Pontifícia Universidade Católica
do Rio de Janeiro

INF 1007 – Programação II

Aula 05 – Cadeias de Caracteres

Edirlei Soares de Lima
<elima@inf.puc-rio.br>

visionlab
visualization laboratory

Caracteres

- Caracteres são representados através de **códigos numéricos**.

- **Tabela de códigos:**
  - Define correspondência entre caracteres e códigos numéricos
  - Exemplo: Tabela **ASCII**
  - Alguns alfabetos precisam de maior representatividade
    - alfabeto chinês tem mais de 256 caracteres
    - no nosso caso, cada letra do alfabeto tem uma representação numérica diferente

Códigos ASCII de Alguns Caracteres

	0	1	2	3	4	5	6	7	8	9
30			sp	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	S	t	u	v	w
120	x	y	z	{	|	}	~			

*sp representa espaço

Exemplo:

82	105	111	32	100	101	32	74	97	110	101	105	114	111
R	i	o		d	e		J	a	n	e	i	r	o

Códigos ASCII de Caracteres de Controle

0	`nul`	*null*: nulo
7	`bel`	*bell*: campainha
8	`bs`	*backspace*: volta e apaga um caractere
9	`ht`	*tab*: tabulação horizontal
10	`nl`	*newline* ou *line feed*: muda de linha
13	`cr`	*carriage return*: volta ao início da linha
127	`del`	*delete*: apaga um caractere

Caracteres

- Um caractere definido como uma **constante** é representado envolvido com **aspas simples**.
  - **Exemplo:**

```
char c = 'a';
printf("%d %c\n", c, c);
```

  - O printf imprime o conteúdo da variável usando dois formatos:
    - com o formato para inteiro, %d, imprime 97
    - com o formato de caractere, %c, imprime a (código 97 em ASCII)

Caracteres

- Manipulando o código ASCII é possível criar uma função para **converter minúsculas em maiúsculas**:

```
char maiuscula(char c)
{
  /* Verifica se é letra minúscula */
  if (c >= 'a' && c <= 'z')
    c = (c - 'a') + 'A';

  return c;
}
```

- Essa função tira proveito da **forma sequencial** em que os caracteres são caracteres representados na tabela ASCII

Cadeias de Caracteres

- Uma cadeia de caracteres, mais conhecida como **string**, é uma sequência de letras e símbolos.

- A linguagem C não possui um tipo de dado para armazenar strings, mas ela permite a criação de vetores de caracteres.

  – Vetor do tipo **char**, terminado pelo caractere **nulo** ('\0')
    - é necessário reservar uma posição adicional no vetor para o caractere de fim da cadeia

  – As funções que manipulam cadeias de caracteres:
    - recebem como parâmetro um vetor de char
    - processam caractere por caractere até encontrar o caractere nulo, sinalizando o final da cadeia

Cadeias de Caracteres

- A **inicialização de uma cadeia de caracteres** pode ser feita de forma semelhante a inicialização de um vetor (colocando cada caracteres entre aspas duplas)

- A linguagem C também fornece uma forma de inicialização **mais simples** utilizando aspas duplas (o caractere nulo é representado implicitamente)

```
int main (void)
{
   char cidade[] = "Rio";
   printf("%s \n", cidade);
   return 0;
}
```

≡

```
int main (void)
{
   char cidade[]={'R', 'i', 'o', '\0'};
   printf("%s \n", cidade);
   return 0;
}
```

Cadeias de Caracteres

- **Exemplos:**

```
char S1[] = "";
char S2[] = "Rio de Janeiro";
char S3[81];
char S4[81] = "Rio";
```

  - **S1** cadeia de caracteres vazia (armazena o caractere '\0')
  - **S2** armazena cadeia de 14 caracteres (em vetor com 15 elementos)
  - **S3** armazena cadeia com até 80 caracteres dimensionada com 81 elementos, mas não inicializada
  - **S4** armazena cadeias com até 80 caracteres com os primeiros quatro elementos atribuídos na declaração {'R', 'i', 'o', '\0'};

Leitura de Caracteres

- É possível utilizar o **scanf** para ler caracteres e cadeias de caracteres.
  - O formato **%c** permite a leitura de um único caractere:

```
char a;
...
scanf("%c", &a);
...
```

Leitura de Cadeias de Caracteres

- É possível utilizar o **scanf** para ler caracteres e cadeias de caracteres.
  - O formato **%s** permite a leitura de uma cadeia de caracteres não brancos:

```
char cidade[81];
...
scanf("%s", cidade);
...
```

  - Não é necessário usar &cidade pois cidade é um vetor (ponteiro)
  - Somente lê palavras simples, se o usuário digitar “Rio de Janeiro”, somente “Rio” será capturado, pois %s lê somente uma sequência de caracteres **não brancos**.

Leitura de Cadeias de Caracteres

- É possível utilizar o especificador de formato **%[...]** no scanf para definir os caracteres que **podem e não podem** ser lidos.
  - **%[…]** – os caracteres **aceitos** devem ser listados dentro dos colchetes.
  - **%[^…]** – os caracteres **não aceitos** devem ser listados dentro dos colchetes.
- **Exemplos:**
  - **%[aeiou]**
    - Lê sequências de vogais
    - A leitura prossegue até encontrar um caractere que não seja uma vogal
  - **%[^aeiou]**
    - Lê sequências de caracteres que não são vogais
    - A leitura prossegue até encontrar um caractere que seja uma vogal

Leitura de Cadeias de Caracteres

- **%[^\n]**
  - Lê uma sequência de caracteres até que seja encontrado o caractere de mudança de linha ('\n')
  - A captura a cadeia de caracteres até que o usuário aperte “Enter” (nova linha)
  - Permite ler cadeias de caracteres que **com espaços em branco**

```
char cidade[81];
...
scanf(" %[^\n]", cidade);
...
```

pula eventuais caracteres brancos que precedam a cadeia de caracteres

- **%80[^\n]**
  - Limita o tamanho máximo da cadeia de caracteres em 80

```
char cidade[81];
...
scanf("%80[^\n]", cidade);
...
```

Manipulação de Cadeias de Caracteres

- **Função “imprime”:**
  - Imprime na tela uma cadeia de caracteres, caractere por caractere, com uma quebra de linha no final:

```
void imprime(char s[])
{
   int i;
   for (i=0; s[i] != '\0'; i++)
      printf("%c", s[i]);
   printf("\n");
}
```

  - Ou usando o printf:

```
void imprime(char s[])
{
  printf("%s\n", s);
}
```

Manipulação de Cadeias de Caracteres

- **Função “comprimento”:**
  - Retorna o comprimento de uma cadeia de caracteres
    - Conta até encontrar o caractere nulo (`'\0'`)
    - O caractere nulo em si não é contado

```
int comprimento(char s[])
{
   int i;
   int n = 0;
   for (i=0; s[i] != '\0'; i++)
   {
      n++;
   }
   return n;
}
```

Manipulação de Cadeias de Caracteres

- **Função “copia”:**
  - copia os caracteres de uma cadeia de origem (orig) para uma cadeia de destino (dest)
    - A cadeia de destino deverá ter espaço suficiente

```
void copia(char dest[], char orig[])
{
   int i;
   for (i=0; orig[i] != '\0'; i++)
   {
      dest[i] = orig[i];
   }

   dest[i] = '\0'; /* fecha a cadeia copiada */
}
```

Manipulação de Cadeias de Caracteres

- **Função “concatena”:**
  - copia os caracteres de uma cadeia de origem (orig) para o final de uma cadeia de destino (dest)

```
void concatena(char dest[], char orig[])
{
  int i = 0;    /* índice usado na cadeia destino */
  int j;        /* índice usado na cadeia origem */
  while (dest[i] != '\0') /* acha o final da cadeia destino */
  {
     i++;
  }
  /* copia elementos da origem para o final do destino */
  for (j=0; orig[j] != '\0'; j++)
  {
    dest[i] = orig[j];
    i++;
  }
  dest[i] = '\0'; /* fecha cadeia destino */
}
```

Manipulação de Cadeias de Caracteres

- **Função “compara”:**
  - Compara, caractere por caractere, duas cadeias de caracteres
    - usa os códigos numéricos associados aos caracteres para determinar a ordem relativa entre eles
  - Valor de retorno da função:

-1	se a primeira cadeia preceder a segunda
1	se a segunda preceder a primeira
0	se ambas as cadeias tiverem a mesma sequência de caracteres

Manipulação de Cadeias de Caracteres

```
int compara(char s1[], char s2[])
{
   int i;
   /* compara caractere por caractere */
   for (i=0; s1[i]!='\0' && s2[i]!='\0'; i++)
   {
      if (s1[i] < s2[i])
         return -1;
      else if (s1[i] > s2[i])
         return 1;
   }
   /* compara se cadeias têm o mesmo comprimento */
   if (s1[i]==s2[i])
      return 0;          /* cadeias iguais */
   else if (s2[i]!= '\0')
      return -1;         /* s1 é menor, pois tem menos caracteres */
   else
      return 1;          /* s2 é menor, pois tem menos caracteres */
}
```

Manipulação de Cadeias de Caracteres

- Biblioteca de cadeias de caracteres **string.h**:
  - **strlen**: determina o comprimento de uma cadeia;
  - **strcpy**: copia uma cadeia origem para outra destino;
  - **strcat**: concatena duas cadeias;
  - **strcmp**: compara duas cadeias;

Funções da Biblioteca `string.h`

- A função **strlen** retorna o número de caracteres em uma cadeia de caracteres:
  - O caractere nulo (`'\0'`) não é contado

```
int strlen(char str[])
```

- Exemplo:

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

void main(void)
{
  char nome[] = "Curso de Programacao 1";
  printf("%s contem %d caracteres", nome, strlen(nome));
}
```

Funções da Biblioteca `string.h`

- A função **strcpy** é usada para copiar o conteúdo de uma cadeia de caracteres (fonte), inclusive o caractere nulo, para outra cadeia (destino):

```
char* strcpy(char destino[], char fonte[])
```

- Exemplo:

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

void main(void)
{
  char nome[]="Joao da Silva";
  char aluno[50];
  strcpy(aluno, nome);
  printf("O nome do aluno eh %s”, aluno);
}
```

Funções da Biblioteca `string.h`

- A função **strcmp** é usada para fazer a comparação lexicográfica de duas cadeias de caracteres:

```
int strcmp(char str1[], char str2[])
```

  - Se as cadeias são **iguais**, isto é, se str1 e str2 têm mesmo comprimento e caracteres iguais nos elementos de mesmo índice, a função retorna 0;
  - Se **str1** for lexicograficamente **maior** do que **str2**, a função retorna um valor positivo;
  - Se **str2** for lexicograficamente **maior** do que **str1**, a função retorna um valor negativo.

Funções da Biblioteca `string.h`

- Exemplo de utilização da função **strcpy:**

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

void main(void)
{
  char nome1[]="Joao da Silva";
  char nome2[]="Maria Fernanda";

  if(!strcmp(nome1, nome2))
    printf("%s e igual a %s", nome1, nome2);
  else if(strcmp(nome1, nome2)>0)
    printf("%s vem depois de %s", nome1, nome2);
  else
    printf("%s vem depois de %s", nome2, nome1);
}
```

Funções da Biblioteca `string.h`

- A função **strcat** é usada para concatenar o conteúdo de uma cadeia de caracteres (fonte) ao final de outra cadeia (destino):

```
char* strcat(char destino[], char fonte[])
```

- Exemplo:

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

void main(void)
{
  char nome[30]="Joao";
  char sobreNome[]=" da Silva";
  strcat(nome, sobreNome);
  printf("O nome do aluno eh %s", nome);
}
```

Alocação Dinâmica de Cadeias de Caracteres

- **Função “duplica”:**
  - copia os elementos de uma cadeia de origem (s) para uma cadeia de destino (d), alocada dinamicamente:

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

char* duplica(char* s)
{
  int n = strlen(s);
  char* d = (char*) malloc((n + 1) * sizeof(char));
  strcpy(d, s);
  return d;
}
```

Alocação Dinâmica - Exemplo

- Encontrar o prefixo de n caracteres em uma cadeia de caracteres:
  - Exemplo: “Programação II” -> “Prog” (prefixo de tamanho 4);

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

char *prefixo(char *str, int n)
{
  char *pre;
  int x;
  pre = (char*)malloc((n+1)*sizeof(char));

  for (x = 0; x < n; x++)
    pre[x] = str[x];

  pre[n] = '\0';
  return pre;
}
```

Alocação Dinâmica - Exemplo

[continuação...]

```
int main (void)
{
  char nome[] = "PROGRAMACAO II";

  char *res = prefixo(nome, 4);

  printf("%s", res);

  return 0;
}
```

Constante de Cadeia de Caracteres

- Representada por sequência de caracteres delimitada por aspas duplas;
- Comporta-se como uma expressão constante, cuja avaliação resulta no ponteiro para onde a cadeia de caracteres está armazenada;

```
#include <string.h>

int main(void)
{
  char cidade[4];
  strcpy(cidade, "Rio");
  printf("%s \n", cidade);
  return 0;
}
```

```
#include <string.h>

int main (void)
{
  char *cidade;
  cidade = "Rio";
  printf("%s \n", cidade);
  return 0;
}
```

Constante de Cadeia de Caracteres

- Exemplos:

```
char s1[] = "Rio de Janeiro";
```

  - s1 é um vetor de char, inicializado com a cadeia Rio de Janeiro, seguida do caractere nulo;
  - s1 ocupa 15 bytes de memória;
  - é válido escrever s1[0]='X', alterando o conteúdo da cadeia para Xio de Janeiro, pois s1 é um vetor, permitindo alterar o valor de seus elementos;

```
char* s2 = "Rio de Janeiro";
```

  - s2 é um ponteiro para char, inicializado com o endereço da área de memória onde a constante Rio de Janeiro está armazenada;
  - s2 ocupa 4 bytes (espaço de um ponteiro);
  - não é válido escrever s2[0]='X', pois não é possível alterar um valor constante;

Manipulação de Cadeias de Caracteres

- Implementação recursiva da função “imprime”:

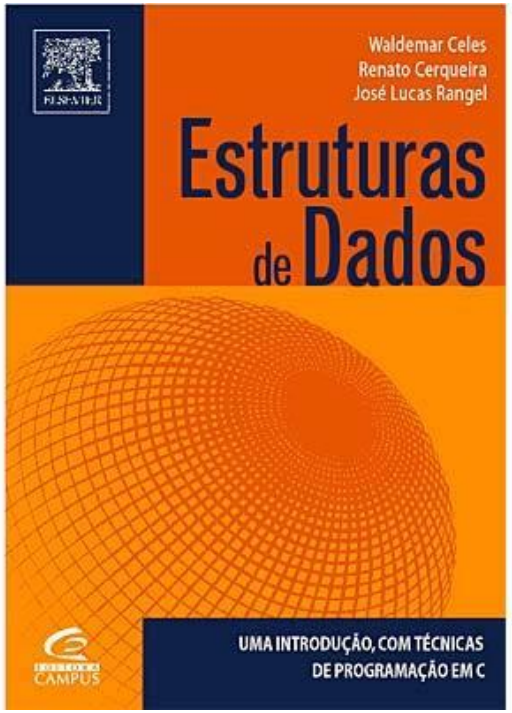
```
void imprime_rec(char* s)
{
  if (s[0] != '\0')
  {
    printf("%c", s[0]);
    imprime_rec(&s[1]);
  }
}
```

```
int main (void)
{
  char cidade[] = "Rio de Janeiro";
  imprime_rec(&cidade[0]);
  return 0;
}
```

Leitura Complementar

- Waldemar Celes, Renato Cerqueira, José Lucas Rangel, **Introdução a Estruturas de Dados**, Editora Campus (2004).

- **Capítulo 6 – Cadeia de caracteres**



Exercícios

Lista de Exercícios 03 – Strings e Recursão

http://www.inf.puc-rio.br/~elima/prog2/