20 Caracteres - Tipo char

Ronaldo F. Hashimoto e Carlos H. Morimoto

Até agora vimos como o computador pode ser utilizado para processar informação que pode ser quantificada de forma numérica. No entanto, há muita informação que não é numérica, como por exemplo o seu nome, seu endereço residencial, uma fotografia sua ou o som de sua voz. Textos são compostos por caracteres do alfabeto, de pontuação, acentuação, etc. Para que possam ser processados pelo computador, precisamos de uma forma para representar caracteres utilizando apenas números.

Ao final dessa aula você deverá saber:

- Declarar, ler e imprimir variáveis do tipo `char`.
- Descrever como caracteres são representados no computador e porque eles podem ser utilizados em expressões.
- Utilizar variáveis do tipo `char` em programas.

20.1 Caracteres

Um caractere pode ser uma letra (maiúscula ou minúscula), um ponto final, ponto de interrogação, colchetes, enfim, símbolos que normalmente encontramos num teclado de um computador. Nesta aula, estudaremos as variáveis utilizadas em C para armazenar caracteres.

Em C, caracteres são armazenados como números inteiros, normalmente utilizando uma tabela de conversão amplamente difundida chamada Tabela ASCII (American Standard Code for Information Interchange).

A Tabela ASCII original possui somente 128 caracteres, aqueles com código entre 0 e 127. Veja na tabela abaixo, a correspondência entre caractere e número (entre os números 32 e 127).

```
  032   ! 033   " 034   # 035   $ 036   % 037   & 038   ' 039
( 040   ) 041   * 042   + 043   , 044   - 045   . 046   / 047
0 048   1 049   2 050   3 051   4 052   5 053   6 054   7 055
8 056   9 057   : 058   ; 059   < 060   = 061   > 062   ? 063
@ 064   A 065   B 066   C 067   D 068   E 069   F 070   G 071
H 072   I 073   J 074   K 075   L 076   M 077   N 078   O 079
P 080   Q 081   R 082   S 083   T 084   U 085   V 086   W 087
X 088   Y 089   Z 090   [ 091   \ 092   ] 093   ^ 094   _ 095
` 096   a 097   b 098   c 099   d 100   e 101   f 102   g 103
h 104   i 105   j 106   k 107   l 108   m 109   n 110   o 111
p 112   q 113   r 114   s 115   t 116   u 117   v 118   w 119
x 120   y 121   z 122   { 123   | 124   } 125   ~ 126     127
```

Existem várias extensões da tabela ASCII utilizando também os números negativos de -1 a -128. Normalmente, estas tabelas estendidas incluem os caracteres acentuados.

A cada caractere corresponde a um número entre 0 e 127 e a cada número entre 0 e 127 corresponde a um caractere. Caractere com código 32 corresponde ao espaço em branco e o caractere com código 127 varia de computador a computador.

Os caracteres com código decimal entre 0 e 31 são chamados de caracteres de controle, ou seja, eles indicam alguma coisa ao que a impressora ou monitor de vídeo devem executar. Entre eles, os mais utilizados são:

Caractere	Código ASCII	Significado
nulo	0	este caractere é simplemente ignorado pela impressora ou vídeo.
backspace	8	volta um caractere
line feed	10	pula uma linha
carriage return	13	o cursor vai para a primeira coluna.

Assim, quando você digita a letra `A` (ou o backspace) no teclado, na verdade, o seu computador armazena na memória o número inteiro `65` (ou o número `8`, respectivamente), ou seja, para armazenar a letra `A`, o computador armazena o número `65`.

20.2 Decorar a Tabela ASCII?

Para evitar a necessidade de decorar a tabela ASCII, na linguagem C, escrever um caractere entre apóstrofes equivale a escrever o seu código ASCII. Assim, escrever `65` e `'A'` são equivalentes. Consequentemente, os comandos `i = 65` e `i = 'A'` são equivalentes, assim como `x = x + 65` e `x = x + 'A'`. Dessa forma, observe o comando `A = 'A'`. Neste caso, `A` é uma variável e `'A'` é o caractere cujo código ASCII é `65`; neste comando, o número `65` é armazenado na variável `A`.

Os caracteres são ordenados de acordo com seu código ASCII. Assim o caractere `'0'` (zero) é menor que o caractere `'1'` (um) pois o código ASCII do caractere zero (`48`) é menor que o código ASCII do caractere um (`49`). Assim, `'0' < '1'`, pois `48 < 49`. Da mesma forma, `'1' < '2' < '9' < ... < 'A' < 'B' < ... < 'Z' < ... < 'a' < 'b' < ... < 'z'`, pois `49 < 50 < ... < 65 < 66 < ... < 90 < ... < 97 < 98 < ... < 122`.

20.3 Variável Tipo Char

Uma variável inteira de 32 bits (4 bytes) pode armazenar números inteiros no intervalo de $-2.147.483.648$ a $+2.147.483.647$. No compilador gcc, este tipo de variável é declarado como `int`. Assim, nestes compiladores, a declaração `int x` declara uma variável inteira `x` que armazena inteiros com 4 bytes, ou seja, a variável inteira `x` pode armazenar inteiros compreendidos entre $-2.147.483.648$ a $+2.147.483.647$.

Para armazenar um caractere, poder-se-ia utilizar uma variável inteira (uma vez que um caractere é um número inteiro). No entanto, observe que estaríamos utilizando mais memória (bytes) do que precisaríamos, uma vez que para qualquer caractere, o seu respectivo código ASCII é inteiro entre -128 a $+127$. Uma variável inteira de 8 bits (1 byte) pode armazenar números inteiros compreendidos entre -128 a $+127$, exatamente o que precisaríamos para armazenar o código ASCII de um caractere. Este tipo de variável é declarado como `char`.

A forma para declarar uma variável do tipo `char` é a mesma para declarar variáveis do tipo `int`; só que em vez de usar a palavra chave `int`, deve-se usar a palavra `char`:

char <`nome_da_variavel`>;

Exemplo: declaração de uma variável do tipo `char` de nome `"ch"`

char `ch`;

Se você quiser declarar várias variáveis, é possível fazer da seguinte forma:

char <`nome_da_variavel_1`>, <`nome_da_variavel_2`>, <`nome_da_variavel_3`>,..., <`nome_da_variavel_n`>;

Exemplo: declaração de duas variáveis do tipo `char` `"ch1"` e `"ch2"`.

char `ch1, ch2`;

20.4 Leitura de um Caractere pelo Teclado

Como vimos nas aulas passadas, para ler um número inteiro pelo teclado, nós usamos o "%d" dentro do comando `scanf`. Assim, para ler um inteiro `x` fazemos:

```
1       int x;
2
3       printf ("Entre com um numero x > 0: ");
4       scanf  ("%d", &x);
```

Para ler um caractere pelo teclado, você deve utilizar "␣%c" dentro do comando `scanf`. Aqui temos duas observações. A primeira é o "espaço em branco" antes do `%c`. A segunda é que, no Linux, a tecla <ENTER> corresponde ao caractere de código ASCII `10`; no Windows, a tecla <ENTER> deve gerar uma sequência de dois caracteres: um cujo código ASCII é `13` e outro de código ASCII `10`. Para o exemplo, vamos considerar que estamos trabalhando no Linux.

Antes de comentar o "truque" do "espaço em branco" antes do `%c`, vamos dar uma idéia do que acontece na leitura de um caractere pelo teclado. O comando `scanf ("%c", &ch)` fica esperando o usuário digitar uma tecla e em seguida um <ENTER>, o programa converte a tecla digitada para o número correspondente ao código ASCII e este número é armazenado na variável inteira de 1 byte de nome `ch`.

Para mostrar o porque do "espaço em branco" antes do `%c` dentro do `scanf`, considere o seguinte trecho de programa que lê dois caracteres usando dois comandos `scanf`:

```
1       char a, b;
2
3       printf ("Entre com um caractere: ");
4       scanf  ("%c", &a);
5
6       printf ("Entre com um outro caractere: ");
7       scanf  ("%c", &b);
```

Note que neste particular exemplo, não foi utilizado "espaço em branco" antes do `%c` dentro de cada `scanf`. Neste trecho de programa, para ler o primeiro caractere, o usuário deverá digitar um caractere do teclado (por exemplo, a letra `'A'`) e posteriormente dar um <ENTER>. Como o <ENTER> é também um caractere (um caractere cujo código ASCII é `10`), ele será lido no segundo `scanf` (que contém a variável `b`). Assim, a variável `a` irá conter o número `65` e a variável `b` o número `10`. Bem, provavelmente, a intenção do programador deste trecho não era ler o <ENTER> na variável `b`.

Para evitar que o seu programa confunda o <ENTER> como caractere a ser lido, é colocado um "espaço em branco" antes do `%c` dentro de cada `scanf`. Assim, o trecho de programa *corrigido* fica:

```
1       char a, b;
2
3       printf ("Entre com um caractere: ");
4       scanf  (" %c", &a);
5
6       printf ("Entre com um outro caractere: ");
7       scanf  (" %c", &b);
```

Só que esta maneira de "enganar" traz consigo um pequeno problema. Além de não ler o caractere <ENTER>, o seu programa não vai ser capaz de ler o caractere "espaço em branco" e o caractere correspondente à tabulação. Mas isso pode não ser incoveniente, uma vez que em muitos problemas de computação envolvendo caracteres, os caracteres de tabulação e "espaço em branco" devem ser mesmo ignorados.

20.5 Impressão de Caracteres

Como vimos nas aulas passadas, para imprimir um número inteiro na tela, nós usamos o "%d" dentro do comando `printf`. Assim, para imprimir um inteiro `x` fazemos:

```
1       int x;
2
3       printf ("Entre com um numero x > 0: ");
4       scanf  ("%d", &x);
5
6       printf ("Número lido foi = %d\n", x);
```

Para imprimir um caractere na tela, nós devemos usar o "%c" dentro do comando `printf`. Note que agora não temos mais o "espaço em branco" antes do %c dentro do `printf`. Assim, considere o seguinte trecho de programa que lê dois caracteres ignorando "espaços em branco", <ENTER> e tabulações:

```
1       char a, b;
2
3       printf ("Entre com um caractere: ");
4       scanf  (" %c", &a);
5       printf ("Primeiro Caractere Digitado: %c\n", a);
6
7       printf ("Entre com um outro caractere: ");
8       scanf  (" %c", &b);
9       printf ("Segundo Caractere Digitado: %c\n", b);
```

Eventualmente, poderíamos estar interessados em imprimir o código ASCII do caractere em vez do próprio caractere. Neste caso, basta colocar %d no lugar do %c. Assim, o seguinte trecho de programa escreve também o código ASCII do caractere digitado:

```
1       char a, b;
2
3       printf ("Entre com um caractere: ");
4       scanf  (" %c", &a);
5       printf ("Primeiro Caractere Digitado: %c (Codigo ASCII = %d)\n",a, a);
6
7       printf ("Entre com um outro caractere: ");
8       scanf  (" %c", &b);
9       printf ("Segundo Caractere Digitado: %c (Codigo ASCII = %d)\n", b, b);
```

20.6 Exemplos de Exercício Envolvendo Caracteres

20.6.1 Exemplo 1

Faça um programa em C que imprima todos os caracteres cujo códigos ASCII estão entre 32 e 126.

Solução:

```c
#include <stdio.h>

int main () {
  char c;

  for (c = 32; c < 127; c++)
  printf("caractere %c com ASCII %d\n", c, c);

  return 0;
}
```

Observe que, como o tipo `char` na verdade armazena um número correspondente ao código ASCII do caractere, esse número pode ser impresso como um número inteiro. No `printf` dentro do `for`, a variável `c` é utilizada tanto como inteiro (usando `'%d'` na impressão), como caractere (usando `'%c'` para impressão).

Você pode utilizar variáveis do tipo `char` dentro de expressões numéricas como se fossem variáveis inteiras, porém, lembre-se de que, como uma variável `char` utiliza apenas um byte, os valores que ela pode representar variam de -128 a $+127$.

Uma solução usando variável inteira é possível. No entanto, há um desperdício de memória, pois uma variável inteira ocupa 4 bytes; enquanto que uma variável do tipo `char` ocupa 1 byte:

```c
#include <stdio.h>

int main () {
  int c;

  for (c = 32; c < 127; c++)
  printf("caractere %c com ASCII %d\n", c, c);

  return 0;
}
```

20.6.2 Exemplo 2

Escreva um programa que leia um caractere minúsculo e transforme-o em maiúsculo.

Solução:

para escrever esse programa, não precisamos utilizar a tabela ASCII diretamente, apenas precisamos saber como ela foi definida.

De uma forma geral, é bom saber que os dígitos de `'0'` a `'9'` ocupam posições na tabela antes das letras maiúsculas `'A'` a `'Z'`, que por sua vez ocupam posições na tabela antes das letras minúsculas de `'a'` a `'z'`. Assim, como vimos anteriormente, é possível comparar dois caracteres, de forma que a seguinte relação é válida: `'0' < '1' < ... < '9' < 'A' < 'B' < ... < 'Z' < 'a' < ... < 'z'`.

Agora, se você observar a tabela, os códigos ASCII dos caracteres `'A'` e `'a'` são 65 e 97, respectivamente. Assim, a diferença entre `'A'` e `'a'` é 32. O mesmo acontece com os caracteres `'B'` e `'b'` e assim por diante. Dessa forma, para converter um caractere minúsculo para maiúculo, basta subtrair o código ASCII do caractere minúsculo de 32 para se obter o código ASCII do caractere maiúsculo. Assim:

```c
char ch;

ch = 'a';

ch = ch - 32;

printf ("Caractere Maiusculo = %c", ch);
```

Na linha 3, a variável `ch` recebe o código ASCII do caractere `'a'` minúsculo, ou seja, o número `97`. Na linha 5, o valor de `ch` é subtraído de `32` ficando com o valor `65`. Na linha 7, o caractere cujo ASCII é `65` é impresso, ou seja, o caractere `'A'`.

Agora, e se você não soubesse que a diferença entre os códigos ASCII entre maiúsculas e minúsculas fosse `32`. E se um dia construirem uma tabela diferente em que esta diferença mudasse? O que a gente poderia fazer é a seguinte conta:

```
1       char dif;
2
3       dif = 'a' - 'A';
```

A variável `dif` guarda o resultado da diferença entre os códigos ASCII dos caracteres `'a'` e `'A'`.

Assim, uma solução para o nosso exercício de conversão de minúscula para maiúscula pode ser:

```
1   #include <stdio.h>
2
3   int main () {
4     char letra, dif;
5     printf ("Digite uma letra: ");
6     scanf (" %c", &letra);
7     dif = ('a' - 'A');
8     if (letra >= 'a' && letra <= 'z') {
9       /* sabemos que eh uma letra minuscula */
10      letra = letra - dif;
11      printf ("Maiuscula: %c\n", letra);
12    }
13    else
14      printf("%c nao e uma letra minuscula\n", letra);
15
16    return 0;
17  }
```

20.7 Exercícios recomendados

1. Dada uma sequência de caracteres terminada por um ponto '.', representando um texto, determinar a frequência relativa de vogais no texto (por exemplo, no texto "Em terra de cego quem tem um olho é caolho", essa frequência é 16/42).
2. Dada uma frase terminada por '.', imprimir o comprimento da palavra mais longa.
3. Dada uma sequência de caracteres terminada por '.', determinar quantas letras minúsculas e maiúsculas aparecem na sequência.
4. Dada uma frase terminada por '.', determinar quantas letras e quantas palavras aparecem no texto. Por exemplo, no texto "O voo GOL547 saiu com 10 passageiros." há 25 letras e 7 palavras.