Tratamento de Caracteres

Notas de Aula

Prof. Francisco Rapchan
www.geocities.com/chicorapchan
rapchan@terra.com.br

A tabela ASCII

Se os computadores manipulam apenas números, como eles podem armazenar os caracteres de um nome ou de um endereço? Uma das formas é o uso da Tabela **ASCII** (*American Standard Code for Information Interchange*). Ela permite converter cada caractere em um número representado por um código de 8 bits (um byte). A cada caractere é atribuído um número de 0 a 127. Abaixo mostramos parte da tabela ASCII. Existe uma tabela ASCII estendida que inclui outros caracteres (inclusive alguns acentuados).

Decimal	Caractere	Decimal	Caractere	Decimal	Caractere	Decimal	Caractere
32	espaço	58	:	84	T	110	n
33	!	59	;	85	U	111	o
34	"	60	<	86	V	112	p
35	#	61	=	87	w	113	q
36	$	62	>	88	X	114	r
37	%	63	?	89	Y	115	s
38	&	64	@	90	Z	116	t
39	'	65	A	91	[	117	u
40	(	66	B	92	\	118	v
41	)	67	C	93	]	119	w
42	*	68	D	94	^	120	x
43	+	69	E	95	_	121	y
44	,	70	F	96	`	122	z
45	-	71	G	97	a	123	{
46	.	72	H	98	b	124	|
47	/	73	I	99	c	125	}
48	0	74	J	100	d	126	~
49	1	75	K	101	e	127	DEL
50	2	76	L	102	F		
51	3	77	M	103	g		
52	4	78	N	104	H		
53	5	79	O	105	I		
54	6	80	P	106	J		
55	7	81	Q	107	K		
56	8	82	R	108	L		
57	9	83	S	109	M		

Ale dos caracteres acima, a tabela ASCII relaciona também alguns caracteres de controle:

Decimal	Símbolo	Significado
0	nul	null: nulo
7	bel	bell: campainha
8	bs	backspace: voltar e apagar um caractere
9	ht	tab ou tabulação horizontal
10	nl	newline ou line feed: mudança de linha
13	cr	carriage return: volta ao início da linha
127	del	delete: apagar um caractere

Exemplo 1. Leia um número e mostre a letra correspondente em código ASCII.

```
#include <stdio.h>

main ()
{
   char c;

   printf ("Digite um número: ");
   scanf ("%i",&c);
   printf ("Número: %i Letra: %c\n",c);
}
```

A variável c foi declarada como char. O tipo char é considerado um interio pequeno que pode assumir qualquer valor de -128 até +127.

A função printf imprime o conteúdo da variável c usando dois formatos distintos: com o especificador de formato para inteiro, %i, será impresso o valor do código numérico (65, 66, 67...) e com o formato de caractere, %c, será impresso o caractere associado ao código (A, B, C...).

Observe que o usuário só irá perceber saídas se digitar números entre 33 e 126. Os caracteres fora dessa faixa são de controle e, na maioria das vezes, não produz efeito visual.

Um caractere interessante é o 7 (Bell). Ele faz soar um breve som no computador.

Exemplo 2. Faça um programa que mostre os caracteres do A ao Z.

```
#include <stdio.h>
main ()
{
   char c;

   for (c = 65 ; c<= 90 ; c++ )
      printf ("%i   %c\n", c , c);

}
```

Vemos no `printf` que o `%i` indica que será mostrado um número inteiro e o `%c` indica que será mostrado um caractere.

Nesta solução a variável c via de 65 até 90 explicitamente.

Outra solução:

```
#include <stdio.h>

main ()
{
   char c, inicio, fim;

   inicio = 'A';
   fim    = 'Z';

   for (c = inicio; c <= fim; c++)
      printf ("%i   %c\n", c , c)
}
```

Nesta solução, a variável inicio recebe o número correspondente ao caractere A e a variável fim recebe o número correspondente ao caractere Z.

O efeito é exatamente o mesmo da solução anterior.

Poderíamos também ter construído uma solução sem usar as variáveis inicio e fim. Poderíamos ter feito:

```
for (c = 'A'; c <= 'Z'; c++)
   printf ("%i   %c\n", c , c)
```

Neste caso, c inicia com o valor de ‘a’ (que é 65).

Cadeia de Caracteres (Strings)

Cadeias de caracteres (strings), em C, são representadas por vetores do tipo char terminadas, obrigatoriamente, pelo caractere nulo ('\0'). Portanto, para armazenarmos uma cadeia de caracteres, devemos reservar uma posição adicional para o caractere de fim da cadeia.

Exemplo 3. Atribua o nome Chico a um vetor de caracteres e mostre o valor dessa variável na tela.

```
#include <stdio.h>

main()
{
   char nome[6];

   nome[0] = 'C';
   nome[1] = 'h';
   nome[2] = 'i';
   nome[3] = 'c';
   nome[4] = 'o';
   nome[5] = '\0';

   printf("%s \n", nome);
}
```

Observe que criamos a variável nome com 20 posições. Assim, ela pode receber até 19 caracteres pois o último é reservado para o caracter '\0' (barra zero) que indica o fim da string. Sem esse caractere muitas funções que manipulam strings (como o printf) não funcionarão corretamente.

Optou-se por incluir a palavra Chico letra por letra (apenas para fins didáticos). Observe que, ao final, foi incluído o caractere '\0'.

Teste este programa sem o '\0' para ver o resultado. O printf mostrará todas as letras inclusive o lixo que houver até encontrar um '\0'.

Note também que não usamos o caractere **&** na passagem do parâmetro nome para a função **scanf.** Isso acontece porque a cadeia de caracteres é um vetor e o nome da variável representa o endereço do primeiro elemento do vetor e a função atribui os valores dos elementos a partir desse endereço. Ou seja, o nome do vetor de caracteres representa o próprio endereço do vetor.

Outra solução:

```
#include <stdio.h>

main()
{
   char nome[] = "Chico";

   printf("%s \n", nome);

}
```

Iniciar cadeias de caracteres é tão comum em códigos C que a linguagem permite que isso seja feito escrevendo os caracteres entre aspas duplas.

Entretanto, só é permitido atribuir um valor na declaração da variável. Assim, não é possível o código abaixo:

```
char nome[6];
nome = "Chico"; // ERRADO!!!
```

O especificador %s pode ser usado na função scanf para capturar cadeias de caracteres. Entretanto, seu uso é muito limitado pois só captura até o primeiro caractere branco. Por exemplo, veja o código abaixo:

```
char nome[21];
scanf ("%s", nome);
printf("%s \n", nome);
```

Se digitarmos Francisco José, só apareceria a palavra: Francisco.

Uma solução é usar a seguinte estrutura de scanf:

```
char nome[21];
scanf (" %[^\n]", nome); // [^\n] lê dados até que tecle enter.
printf("%s \n", nome);
```

A função scanf agora lê uma seqüência de caracteres até que seja encontrado o caractere de mudança de linha ('\n'). Ou seja: captura-se o texto digitado pelo usuário até que ele tecle "*Enter*". A **inclusão do espaço** no formato (antes do sinal %) garante que eventuais caracteres brancos que precedam o nome serão pulados (**isso é muito importante!**).

Observe que o código acima não controla o número de caracteres que é atribuído à variável nome. Isso é perigoso, pois, se o usuário digitar um nome que tenha mais de 20 caracteres, estaremos invadindo um espaço de memória que não está reservado (o vetor nome foi dimensionado com 21 elementos).

Para evitar esta invasão, podemos limitar o número máximo de caracteres que serão capturados:

```
char nome[21];
scanf (" %20[^\n]", nome); // %20 limita em 20 caracteres o tamanho de nome
printf("%s \n", nome);
```

A função **gets( )** também é usada para ler uma string do teclado. Entretanto essa função **não controla quantos caracteres foram lidos**.

Exemplo de uso da função gets( ):

```
char nome[21];
gets(nome);      // a função gets deve ser evitada!!!
printf("%s \n", nome);
```

A biblioteca **<string.h>** possui uma série de funções para manipulação de string. Dentre elas, destacamos:

strcpy (string_destino, string_origem);	Copia a string-origem para a string- destino.
strcat (string_destino,string_origem);	A string de origem permanecerá inalterada e será anexada ao fim da string de destino.
strlen (string);	Retorna o comprimento da string fornecida. O terminador nulo não é contado.
strcmp (string1,string2);	Compara a string 1 com a string 2. Se as duas forem idênticas a função retorna zero (falso). Se elas forem diferentes a função retorna não-zero (verdadeiro): positivo se string1 > string2 e negativo se string1 < string2

Exemplo 4. O programa abaixo faz uso das funções apresentadas.

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

main()
{
   char nome1[11];             // Cria 3 strings
   char nome2[11];
   char aux  [21];
   int tamanho;                // Conterá o tamanho da string

   scanf (" %10[^\n]", nome1); // Lê as strings nome1 e nome2 do teclado
   scanf (" %10[^\n]", nome2);

   // Compara se as strings são iguais
   if (strcmp (nome1, nome2))
      printf ("Os nomes são diferentesn\n");
   else
      printf ("Os nomes são iguais\n");

   // Obtém o tamanho da string em nome1
   tamanho = strlen (nome1);
   printf ("%s tem %i caracteres\n",nome1, tamanho);

   // Coloca nome1 e nome2 na string aux e mostra
   strcpy (aux, nome1);
   strcat (aux, " e ");
   strcat (aux, nome2);

   printf ("%s\n",aux);

}
```

Matrizes de Caracteres ou Vetores de Strings

Uma string é um vetor de caracteres. Assim, ao criarmos um vetor de strings estaremos fazendo, na verdade, um vetor de vetores.

Podemos ver a forma geral de uma matriz de strings como sendo:

```
char nome_da_variável [quantidade_de_strings][tamanho_das_strings];
```

Exemplo 5:

Suponha o seguinte segmento de código:

```
char nome [10];
strcpy (nome, "Maria");
```

Teríamos na memória:

0	1	2	3	4	5	6	7	8	9
M	a	r	i	a	\0	lixo	lixo	lixo	lixo

No caso do segmento de código abaixo:

```
char nomes [10][4];
strcpy (nome[0],"Maria");
strcpy (nome[1],"Ana Luiza");
strcpy (nome[2],"Jona");
strcpy (nome[3],"Carla");
```

Teríamos na memória:

	0	1	2	3	4	5	6	7	8	9
0	**M**	**a**	**r**	**i**	**a**	\0	lixo	lixo	lixo	lixo
1	**A**	**n**	**a**		**L**	**u**	**i**	**z**	**a**	\0
2	**J**	**o**	**a**	**n**	**a**	\0	lixo	lixo	lixo	lixo
3	**C**	**a**	**r**	**l**	**a**	\0	lixo	lixo	lixo	lixo

Exemplo 6. Faça um programa que leia 10 nomes e coloque em um vetor. Depois, mostre estes nomes em ordem inversa da que foram digitados.

```
#include <stdio.h>

main()
{
   char nomes[10][41];
   int c;

   for (c=0; c < 10 ; c++)
   {
      printf ("Nome %i:",c+1);
      scanf (" %40[^\n]",nomes[c]);
   }

   for (c=9; c >= 0 ; c--)
      printf ("%s\n",nomes[c]);
}
```

Neste exemplo teremos 10 nomes de até 40 caracteres cada um (o 41 é usado para conter o \0).

No `printf` mostramos o valor `c + 1`. Dessa forma, embora o vetor comece na posição zero, são solicitados os nomes a partir da posição 1 (apenas por questões de interface com o usuário).

Poderíamos ter usado `gets( )`:

```
for (c=0; c < 10 ; c++)
{
   printf ("Nome %i:",c+1);
   gets (nomes[c]);
}
```

Observe último `for` sendo usado com contador decrescente para varrer o vetor ao contrário.

Exemplo 7. Faça um programa que leia 10 nomes e coloque em um vetor. Depois, leia um nome e mostre em quais posições do vetor ele aparece.

```
#include <stdio.h>
#include <string.h>
main()
{
   char nomes[10][41];
   char busca [41];
   int c;

   for (c=0; c < 10 ; c++) {
      printf ("Nome %i:",c+1);
      gets (nomes[c]);
   }

   printf ("Nome para procurar: ");
   gets (busca);

   for (c=0; c < 10 ; c++)
     if (strcmp (busca, nomes[c]) == 0)
        printf ("posicao: %i\n",c+1);

}
```

Na variável busca teremos o nome que desejamos procurar no vetor.

A função strcmp permite comparar duas strings e, se elas forem iguais o resultado será zero. Isso é feito no seguinte trecho de código:

```
if (strcmp (busca, nomes[c]) == 0)
        printf ("posicao: %i\n",c+1);
```

Neste caso, se busca for igual a nomes[c] então o strcmp retorna zero e o printf será executado. Caso contrário não retorna zero e o printf não será executado.

Observe que neste exemplo usamos a função gets( ) para ler uma string do teclado (poderíamos ter usado scanf).

Exemplo 8. Faça um programa que leia 5 nomes e coloque em um vetor. Depois, coloque-os em ordem alfabética e mostre-os.

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

main(){

   char nomes[5][41], aux[41];
   int i,j;

   // Preenche o vetor de nomes
   for (i=0; i < 5 ; i++)
   {
      printf ("Nome:");
      scanf (" %40[^\n]",nomes[i]);
   }

   // Ordena o vetor de nomes
   for (i=0; i < 5; i++)
      for (j=0; j< 4; j++)
         if (strcmp (nomes[j],nomes[j+1])>0)
         {
            strcpy (aux,nomes[j]);
            strcpy (nomes[j],nomes[j+1]);
            strcpy (nomes[j+1],aux);
         }

   // Mostra o vetor de nomes
   for (i=0; i < 5 ; i++)
      printf ("%s\n",nomes [i]);
}
```

Observe o uso da função strcmp. Ela retorna um valor positivo (maior que zero) se nomes[j] > nomes[j+1].

Exemplo:

```
strcmp (“Adão”, “Eva”)
```

irá retornar um valor negativo pois Eva é maior do que Adão do ponto de vista da ordem alfabética.

Já:

```
strcmp (“Eva”, “Adão”)
```

Retorna um valor negativo.

Dessa forma podemos colocar as cadeias de caracteres em ordem alfabética.

Observe que neste exemplo usamos a função scanf para ler uma string do teclado (poderíamos ter usado gets).