','r','a','n','d','e','\0'};
```

Cadeias de caracteres

- a declaração de um vetor de caracteres com dimensão `MAX+1` não quer dizer que ele sempre conterá uma cadeia de caracteres com `MAX` caracteres
- o comprimento de uma cadeia de caracteres depende da posição do caractere nulo na cadeia, não do comprimento do vetor onde a cadeia está armazenada
- como em qualquer vetor, uma cadeia de caracteres também pode ser declarada e inicializada simultaneamente. Por exemplo,

```
char cidade[13] = "Campo Grande";
```

```
char cidade[13] = {'C','a','m','p','o',' ','G','r','a','n','d','e','\0'};
```

Cadeias de caracteres

- a declaração de um vetor de caracteres com dimensão `MAX+1` não quer dizer que ele sempre conterá uma cadeia de caracteres com `MAX` caracteres
- o comprimento de uma cadeia de caracteres depende da posição do caractere nulo na cadeia, não do comprimento do vetor onde a cadeia está armazenada
- como em qualquer vetor, uma cadeia de caracteres também pode ser declarada e inicializada simultaneamente. Por exemplo,

```
char cidade[13] = "Campo Grande";
```

```
char cidade[13] = {'C','a','m','p','o',' ','G','r','a','n','d','e','\0'};
```

Cadeias de caracteres

- se um inicializador tem menor comprimento que o comprimento do vetor, o compilador preencherá os caracteres restantes do vetor com o caractere nulo
- **Cuidado!** se o inicializador tem comprimento maior que a capacidade de armazenamento do vetor associado, os caracteres iniciais do inicializador serão armazenados no vetor, sem que o último deles seja o caractere nulo, impedindo assim que essa variável seja usada como uma legítima cadeia de caracteres

Cadeias de caracteres

- se um inicializador tem menor comprimento que o comprimento do vetor, o compilador preencherá os caracteres restantes do vetor com o caractere nulo
- **Cuidado!** se o inicializador tem comprimento maior que a capacidade de armazenamento do vetor associado, os caracteres iniciais do inicializador serão armazenados no vetor, sem que o último deles seja o caractere nulo, impedindo assim que essa variável seja usada como uma legítima cadeia de caracteres

Cadeias de caracteres

```
#include <stdio.h>

int main(void)
{
   char palavra[10] = "Ola!";
   int n;

   n = 0;
   while (palavra[n] != '\0')
      n++;
   printf("O comprimento da palavra é %d\n", n);

   return 0;
}
```

Cadeias de caracteres

- podemos ler e escrever cadeias de caracteres de uma maneira mais simples na linguagem C
- se `palavra` é um vetor de caracteres terminado com o caractere nulo, podemos usar o especificador de conversão `%s` para mostrar o conteúdo completo da cadeia de caracteres `palavra` na saída padrão

```
printf("%s\n", palavra);
```

- a função `scanf` pode ser usada com o especificador de conversão `%s` para ler uma cadeia de caracteres até que a leitura de um branco seja realizada

```
scanf("%s", palavra);
```

- na leitura de cadeias de caracteres, o símbolo `&` **não** é adicionado como prefixo do identificador da variável

Cadeias de caracteres

- podemos ler e escrever cadeias de caracteres de uma maneira mais simples na linguagem C
- se `palavra` é um vetor de caracteres terminado com o caractere nulo, podemos usar o especificador de conversão `%s` para mostrar o conteúdo completo da cadeia de caracteres `palavra` na saída padrão

```
printf("%s\n", palavra);
```

- a função `scanf` pode ser usada com o especificador de conversão `%s` para ler uma cadeia de caracteres até que a leitura de um branco seja realizada

```
scanf("%s", palavra);
```

- na leitura de cadeias de caracteres, o símbolo `&` **não** é adicionado como prefixo do identificador da variável

Cadeias de caracteres

- podemos ler e escrever cadeias de caracteres de uma maneira mais simples na linguagem C
- se `palavra` é um vetor de caracteres terminado com o caractere nulo, podemos usar o especificador de conversão `%s` para mostrar o conteúdo completo da cadeia de caracteres `palavra` na saída padrão

```
printf("%s\n", palavra);
```

- a função `scanf` pode ser usada com o especificador de conversão `%s` para ler uma cadeia de caracteres até que a leitura de um branco seja realizada

```
scanf("%s", palavra);
```

- na leitura de cadeias de caracteres, o símbolo `&` **não** é adicionado como prefixo do identificador da variável

Cadeias de caracteres

- podemos ler e escrever cadeias de caracteres de uma maneira mais simples na linguagem C
- se `palavra` é um vetor de caracteres terminado com o caractere nulo, podemos usar o especificador de conversão `%s` para mostrar o conteúdo completo da cadeia de caracteres `palavra` na saída padrão

```
printf("%s\n", palavra);
```

- a função `scanf` pode ser usada com o especificador de conversão `%s` para ler uma cadeia de caracteres até que a leitura de um branco seja realizada

```
scanf("%s", palavra);
```

- na leitura de cadeias de caracteres, o símbolo `&` **não** é adicionado como prefixo do identificador da variável

Cadeias de caracteres

- se na execução da função `scanf` um(a) usuário(a) digita os caracteres `abcdefg`, a cadeia de caracteres `"abcdefg"` é armazenada no vetor `palavra`
- se, diferentemente, um(a) usuário(a) digita os caracteres `Campo Grande`, então apenas a cadeia de caracteres `"Campo"` é armazenada no vetor `palavra`, devido ao branco (`␣`)
- os caracteres restantes da cadeia digitada ficarão disponíveis no *buffer* de entrada até que uma próxima chamada à função `scanf` seja realizada

Cadeias de caracteres

- se na execução da função `scanf` um(a) usuário(a) digita os caracteres `abcdefg`, a cadeia de caracteres `"abcdefg"` é armazenada no vetor `palavra`
- se, diferentemente, um(a) usuário(a) digita os caracteres `Campo Grande`, então apenas a cadeia de caracteres `"Campo"` é armazenada no vetor `palavra`, devido ao branco (`␣`)
- os caracteres restantes da cadeia digitada ficarão disponíveis no *buffer* de entrada até que uma próxima chamada à função `scanf` seja realizada

Cadeias de caracteres

- se na execução da função `scanf` um(a) usuário(a) digita os caracteres `abcdefg`, a cadeia de caracteres `"abcdefg"` é armazenada no vetor `palavra`
- se, diferentemente, um(a) usuário(a) digita os caracteres `Campo Grande`, então apenas a cadeia de caracteres `"Campo"` é armazenada no vetor `palavra`, devido ao branco (`␣`)
- os caracteres restantes da cadeia digitada ficarão disponíveis no *buffer* de entrada até que uma próxima chamada à função `scanf` seja realizada

Cadeias de caracteres

- para evitar os brancos na leitura de uma cadeia de caracteres, usamos o especificador de conversão `%[...]`, que também é usado na leitura de cadeias de caracteres, delimitando, dentro dos colchetes, quais são os caracteres permitidos em uma leitura
- qualquer outro caractere diferente dos especificados dentro dos colchetes finalizam a leitura
- podemos inverter essas permissões, indicando o caractere `^` como o primeiro caractere dentro dos colchetes

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

Cadeias de caracteres

- para evitar os brancos na leitura de uma cadeia de caracteres, usamos o especificador de conversão `%[...]`, que também é usado na leitura de cadeias de caracteres, delimitando, dentro dos colchetes, quais são os caracteres permitidos em uma leitura
- qualquer outro caractere diferente dos especificados dentro dos colchetes finalizam a leitura
- podemos inverter essas permissões, indicando o caractere `^` como o primeiro caractere dentro dos colchetes

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

Cadeias de caracteres

- para evitar os brancos na leitura de uma cadeia de caracteres, usamos o especificador de conversão `%[...]`, que também é usado na leitura de cadeias de caracteres, delimitando, dentro dos colchetes, quais são os caracteres permitidos em uma leitura
- qualquer outro caractere diferente dos especificados dentro dos colchetes finalizam a leitura
- podemos inverter essas permissões, indicando o caractere `^` como o primeiro caractere dentro dos colchetes

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

Cadeias de caracteres

- É importante destacar que a função `scanf` termina automaticamente uma cadeia de caracteres que é lida com o especificador de conversão `"%s"` ou `"%[...]"` com um caractere nulo, fazendo assim que o vetor de caracteres se torne de fato uma cadeia de caracteres após sua leitura

Cadeias de caracteres

```
#include <stdio.h>

#define MAX 20

int main(void)
{
   char palavra[MAX+1];
   int n;

   printf("Informe uma palavra (com até %d caracteres): ", MAX);
   scanf("%s", palavra);
   n = 0;
   while (palavra[n] != '\0')
      n++;

   printf("A palavra [%s] tem %d caracteres\n", palavra, n);

   return 0;
}
```

Exercícios

1. Dada uma frase com no máximo 100 caracteres, determinar quantos caracteres espaço a frase contém.

```c
#include <stdio.h>

#define MAX 100

int main(void)
{
   char frase[MAX+1];
   int esp, i;

   printf("Informe uma frase: ");
   scanf(" %[^\n]", frase);
   esp = 0;
   for (i = 0; frase[i] != '\0'; i++)
      if (frase[i] == ' ')
         esp++;
   printf("Frase tem %d espaços\n", esp);

   return 0;
}
```

Exercícios

1. Dada uma frase com no máximo 100 caracteres, determinar quantos caracteres espaço a frase contém.

```
#include <stdio.h>

#define MAX 100

int main(void)
{
   char frase[MAX+1];
   int esp, i;

   printf("Informe uma frase: ");
   scanf(" %[^\n]", frase);
   esp = 0;
   for (i = 0; frase[i] != '\0'; i++)
      if (frase[i] == ' ')
         esp++;
   printf("Frase tem %d espaços\n", esp);

   return 0;
}
```

Exercícios

2. Dada uma cadeia de caracteres com no máximo 100 caracteres, contar a quantidade de letras minúsculas, letras maiúsculas, dígitos, espaços e símbolos de pontuação que essa cadeia possui.
3. Dadas duas cadeias de caracteres `cadeia1` e `cadeia2`, concatenar `cadeia2` no final de `cadeia1`, colocando o caractere nulo no final da cadeia resultante. A cadeia resultante a ser mostrada deve estar armazenada em `cadeia1`. Suponha que as cadeias sejam informadas com no máximo 100 caracteres.
4. Dada uma cadeia de caractere `cadeia` com no máximo 100 caracteres e um caractere `c`, buscar a primeira ocorrência de `c` em `cadeia`. Se `c` ocorre em `cadeia`, mostrar a posição da primeira ocorrência; caso contrário, mostrar o valor `-1`.

Exercícios

2. Dada uma cadeia de caracteres com no máximo 100 caracteres, contar a quantidade de letras minúsculas, letras maiúsculas, dígitos, espaços e símbolos de pontuação que essa cadeia possui.
3. Dadas duas cadeias de caracteres `cadeia1` e `cadeia2`, concatenar `cadeia2` no final de `cadeia1`, colocando o caractere nulo no final da cadeia resultante. A cadeia resultante a ser mostrada deve estar armazenada em `cadeia1`. Suponha que as cadeias sejam informadas com no máximo 100 caracteres.
4. Dada uma cadeia de caractere `cadeia` com no máximo 100 caracteres e um caractere `c`, buscar a primeira ocorrência de `c` em `cadeia`. Se `c` ocorre em `cadeia`, mostrar a posição da primeira ocorrência; caso contrário, mostrar o valor `-1`.

Exercícios

2. Dada uma cadeia de caracteres com no máximo 100 caracteres, contar a quantidade de letras minúsculas, letras maiúsculas, dígitos, espaços e símbolos de pontuação que essa cadeia possui.
3. Dadas duas cadeias de caracteres `cadeia1` e `cadeia2`, concatenar `cadeia2` no final de `cadeia1`, colocando o caractere nulo no final da cadeia resultante. A cadeia resultante a ser mostrada deve estar armazenada em `cadeia1`. Suponha que as cadeias sejam informadas com no máximo 100 caracteres.
4. Dada uma cadeia de caractere `cadeia` com no máximo 100 caracteres e um caractere `c`, buscar a primeira ocorrência de `c` em `cadeia`. Se `c` ocorre em `cadeia`, mostrar a posição da primeira ocorrência; caso contrário, mostrar o valor `-1`.

Exercícios

5. Dadas duas cadeias de caracteres `cadeia1` e `cadeia2`, cada uma com no máximo 100 caracteres, compará-las e devolver um valor menor que zero se `cadeia1` é lexicograficamente menor que `cadeia2`, o valor zero se `cadeia1` é igual ou tem o mesmo conteúdo que `cadeia2`, ou um valor maior que zero se `cadeia1` é lexicograficamente maior que `cadeia2`.
6. Dadas duas sequências de caracteres (uma contendo uma frase e outra contendo uma palavra), determine o número de vezes que a palavra ocorre na frase. Considere que essas sequências têm no máximo 100 caracteres cada uma.
   Exemplo:
   Para a palavra ANA e a frase:

   ANA E MARIANA GOSTAM DE BANANA.

   Temos que a palavra ocorre 4 vezes na frase.

Exercícios

5. Dadas duas cadeias de caracteres `cadeia1` e `cadeia2`, cada uma com no máximo 100 caracteres, compará-las e devolver um valor menor que zero se `cadeia1` é lexicograficamente menor que `cadeia2`, o valor zero se `cadeia1` é igual ou tem o mesmo conteúdo que `cadeia2`, ou um valor maior que zero se `cadeia1` é lexicograficamente maior que `cadeia2`.
6. Dadas duas sequências de caracteres (uma contendo uma frase e outra contendo uma palavra), determine o número de vezes que a palavra ocorre na frase. Considere que essas sequências têm no máximo 100 caracteres cada uma.
   Exemplo:
   Para a palavra ANA e a frase:

   ANA E MARIANA GOSTAM DE BANANA.

   Temos que a palavra ocorre 4 vezes na frase.