Tarjeta de referencia ANSI C

Estructura de programa/funciones

tipo func(*tipo*$_1$,...)	declaración de funciones
tipo nombre	declaración de variables globales
`main() {`	función principal
declaraciones	declaración de variables locales
instrucciones	
`}`	
tipo func(*arg*$_1$,...) {	definición de función
declaraciones	declaración de variables locales
instrucciones	
`return` *valor*;	
`}`	
`/* */`	comentarios
`main(int argc, char *argv[])`	programa con argumentos

Preprocesador de C

incluir fichero de cabeceras	`#include <`*fichero*`>`
incluir fichero de usuario	`#include "`*fichero*`"`
sustitución de texto	`#define` *nombre texto*
macro con argumentos	`#define` *nombre*(*var*) *texto*
Ejemplo. `#define max(A,B) ((A)>(B) ? (A) : (B))`	
anular definición	`#undef` *nombre*
entrecomillar al reemplazar	`#`
concatenar argumentos y reescanear	`##`
compilación condicional	`#if, #else, #elif, #endif`
¿*nombre* definido, no definido?	`#ifdef, #ifndef`
¿*nombre* definido?	`defined(`*nombre*`)`
carácter de continuación de línea	`\`

Tipos de datos. Declaraciones

carácter (1 byte)	`char`
entero	`int`
real (precisión simple)	`float`
real (precisión doble)	`double`
corto (entero de 16 bits)	`short`
largo (entero de 32 bits)	`long`
positivo y negativo	`signed`
sólo positivo	`unsigned`
puntero a `int`, `float`,...	`*int, *float,...`
enumeración	`enum`
valor constante (inalterable)	`const`
declaración de variable externa	`extern`
variable registro	`register`
variable estática	`static`
sin tipo	`void`
estructura	`struct`
crear un tipo de datos	`typedef` *tipo nombre*
talla de un objeto (devuelve un `size_t`)	`sizeof` *objeto*
talla de un tipo de datos (dev. un `size_t`)	`sizeof(`*tipo*`)`

Inicialización

Inicializar variable	*tipo nombre*`=`*valor*
Inicializar vector	*tipo nombre*`[]={`*valor*$_1$`,...}`
Inicializar cadena	`char` *nombre*`[]="`*cadena*`"`

© 2002 Joseph H. Silverman Permisos en el reverso. v1.3s

Constantes

largo (sufijo)	`L` o `l`
real de precisión simple (sufijo)	`F` o `f`
notación científica	`E` o `e`
octal (prefijo cero)	`0`
hexadecimal (prefijo cero-equis)	`0x` o `0X`
carácter constante (char, octal, hex.)	`'a', '\`*ooo*`', '\x`*hh*`'`
nueva línea, ret. de carro, tab., borrado	`\n, \r, \t, \b`
caracteres especiales	`\\, \?, \', \"`
cadena constante (termina con `'\0'`)	`"abc...de"`

Punteros, vectores y estructuras

declarar un puntero a *tipo*	*tipo* `*`*nombre*
decl. una func. que dev. un punt. a *tipo*	*tipo* `*f()`
decl. un punt. a func. que devuelve *tipo*	*tipo* `(*pf)()`
puntero genérico	`void *`
valor de puntero a nulo	`NULL`
objeto apuntado por *puntero*	`*`*puntero*
dirección del objeto *nombre*	`&`*nombre*
vector	*nombre*`[`*dim*`]`
vector multidimensional	*nombre*`[`*dim*$_1$`][`*dim*$_2$`]...`

Estructuras

`struct` *etiqueta* `{`	plantilla de estructura
declaraciones	declaración de campos
`};`	
crear estructura	`struct` *etiqueta nombre*
campo de estructura	*nombre*`.`*campo*
campo de estructura a través de puntero	*puntero*`->`*campo*
Ejemplo. `(*p).x` y `p->x` son lo mismo	
estructura múltiple, valor único	`union`
campo de bits con *b* bits	*campo* `:` *b*

Operadores (según precedencia)

acceso a campo de estructura	*nombre*`.`*campo*
acceso por puntero	*puntero*`->`*campo*
acceso a elemento de vector	*nombre*`[`*índice*`]`
incremento, decremento	`++, --`
más, menos, no lógico, negación bit a bit	`+, -, !, ~`
acceso por puntero, direcc. de objeto	`*`*puntero*`, &`*nombre*
convertir tipo de expresión	(*tipo*) *expr*
tamaño de un objeto	`sizeof`
producto, división, módulo (resto)	`*, /, %`
suma, resta	`+, -`
desplazamiento a izda., dcha. (bit a bit)	`<<, >>`
comparaciones	`>, >=, <, <=`
comparaciones	`==, !=`
"Y" bit a bit	`&`
"O exclusiva" bit a bit	`^`
"O" bit a bit	`|`
"Y" lógico	`&&`
"O" lógico	`||`
expresión condicional	*expr*$_1$ `?` *expr*$_2$ `:` *expr*$_3$
operadores de asignación	`=, +=, -=, *=, ...`
separador de evaluación de expresiones	`,`

Los operadores unarios, expresión condicional y operadores de asignación se agrupan de dcha. a izda.; todos los demás de izda. a dcha.

Control de flujo

finalizador de instrucción	`;`
delimitadores de bloque	`{ }`
salir de `switch`, `while`, `do`, `for`	`break`
siguiente iteración de `while`, `do`, `for`	`continue`
ir a	`goto` *etiqueta*
etiqueta	*etiqueta*`:`
valor de retorno de función	`return` *expr*

Construcciones de flujo

instrucción `if`	`if (`*expr*`)` *instrucción* `else if (`*expr*`)` *instrucción* `else` *instrucción*
instrucción `while`	`while (`*expr*`)` *instrucción*
instrucción `for`	`for (`*expr*$_1$`;` *expr*$_2$`;` *expr*$_3$`)` *instrucción*
instrucción `do`	`do` *instrucción* `while(`*expr*`);`

instrucción `switch`	`switch (`*expr*`) {`
	`case` *const*$_1$`:` *instrucción*$_1$ `break;`
	`case` *const*$_2$`:` *instrucción*$_2$ `break;`
	`default:` *instrucción*
	`}`

Bibliotecas ANSI estándar

`<assert.h>` `<ctype.h>` `<errno.h>` `<float.h>` `<limits.h>`
`<locale.h>` `<math.h>` `<setjmp.h>` `<signal.h>` `<stdarg.h>`
`<stddef.h>` `<stdio.h>` `<stdlib.h>` `<string.h>` `<time.h>`

Consulta de tipos de carácter `<ctype.h>`

`c` es un carácter

¿alfanumérico?	`isalnum(c)`
¿alfabético?	`isalpha(c)`
¿carácter de control?	`iscntrl(c)`
¿dígito decimal?	`isdigit(c)`
¿carácter imprimible (excluído espacio)?	`isgraph(c)`
¿letra minúscula?	`islower(c)`
¿carácter imprimible (incl. espacio)?	`isprint(c)`
¿car. impr. excepto espacio, letra, dígito?	`ispunct(c)`
¿separador?	`isspace(c)`
¿letra mayúscula?	`isupper(c)`
¿dígito hexadecimal?	`isxdigit(c)`
convertir a minúscula	`tolower(c)`
convertir a mayúscula	`toupper(c)`

Operaciones con cadenas `<string.h>`

`s`,`t` son cadenas, `cs`,`ct` son cadenas constantes

longitud de `s`	`strlen(s)`
copiar `ct` en `s`	`strcpy(s,ct)`
...hasta `n` caracteres	`strncpy(s,ct,n)`
concatenar `ct` tras `s`	`strcat(s,ct)`
...hasta `n` caracteres	`strncat(s,ct,n)`
comparar `cs` con `ct`	`strcmp(cs,ct)`
...sólo los primeros `n` caracteres	`strncmp(cs,ct,n)`
puntero al primer `c` en `cs`	`strchr(cs,c)`
puntero al último `c` en `cs`	`strrchr(cs,c)`
copiar `n` caracteres de `ct` en `s`	`memcpy(s,ct,n)`
copiar `n` cars. de `ct` en `s` (sobreescribe)	`memmove(s,ct,n)`
comparar `n` caracteres de `cs` con `ct`	`memcmp(cs,ct,n)`
punt. al 1er `c` en los `n` 1os cars. de `cs`	`memchr(cs,c,n)`
poner `c` en los `n` primeros cars. de `cs`	`memset(s,c,n)`

Tarjeta de referencia ANSI C

Entrada/Salida <stdio.h>

E/S estándar

flujo de entrada estándar	stdin
flujo de salida estándar	stdout
flujo de error estándar	stderr
final de fichero	EOF
obtener un carácter	getchar()
imprimir un carácter	putchar(*car*)
imprimir con formato	printf("*formato*",arg_1,...)
imprimir en cadena s	sprintf(s,"*formato*",arg_1,...)
leer con formato	scanf("*formato*",&$nombre_1$,...)
leer de cadena s	sscanf(s,"*formato*",&$nombre_1$,...)
leer línea en cadena s	gets(s)
imprimir cadena s	puts(s)

E/S de ficheros

declarar puntero a fichero	FILE *fp
obtener puntero a fichero	fopen("*nombre*","*mode*")
modos: r (leer), w (escribir), a (añadir)	
obtener un carácter	getc(*fp*)
escribir un carácter	putc(*car*,*fp*)
escribir en fichero	fprintf(*fp*,"*formato*",arg_1,...)
leer de fichero	fscanf(*fp*,"*formato*",arg_1,...)
cerrar fichero	fclose(*fp*)
distinto de cero si error	ferror(*fp*)
distinto de cero si EOF	feof(*fp*)
leer línea en cadena s (< max cars.)	fgets(s,max,*fp*)
escribir cadena s	fputs(s,*fp*)

Códigos de E/S con formato: "%-+ 0*w.pmc*"

-	alineación a izquierda
+	imprimir con signo
space	imprimir espacio si no hay signo
0	rellenar por delante con ceros
w	anchura mínima del campo
p	precisión
m	carácter de conversión: h short, l long, L long double
c	carácter de conversión:
d,i	entero
u	sin signo
c	carácter
s	cadena de caracteres
f	doble
e,E	exponencial
o	octal
x,X	hexadecimal
p	puntero
n	número de caracteres escritos
g,G	como f o e,E según cuál sea el exponente

Lista variable de argumentos <stdarg.h>

declarar puntero a argumentos	va_list *nombre*;
inicializar puntero a args.	va_start(*nombre*,*ultarg*)
ultarg es el último parámetro con nombre de la función	
siguiente arg. sin nom., actualizar punt.	va_arg(*nombre*,*tipo*)
invocar antes de salir de la función	va_end(*nombre*)

Funciones útiles <stdlib.h>

valor absoluto del entero n	abs(n)
valor absoluto del largo n	labs(n)
cociente y resto de enteros n,d	div(n,d)
devuelve una estructura con div_t.quot y div_t.rem	
cociente y resto de largos n,d	ldiv(n,d)
devuelve una estructura con ldiv_t.quot y ldiv_t.rem	
entero pseudo-aleatorio en [0,RAND_MAX]	rand()
fijar la semilla aleatoria a n	srand(n)
finalizar ejecución del programa	exit(estado)
ejecutar cadena s en el sistema	system(s)

Conversiones

convertir cadena s a double	atof(s)
convertir cadena s a int	atoi(s)
convertir cadena s a long	atol(s)
convertir prefijo de s a double	strtod(s,finp)
convertir prefijo de s (base b) a long	strtol(s,finp,b)
igual, pero unsigned long	strtoul(s,finp,b)

Reserva de memoria

reserva memoria	malloc(talla), calloc(nobj,talla)
cambiar tamaño de la reserva	realloc(pts,talla)
liberar memoria	free(ptr)

Funciones de vectores

buscar clave en vect	bsearch(clave,vect,n,talla,cmp())
ordenar vect ascendentemente	qsort(vect,n,talla,cmp())

Funciones de hora y fecha <time.h>

tiempo de proc. usado por el programa	clock()
Ejemplo. clock()/CLOCKS_PER_SEC da el tiempo en segundos	
segundos desde 1/1/1.970 (hora de ref.)	time()
tpo_2-tpo_1 en segs. (double)	difftime(tpo_2,tpo_1)
tipos numéricos para representar horas	clock_t,time_t
estructura estándar usada para fecha y hora	tm

tm_sec	segundos en el minuto
tm_min	minutos en la hora
tm_hour	horas desde medianoche
tm_mday	día del mes
tm_mon	meses desde enero
tm_year	años desde 1.900
tm_wday	días desde el domingo
tm_yday	días desde el 1 de enero
tm_isdst	indicador del cambio de horario (verano/invierno)

convertir hora local a hora de ref.	mktime(tp)
convertir hora en tp a cadena	asctime(tp)
convertir hora de ref. en tp a cadena	ctime(tp)
convertir hora de ref. a GMT	gmtime(tp)
convertir hora de ref. a hora local	localtime(tp)
formatear fecha y hora	strftime(s,smax,"*formato*",tp)
tp es un puntero a una estructura de tipo tm	

Funciones matemáticas <math.h>

los argumentos y valores devueltos son double

funciones trigonométricas	sin(x), cos(x), tan(x)
funciones trig. inversas	asin(x), acos(x), atan(x)
arctg(y/x)	atan2(y,x)
funciones trig. hiperbólicas	sinh(x), cosh(x), tanh(x)
exponenciales y logaritmos	exp(x), log(x), log10(x)
exps. y logs. (base 2)	ldexp(x,n), frexp(x,*e)
división y resto	modf(x,*ip), fmod(x,y)
potencia y raíz	pow(x,y), sqrt(x)
redondeo	ceil(x), floor(x), fabs(x)

Límites del tipo entero <limits.h>

límites típicos para un sistema Unix de 32 bits

CHAR_BIT	bits en char	(8)
CHAR_MAX	máximo valor de char	(127 o 255)
CHAR_MIN	mínimo valor de char	(-128 o 0)
INT_MAX	máximo valor de int	(+32767)
INT_MIN	mínimo valor de int	(-32768)
LONG_MAX	máximo valor de long	(+2147483647)
LONG_MIN	mínimo valor de long	(-2147483648)
SCHAR_MAX	máximo valor de signed char	(+127)
SCHAR_MIN	mínimo valor de signed char	(-128)
SHRT_MAX	máximo valor de short	(+32767)
SHRT_MIN	mínimo valor de short	(-32768)
UCHAR_MAX	máximo valor de unsigned char	(255)
UINT_MAX	máximo valor de unsigned int	(65535)
ULONG_MAX	máximo valor de unsigned long	(4294967295)
USHRT_MAX	máximo valor de unsigned short	(65536)

Límites del tipo real <float.h>

FLT_RADIX	dígitos del exponente	(2)
FLT_ROUNDS	modo de redondeo	
FLT_DIG	precisión (dígitos decimales)	(6)
FLT_EPSILON	menor x tal que $1.0 + x \neq 1.0$	(10^{-5})
FLT_MANT_DIG	dígitos de la mantisa	
FLT_MAX	máximo número en coma flotante	(10^{37})
FLT_MAX_EXP	exponente máximo	
FLT_MIN	mínimo número en coma flotante	(10^{-37})
FLT_MIN_EXP	mínimo exponente	
DBL_DIG	precisión de double (díg. decimales)	(10)
DBL_EPSILON	menor x t.q. $1.0 + x \neq 1.0$ (double)	(10^{-9})
DBL_MANT_DIG	díg. de la mantisa (double)	
DBL_MAX	máx. núm. en coma flot.(double)	(10^{37})
DBL_MAX_EXP	máximo exponente (double)	
DBL_MIN	mín. núm. en coma flot.(double)	(10^{-37})
DBL_MIN_EXP	mínimo exponente (double)	

Octubre 2002 v1.3s. Copyright © 2002 Joseph H. Silverman

La copia y distribución de esta tarjeta están permitidas siempre que el copyright y este permiso se mantengan en todas las copias.

Puede enviar comentarios y correcciones a J.H. Silverman, Math. Dept., Brown Univ., Providence, RI 02912 USA. ⟨jhs@math.brown.edu⟩

Traducido por F. Abad, C.D. Martínez, D. Picó, J.A. Sánchez

Ayuda para la programación en C

Estructura de un programa C

```
/*
Programa de Ejemplo
Fecha_
Autor_
*/
#include ____
#define ____
typedef ____
[Prototipos]

int main(void)
{
   [variables] /* descripcion */

   [instrucciones]
   return 0;
}
```

Caracteres especiales

`'\n'` cambio de línea (newline)
`'\r'` retorno de carro
`'\0'` caracter 0 (`NULL`)
`'\t'` TAB
`'\''` comilla simple '
`'\"'` comilla doble "
`'\\'` la barra \

Formatos de `printf` y `scanf`

`%d` int
`%hd` short
`%ld` long
`%u` unsigned int
`%hu` unsigned short
`%lu` unsigned long
`%f` float, double
`%lf` double (sólo `scanf`)
`%c` char
`%s` cadena de caracteres

Operadores

Aritméticos `int`: `+ - * / %`
Aritméticos `double`: `+ - * /`
Otros aritméticos: `++ -- += -= *= /=`
Lógicos y relacionales:
`> < >= <= == != && || !`

Bucles

Bucle `for`

```
for(inicialización, condición, instrucción_final)
    {
   [instrucciones]
}
```

Ejemplo: `for(i=0; i<10; i++)`

Bucle `while`

```
while (condición) {
   [instrucciones]
}
```

Bucle `do-while`

```
do  {
   [instrucciones]
} while(condición);
```

Bloque if

caso 1:

```
if (condición)  {
   [instrucciones]
}
```

caso 2:

```
if (condición)  {
   [instrucciones_1]
} else  {
   [instrucciones_2]
}
```

caso 3:

```
if (condición_1)  {
   [instrucciones_1]
} else if (condición_2) {
   [instrucciones_2]
...
} else if (condición_n) {
   [instrucciones_n]
} else  {
   [instrucciones]
}
```

Sintaxis del switch

```
switch(expresión_entera)  {
case constante_1:
   [instrucciones_1]
   break;
case constante_2:
   [instrucciones_2]
   break;
...
case constante_3:
   [instrucciones_3]
   break;
default:
   [instrucciones]
}
```

Vectores y matrices

```
double vector[10];
char cadena[256];
char matriz[10][20];

vector[2]=3;
scanf("%lf",&vector[7]);
```

Cadenas de caracteres

```
char cadena[N];
```

Lectura:

`scanf("%s",cadena);`
lee una palabra

`gets(cadena);`
lee una frase hasta fin de linea

`fgets(cadena, N, stdin);`
lee una frase con control de tamaño. También lee `\n`

Escritura:

`printf("%s",cadena);`
escribe una cadena por pantalla, vale para frase o palabra

Funciones estandar de string.h

`size_t strlen( char *str );`
devuelve la longitud de la cadena

`strcpy( char *to, char *from );`
copia o inicializa

`int strcmp(char *s1, char *s2 );`
compara las cadenas `s1` y `s2`
`0` → `s1` es igual a `s2`
`<0` → `s1` es menor que `s2`
`>0` → `s1` es mayor que `s2`

Funciones

Prototipo:
```
tipo NombreFun(tipo var1, ... , tipo varN);
```

Estructura de la función:
```
tipo NombreFun(tipo var1, ... , tipo varN)
/* Descripción general
Argumentos: ...
Valor Retornado: ...
Advertencias de uso: ...
*/
{
   [variables locales]

   [instrucciones]

   return expresión;
}
```

Ejemplos de prototipos y llamadas:
```
int Sumar(int a, int b);
void Cambio(int *a, int *b);
double CalcularMedia(double a[], int n);
float Traza(float mat[][20], int n, int m);

res=Sumar(x,y);
Cambio(&x, &y);
med=CalcularMedia(vec,n);
tra=Traza(mat,n,m);
```

Asignación Dinámica de Memoria
```
char *pc;
pc=(char *)calloc(100, sizeof(char);
pc=(char *)malloc(100*sizeof(char));
pc=(char *)realloc(pc, 200*sizeof(char));
free(pc);  /*libera memoria */
```
Estas funciones devuelven NULL en caso de error

Estructuras

Declaración de un tipo estructura
```
typedef struct persona {
   char nombre[N];
   int edad;
   long dni;
} PERSONA;
```

Declaración de variables:
```
PERSONA p;   /* una estructura */
PERSONA *pp; /* puntero a estructuras */
PERSONA vec[20]; /* vector de estructuras */
```

Acceso a los miembros:
```
p.edad=27;
pp->edad=30;
vec[7].edad=37;
```

Declaración de listas enlazadas:
```
typedef struct lista {
   char nombre[N];
   int edad;
   long dni;
   struct lista *siguiente;
} LISTA;
```

Archivos

Abrir y cerrar
```
FILE *fopen(char *nombre, char *modo);
```
Devuelve NULL en caso de error
modo="r" Lectura
modo="r+" Lectura (y escritura)
modo="w" Escritura
modo="w+" Escritura (y lectura)
modo="a" Añadir al final
modo="a+" Añadir al final (y lectura)
modos=rb, rb+, wb, wb+, ab, ab+ binario
```
int fclose(FILE *puntero al archivo);
```
Devuelve 0 si no hay error

Archivos de texto
```
int fscanf(FILE *puntero_archivo, char *cadena_formato, ...);
```
Devuelve el número de variables leídas
Devuelve 0 si no hay podido leer ninguna variable
Devuelve EOF si ha llegado al final de fichero
```
int fprintf(FILE *puntero_archivo, char *cadena_formato, ...);
char *fgets(char *cadena, int tam_cad, FILE *puntero_archivo);
```
Devuelve el puntero a la cadena si no hay error
Devuelve NULL en caso de error
```
int fputs(char *cadena, FILE *puntero_archivo);
```

Archivos binarios (acceso directo)
```
int fwrite(void *buffer, size_t size, size_t num, FILE *stream);
int fread(void *buffer, size_t size, size_t num, FILE *stream);
```
Devuelve el número de elementos leídos, normalmente `num`

```
int fseek(FILE *stream, long offset, int origin);
```
El tercer argumento puede tomar los valores: `SEEK_SET` (comienzo), `SEEK_END` (final), `SEEK_CUR` (actual)

Otras Funciones generales
```
int fgetc(FILE *puntero_archivo);
```
Devuelve el caracter leído (lo devuelve como `int`)
Devuelve EOF si ha llegado al final de fichero

```
int fputc(int caracter, FILE *puntero_archivo);
int feof( FILE *stream );
```
Devuelve distinto de cero si estamos al final del fichero. En caso contrario, devuelve cero

```
void rewind( FILE *stream );
```
Vuelve al principio del archivo. Equivale a `fseek(fp,0,SEEK_SET);`

Tabla 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	&#127;	DEL

Source: www.asciitable.com

ASCII Extendido

128	Ç	144	É	160	á	176	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	161	í	177	▒	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	162	ó	178	▓	195	├	211	╙	227	π	243	≤
131	â	147	ô	163	ú	179	|	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	164	ñ	180	┤	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	165	Ñ	181	╡	198	╞	214	╓	230	µ	246	÷
134	å	150	û	166	ª	182	╢	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	167	º	183	╖	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	168	¿	184	╕	201	╔	217	┘	233	⊕	249	·
137	ë	153	Ö	169	_	185	╣	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	170	¬	186	║	203	╦	219	▉	235	δ	251	√
139	ï	156	£	171	½	187	╗	204	╠	220	▄	236	∞	252	_
140	î	157	¥	172	¼	188	╝	205	═	221	▌	237	ϕ	253	²
141	ì	158	_	173	¡	189	╜	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	174	«	190	╛	207	╧	223	▀	239	∩	255	
143	Å	192	└	175	»	191	┐	208	╨	224	α	240	≡		

Source: www.asciitable.com