

Char e Strings de Caracteres

Caracteres

Caracteres ocupam 1 byte (8 bits) na memória e são declarados com o tipo `char`.

Declaração:

```
/* variáveis simples */
char a, b;
char x = 'a';

/* vetores */
char y[12];
char z[26] = {"abcdefghijklmnopqrstuvwxy"};
/* são equivalentes */
char w[5] = {'a', 'e', 'i', 'o', 'u'};
char ww[] = {'a', 'e', 'i', 'o', 'u'};
char wl[5] = {"aeiou"};
char ww1[] = {"aeiou"}; char bb[10][20];

/* matrizes */
char mc[20][10];
/* cada nome fica numa linha da matriz */
char nomes[5][30] = {"maria", "jose", "antonio", "chico",
"tania"};
char nomes1[][30] = {"maria", "jose", "antonio", "chico",
"tania"};
```

Usos:

Uma variável do tipo `char` ocupa 1 byte (8 bits) e pode ser usada para armazenar um valor inteiro sem sinal (`unsigned char`) entre 0 e 255 ou um valor com sinal (`char`) entre -128 a 127. Pode ser usada como um número ou como caractere.

Exemplos:

```
char a, b[26];
:
:
/* uma constante do tipo char fica entre apóstrofes,
   não entre aspas */
a = '.';
/* branquear a vetor b usando a como contador */
for (a = 0; a < 26; a++) b[a] = ' ';
/* colocar em b as letras maiúsculas */
/* em ASCII 65=A, 66 = B, . . . , 90 = Z */
for (a = 0; a < 26; a++) b[a] = a + 65;
/* imprimir b com formato numérico e formato char */
for (a = 0; a < 26; a++)
    printf("b[%2d] = %2d e b[%2d] = %1c", a, b[a], a, b[a]);
```

Codificação ASCII (American Standard Code for Information Interchange)

Caracteres são armazenados internamente como números. Assim é necessária uma codificação, isto é, uma associação entre um caractere e um valor numérico. Existem várias tipos de codificação (EBCDIC, BCD, BAUDOT, etc . . .), mas a mais utilizada é o ASCII.

O ASCII é um código de 7 bits, portanto possui 128 possíveis combinações, que hoje em dia é a codificação básica usada em informática, e que admite várias extensões, principalmente para possibilitar a representação de caracteres especiais, caracteres gráficos e letras em alfabetos de outras línguas além do inglês (letras acentuadas por exemplo, alfabeto russo, japonês, etc . . .). As extensões estão associadas normalmente aos valores de 129 a 255, usando, portanto todos os 8 bits de um byte.

O programa abaixo imprime todos os caracteres, com os correspondentes valores decimal e hexadecimal desde 32 até 254. Antes de 32, estão alguns caracteres gráficos, bem como os caracteres especiais que tem alguma função especial quando enviados ou recebidos de dispositivos de entrada e saída como impressoras, teclados, vídeos, etc. Por exemplo:

CR (13) – carriage return (retorna para o início da linha corrente)
LF (10) – line feed (pula para a próxima linha)
FF (14) – form feed (pula para a próxima página)
BEL(07) – beep (aciona o dispositivo sonoro)
etc.

Veja no final destas notas uma tabela completa com os caracteres ASCII.

Veja um trecho da saída também abaixo.

```
#include <stdio.h>
/* imprime a tabela com todos os caracteres ascii
   do branco (32) até o último (255) */
int main() {
    unsigned char i;
    for (i = 32; i < 255; i++)
        printf("\ndecimal = %3d * hexadecimal = %2x * caracter =
%1c", i, i, i);
}
```

```
decimal = 32 * hexadecimal = 20 * caracter =
decimal = 33 * hexadecimal = 21 * caracter = !
decimal = 34 * hexadecimal = 22 * caracter = "
decimal = 35 * hexadecimal = 23 * caracter = #
decimal = 36 * hexadecimal = 24 * caracter = $
decimal = 37 * hexadecimal = 25 * caracter = %
decimal = 38 * hexadecimal = 26 * caracter = &
decimal = 39 * hexadecimal = 27 * caracter = '
decimal = 40 * hexadecimal = 28 * caracter = (
decimal = 41 * hexadecimal = 29 * caracter = )
decimal = 42 * hexadecimal = 2a * caracter = *
decimal = 43 * hexadecimal = 2b * caracter = +
decimal = 44 * hexadecimal = 2c * caracter = ,
decimal = 45 * hexadecimal = 2d * caracter = -
decimal = 46 * hexadecimal = 2e * caracter = .
decimal = 47 * hexadecimal = 2f * caracter = /
decimal = 48 * hexadecimal = 30 * caracter = 0
decimal = 49 * hexadecimal = 31 * caracter = 1
```

```
decimal = 50 * hexadecimal = 32 * caracter = 2
decimal = 51 * hexadecimal = 33 * caracter = 3
decimal = 52 * hexadecimal = 34 * caracter = 4
decimal = 53 * hexadecimal = 35 * caracter = 5
decimal = 54 * hexadecimal = 36 * caracter = 6
decimal = 55 * hexadecimal = 37 * caracter = 7
decimal = 56 * hexadecimal = 38 * caracter = 8
decimal = 57 * hexadecimal = 39 * caracter = 9
decimal = 58 * hexadecimal = 3a * caracter = :
decimal = 59 * hexadecimal = 3b * caracter = ;
decimal = 60 * hexadecimal = 3c * caracter = <
.....
.....
.....
```

Entrada e saída de caracteres

O formato %c é usado para ler ou imprimir caracteres. Na saída, se especificado um comprimento, por exemplo, %3c, são colocados brancos à esquerda.

O programa abaixo lê um vetor de 20 caracteres e imprime os caracteres lidos intercalando-os com um branco.

```
include <stdio.h>
int main() {
    char a[100];
    int i;
    for (i = 0; i < 20; i++)
        scanf("%c", &a[i]);

    for (i = 0; i < 20; i++)
        printf("%2c", a[i]);
}
```

Outra forma é o uso das funções getchar e putchar:

int getchar () – devolve o próximo caractere digitado.
void putchar (char x) – imprime o caractere x

Veja o exemplo abaixo que também lê e imprime uma seqüência de caracteres.

```
#include <stdio.h>
#define fim '#'
int main() {
    char c;
    /* Le um conjunto de caracteres ate encontrar '#'.
       Como o controle só volta ao programa após o enter, só imprime após o enter.
    */
    c = getchar();
    while (c != fim) {
        putchar(c);
        c = getchar();
    }

    /* outra forma */
    while ((c = getchar()) != fim) putchar (c);
}
```

Char e unsigned char

Conforme vimos nos exemplos acima, uma variável do tipo `char`, pode ser usada para armazenar um valor inteiro da mesma forma que uma variável `int` ou `short`. A diferença é que tem apenas 8 bits enquanto `short` tem 16 bits e `int` tem 32 bits.

Quando se declara `char` (ou `short` ou `int`), um bit é o bit de sinal na notação complemento de 2. Se não há necessidade do sinal, ou seja, apenas valores maiores ou iguais a zero, declara-se `unsigned char` (ou `unsigned short` ou `unsigned int`).

| Tipo | Valores |
|-----------------------------|-------------------------|
| <code>Char</code> | -128 a +127 |
| <code>unsigned char</code> | 0 a 255 |
| <code>Short</code> | -2^{15} a $+2^{15}-1$ |
| <code>unsigned short</code> | 0 a $2^{16}-1$ |
| <code>int</code> | -2^{31} a $+2^{31}-1$ |
| <code>unsigned int</code> | 0 a $2^{32}-1$ |

Quando usamos uma variável do tipo `char` para conter apenas caracteres, a configuração do caractere armazenado pode corresponder a um número negativo. Assim, o uso numérico de valores armazenados em variáveis do tipo `char` deve ser usado com cuidado.

Para evitar inconsistências, é conveniente sempre usar-se `unsigned char` quando se compara caracteres armazenados em variáveis do tipo `char`.

Cadeias de Caracteres ou Strings

A manipulação de seqüências ou cadeias de caracteres (strings) é uma operação muito comum em processamento não numérico. Imagine os programas editores de texto que precisam fazer operações do tipo:

- Procurar uma cadeia de caracteres no texto
- Eliminar uma cadeia do texto
- Substituir uma cadeia por outra
- Etc.

A manipulação de caracteres merece uma especial atenção das linguagens de programação e não é diferente em C.

Cadeias de caracteres são seqüências de caracteres terminadas pelo caractere especial zero binário, ou a constante `'\0'` em C. São armazenadas em vetores do tipo **`char`**.

Entrada e Saída de Cadeias de Caracteres

Com o comando `printf` e o formato `%s`. Veja o exemplo abaixo:

```
#include <stdio.h>
int main() {
    char a[100];
    /* le todos os caracteres digitados e coloca em a. Insere um \0 no final */
    scanf("%s", a);

    /* imprime todos os caracteres até o '\0' */
    printf("%s", a);
}
```

Com as funções gets e puts.

char *gets(char *s) - Lê os caracteres digitados e os coloca no vetor s até que seja digitado <enter>. O <enter> é descartado e é inserido um '\0' no final.

int puts(const char *s) - Imprime os caracteres do vetor s até encontrar um '\0'. Adiciona um caractere '\n' no final, ou seja muda de linha.

Veja o exemplo abaixo:

```
#include <stdio.h>
int main() {
    char a[100];
    /* le e imprime um string */
    gets (a);
    puts (a);
}
```

Algumas funções usando strings

```
/* move string a para b */
void move ( char a[], char b[]) {
    int i = 0;
    while (a[i] != '\0') {b[i] = a[i]; i++;}
    b[i] = a[i];
}
```

```
/* devolve o tamanho do string a */
int tamanho (char a[]) {
    int i = 0;
    while (a[i] != '\0') i++;
    return i
}
```

```
/* idem a move usando a função tamanho */
void move ( char a[], char b[]) {
    int i, k;
    k = tamanho (a);
    for (i = 0; i <= k; i++) b[i] = a[i];
}
```

```
/* outra versão da move */
void copia (char s[],char t[]) {
    int i = 0;
    while((s[i] = t[i]) != '\0') i++;
}
```

```
/* concatena 2 strings */
void concatena (char s[], char t[]) {
    int i = 0, j = 0;
    while (s[i] != '\0') i++;
    while((s[i++] = t[j++]) != '\0');
```

Comparação de strings:

A comparação entre dois strings é feita, levando em conta a codificação ASCII de cada caractere. Na codificação ASCII a ordem alfabética é compatível com a ordem numérica ('a' < 'b' < ... < 'z').

O mesmo ocorre para as letras maiúsculas e caracteres numéricos ('0' < '1' < ... < '9').

Na comparação de strings, como usual, o primeiro caractere diferente determina qual é o menor e qual é o maior.

"maria da silva" > "maria da selva"

"xico" > "francisco"

"maria" < "mariana" (tamanhos diferentes – o de maior comprimento é o maior)

"antonio dos santos" < "antonio santos"

```
/* compara dois strings a e b e devolve 0 se a=b, 1 se a>b e -1 se a<b */
int compara (unsigned char a[], unsigned char b[]) {
    int i, k;
    k = tamanho (a);
    for (i = 0; i < k+1; i++)
        if (a[i] != b[i])
            if (a[i] > b[i]) return 1;
            else return -1;
    /* se chegou aqui é porque todos eram iguais inclusive o (k+1)-ésimo
       que é o zero binário (fim do string)
       note que isto é verdade mesmo se tamanho(a) diferente do tamanho(b)
    */
    return 0;
}
```

```
/* idem, outra solução */
int compara (unsigned char a[], unsigned char b[]) {
    int i = 0;
    while (a[i] != 0 && b[i] != 0)
        if (a[i] != b[i])
            if (a[i] > b[i]) return 1;
            else return -1;
    /* Se chegou aqui é porque chegamos ao final de um dos strings
       (ou os dois)
       Em qualquer caso o i-esimo caracter decide se a==b, a>b ou a<b
    */
    if (a[i] == b[i]) return 0;
    else if (a[i] > b[i]) return 1;
    else return -1;
}
```

```
/* idem devolvendo 0 se a=b, valor>0 se a>b e valor<0 se a<b */
int compara (unsigned char a[], unsigned char b[]) {
    int i = 0;
    while (a[i] == b[i])
        if (a[i] == '\0') return 0;
        else i++;
    /* se chegou aqui é porque são diferentes */
    return a[i] - b [i];
}
```

Outra solução para a comparação:

```
int comparar (unsigned char s[], unsigned char t[]) {
    int i;
```

```
for(i=0;s[i]==t[i];i++) if(s[i]=='\0') return 0;
return s[i]-t[i];
}
```

Letras maiúsculas, minúsculas e vogais acentuadas

Como as letras maiúsculas são menores que as minúsculas, quando estas são misturadas numa string, a comparação fica confusa. Por exemplo:

"antonio" > "Antonio" e "antonio" > "anTonio"

Uma solução normalmente usada para evitar esse tipo de confusão é transformar todas as letras para maiúsculas antes de fazer a comparação.

```
/* transforma as minúsculas em maiúsculas da string a */
void tudo_maiuscula(char a[]) {
    int i = 0;
    while (a[i] != 0 )
        /* verifica se a[i] é minúscula */
        if (a[i] >= 'a' && a[i] <= 'z')
            /* transforma em maiúscula */
            a[i] = a[i] - 'a' + 'A';
}
```

Agora vamos refazer a compara usando a tudo_maiuscula:

```
int novacompara (char a[], char b[]) {
    int i = 0;
    /* transforma a e b */
    tudo_maiuscula(a);
    tudo_maiuscula(b);
    /* retorna o mesmo que compara (a, b) */
    return compara(a, b);
}
```

Outro problema na comparação alfabética ocorre com as vogais acentuadas: á, ã, â, é, ê, í, ó, õ, ô, ú. Cada uma delas tem o seu código numérico correspondente e a comparação pode ficar confusa. A melhor solução para uma comparação alfabética mais limpa e completa é:

- 1 - Trocar todas as vogais acentuadas pelas não acentuadas;
- 2 - Transformar para maiúsculas (ou minúsculas);
- 3 - Comparar.

Exercícios

1) Escreva uma função compacta (char a[], char b[]) que recebe o string a e devolve o string b eliminando os brancos de a.

- 2) Idem `compacta (char a[])` que devolve no próprio `a`. Faça sem usar vetores auxiliares, isto é, o algoritmo deve eliminar os brancos movendo os caracteres para novas posições do próprio `a`.
- 3) Idem `compacta (char a[])`, substituindo cadeias de mais de um branco, por um só branco.
- 4) Escreva a função `int contapal (char a[])`, que devolve como resultado o número de palavras do string `a`. Uma palavra é uma seqüência de caracteres não brancos, precedida e seguida por brancos.
- 5) Escreva a função `int substring (char a[], char b[])`, que verifica se o string `a` é sub-string de `b` devolvendo 1 (sim) ou 0 (não)

Funções pré-definidas com strings

Existem várias funções pré-definidas que mexem com cadeias de caracteres. Para usá-las é necessário um `#include <string.h>`. Veja algumas abaixo:

`char *strcpy(s, r)` – copia string `r` para a cadeia `s`. Devolve ponteiro para `s`.

`char *strncpy(s, r, n)` – copia no máximo `n` caracteres de `r` para `s`. Retorna ponteiro para `s`. Se `r` tiver menos que `n` caracteres preenche `s` com `'\0'`.

`char *strcat(s, r)` – concatena cadeia `r` no final da cadeia `s`. Retorna ponteiro para `s`.

`char *strncat(s, r, n)` – concatena no máximo `n` caracteres da cadeia `r` para a cadeia `s`. Termina `s` com `'\0'`. Retorna ponteiro para `s`.

`int strcmp(s, r)` – compara a cadeia `s` com a cadeia `r`. Retorna `<0` se `s < r`, `0` se `s = r` e `>0` se `s > r`.

`int strncmp(s, r, n)` – compara no máximo `n` caracteres da cadeia `s` com a cadeia `r`. Retorna `<0` se `s < r`, `0` se `s = r` e `>0` se `s > r`.

`char *strchr(s, c)` – retorna ponteiro para a primeira ocorrência do caractere `c` na cadeia `s` ou `NULL` se não estiver presente.

`char *strrchr(s, c)` – retorna ponteiro para a última ocorrência do caractere `c` na cadeia `s` ou `NULL` se não estiver presente.

`int strstrn(s, r)` – retorna tamanho do maior prefixo em `s` que coincide com `r`.

`int strcspn(s, r)` – retorna tamanho do maior prefixo em `s` que não coincide com `r`.

`char *strpbrk(s, r)` – retorna ponteiro para a primeira ocorrência na cadeia `s` de qualquer caractere na cadeia `r`, ou `NULL` se não achar.

`char *strstr(s, r)` – retorna ponteiro para a primeira ocorrência da cadeia `r` na cadeia `s`, ou `NULL` se não achar.

`int strlen(cs)` – retorna tamanho de `s`.

Embora pré-definidas, as funções acima tem uma implementação simples. Por exemplo:

`strlen(s)` é a função `tamanho(s)` definida acima.

`strncmp(s, r, n)` é a função `compara(s, r, n)` definida acima.

`strcpy(s, r)` é a função `move(s, r)` definida acima.

Quando se usa tais funções deve-se levar isso em conta se existe preocupação com a eficiência dos algoritmos. Veja no exemplo abaixo que conta o número de brancos de uma string:

```
c = 0;
for (i = 0; i < strlen(a); i++)
    if (a[i] == ' ') c++;
```

Melhor seria:

```
c = 0;
k = strlen(a);
for (i = 0; i < k; i++)
    if (a[i] == ' ') c++;
```

Funções de comparação de caracteres

O argumento das funções abaixo é sempre um **int**. Portanto podemos passar um **char** como parâmetro. As funções devolvem um valor diferente de zero se forem verdadeiras, ou zero se forem falsas.

`isupper (c)` – se `c` é letra maiúscula.

`islower (c)` – se `c` é letra minúscula.

`isalpha (c)` – equivalente a `isupper (c) || islower (c)`

`isdigit (c)` – caractere entre `'0'` e `'9'`.

`isalnum (c)` – equivalente a `isalpha (c) || isdigit (c)`

`isspace (c)` – se `c` é igual a branco.

Tabela de códigos ASCII

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|-----|----|-----|------------------------------------|-----|----|-----|-------|-------|-----|----|-----|-------|-----|-----|----|-----|--------|-----|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 | | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | \$ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | (| (| 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 |) |) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [| [| 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 |] |] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | | DEL |

Source: www.LookupTables.com

Tabela estendida de códigos ASCII

| | | | | | | | | | | | | | | | |
|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|
| 128 | Ç | 144 | É | 161 | í | 177 | ☒ | 193 | ⊥ | 209 | 〒 | 225 | β | 241 | ± |
| 129 | ü | 145 | æ | 162 | ó | 178 | ☓ | 194 | ⊤ | 210 | π | 226 | Γ | 242 | ≥ |
| 130 | é | 146 | Æ | 163 | ú | 179 | | 195 | ⊥ | 211 | ⊥ | 227 | π | 243 | ≤ |
| 131 | â | 147 | ô | 164 | ñ | 180 | † | 196 | — | 212 | ⊥ | 228 | Σ | 244 | ∫ |
| 132 | ä | 148 | ö | 165 | Ñ | 181 | ‡ | 197 | + | 213 | ⊥ | 229 | σ | 245 | ∫ |
| 133 | à | 149 | ò | 166 | • | 182 | ‡ | 198 | † | 214 | ⊥ | 230 | μ | 246 | ÷ |
| 134 | â | 150 | û | 167 | ° | 183 | ¶ | 199 | ‡ | 215 | ‡ | 231 | τ | 247 | ≈ |
| 135 | ç | 151 | ù | 168 | ¿ | 184 | ¶ | 200 | ⊥ | 216 | ‡ | 232 | Φ | 248 | ° |
| 136 | ê | 152 | — | 169 | — | 185 | ‡ | 201 | ⊥ | 217 | ∫ | 233 | ⊙ | 249 | . |
| 137 | ë | 153 | Ö | 170 | ¬ | 186 | | 202 | ⊥ | 218 | ∫ | 234 | Ω | 250 | . |
| 138 | è | 154 | Û | 171 | ½ | 187 | ¶ | 203 | 〒 | 219 | ■ | 235 | δ | 251 | √ |
| 139 | ì | 156 | £ | 172 | ¼ | 188 | ¶ | 204 | † | 220 | ■ | 236 | ∞ | 252 | — |
| 140 | î | 157 | ¥ | 173 | ı | 189 | ¶ | 205 | = | 221 | ■ | 237 | φ | 253 | z |
| 141 | ï | 158 | — | 174 | « | 190 | ¶ | 206 | ‡ | 222 | ■ | 238 | e | 254 | ■ |
| 142 | Ä | 159 | f | 175 | » | 191 | ¶ | 207 | ⊥ | 223 | ■ | 239 | ∩ | 255 | |
| 143 | Å | 160 | á | 176 | ☒ | 192 | L | 208 | ⊥ | 224 | α | 240 | ≡ | | |

Source: www.LookupTables.com